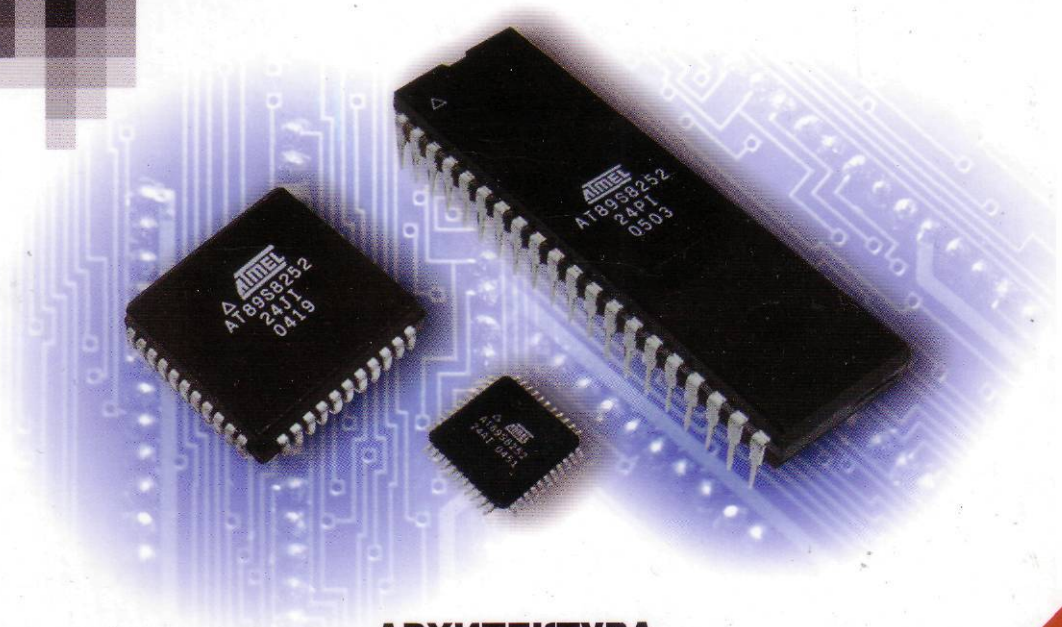
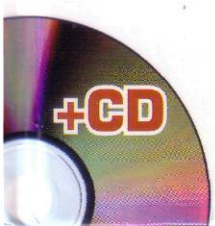
ПРАКТИКА
ИНЖЕНЕРНОЙ ЭЛЕКТРОНИКИ

Вольфганг Трамперт

AVR-RISC Микроконтроллеры



АРХИТЕКТУРА
АППАРАТНЫЕ РЕСУРСЫ
СИСТЕМА КОМАНД
ПРОГРАММИРОВАНИЕ
ПРИМЕНЕНИЕ



МК-ПРЕСС

WWW.MK-PRESS.COM

AVR-RISC Mikrocontroller

Architektur, Hardware-Ressourcen, Befehlsvorrat,
Programmierung, Applikationen

Wolfgang Trampert

Scanned by Grey

Franzis' Verlag GmbH

Вольфганг Трамперт

AVR-RISC микроконтроллеры

АРХИТЕКТУРА, АППАРАТНЫЕ РЕСУРСЫ, СИСТЕМА КОМАНД, ПРОГРАММИРОВАНИЕ, ПРИМЕНЕНИЕ

Перевод с немецкого:
В. П. Репало, В. И. Кириченко, Ю. А. Шпак

"МК-Пресс"
Киев, 2006

ББК 32.973-04
Т 65
УДК 004.312

Трамперт В.

Т65 AVR-RISC микроконтроллеры.: Пер. с нем. — К.: "МК-Пресс", 2006. — 464 с., ил.

ISBN 966–8806-07-7 (рус.)

В книге дано исчерпывающее описание базовой серии микроконтроллеров семейства AVR от компании Atmel, построенных на базе прогрессивной архитектуры RISC с применением программируемой флэш-памяти EPROM. Кроме того, подробно рассматривается программирование микроконтроллеров данной серии на языке ассемблера, а также среда отладки AVR-Studio и программно-аппаратный набор STK200.

Книга предназначена для всех, кто уже обладает основными познаниями в области построения и функционирования микрокомпьютеров, желает изучить однокристальные микроконтроллеры AVR и успешно претворять в жизнь задачи внутриплатного управления.

ББК 32.973-04

Главный редактор: Ю. А. Шпак

Подписано в печать 26.12.2005. Формат 70 × 100 1/16.
Бумага офсетная. Печать офсетная. Усл. печ. л. 37,6. Уч.-изд. л. 25,3.
Тираж 3000 экз. Заказ 5-2384.

ЧП Савченко Л.А., Украина, г.Киев, тел./ф.: (044) 517-73-77; e-mail: info@mk-press.com.
Свидетельство о внесении субъекта издательского дела в Государственный реестр издателей, производителей и распространителей издательской продукции: серия ДК №51582 от 28.11.2003г.

Отпечатано в ЗАО "ВИПОЛ". 03151, г.Киев, ул. Волынская, 60

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Franzis' Verlag GmbH.

Authorized translation from the German language edition published by Franzis', Copyright © 2003.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

Russian language edition published by MK-Press according to the Agreement with Franzis' Verlag GmbH, Copyright © 2006.

ISBN 966–8806-07-7 (рус.)
ISBN 3-7723-5476-9 (нем.)

© "МК-Пресс", 2006
© Franzis' Verlag GmbH, 2003

Краткое оглавление

Содержание 6
Предисловие 17
Условные обозначения 19
1. Введение 20
2. Обзор 24
3. Центральный процессор и внутренняя память 41
4. Таймеры/счетчики микроконтроллеров базовой серии семейства AVR 100
5. Сторожевой таймер 120
6. Асинхронная передача данных через приемопередатчик UART 123
7. Синхронная передача данных через последовательный интерфейс (SPI) 139
8. Последовательная передача данных по шине I^2C 151
9. Интегрированный аналоговый компаратор 166
10. Порты ввода/вывода 169
11. Программирование памяти 179
12. Система команд 199
13. Ассемблер 275
14. Отладка программ в среде AVR-Studio 298
15. Набор STK200 для тестирования и записи в память собственных программ 314
16. Применение микроконтроллеров семейства AVR 327
17. Приложение 443
18. Содержимое прилагаемого к книге компакт-диска 445
Internet-ссылки 450
Предметный указатель 451

Содержание

ПРЕДИСЛОВИЕ **17**

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ **19**

1. ВВЕДЕНИЕ **20**
Архитектура RISC 20
Представители базовой серии AVR 23

2. ОБЗОР **24**
Основные характеристики семейства микроконтроллеров AVR 24
Арифметико-логическое устройство 25
Структура команд 25
Время выполнения команды 25
Архитектура памяти 25
Область ввода/вывода 26
Прерывания и подпрограммы *26*
Периферийные функции *27*
Блок-схема микроконтроллеров AT90S1200 и AT90S8515 28
Конструктивное исполнение корпусов и расположение выводов 28
Генерирование такта системной синхронизации в микроконтроллерах AVR 33
Интегрированный кварцевый осциллятор базовой серии микроконтроллеров AVR *33*
Генерирование такта системной синхронизации с помощью контура RC-осциллятора *34*
Программирование для AVR на языке высокого уровня C 35
Стендовые испытания для сравнения микроконтроллеров AVR с главными конкурентами 36
Результаты *39*

3. ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР И ВНУТРЕННЯЯ ПАМЯТЬ **41**
Система управления и АЛУ 41
Статическая память RAM (SRAM) 41
Организация статической памяти SRAM семейства микроконтроллеров AVR 43
Регистровый файл *44*
Регистры двойной длины X, Y и Z 44
Область ввода/вывода *45*
Регистр состояния (SREG) 48
Регистр управления MCU (MCUCR) 49
Внутренняя память SRAM *51*
Внешняя память SRAM *52*
Стек базовой серии микроконтроллеров AVR *54*
Указатель стека 55

ПАМЯТЬ КОМАНД (ТЕХНОЛОГИЯ FLASH-EPROM) ... 59
Физическая организация флэш-памяти базовой серии микроконтроллеров семейства AVR ... 60
Технологии памяти Flash-EPROM ... 60
Процесс программирования ... 61
Процесс стирания ... 63
ПАМЯТЬ ДЛЯ ЭНЕРГОНЕЗАВИСИМЫХ ДАННЫХ (ТЕХНОЛОГИЯ EEPROM) ... 64
Память данных типа EEPROM ... 64
Технология памяти EEPROM ... 64
Доступ ЦП к памяти EEPROM на запись/чтение ... 65
Регистр адреса EEAR памяти EEPROM ... 66
Регистр данных EEDR памяти EEPROM ... 66
Регистр управления EECR памяти EEPROM ... 66
Чтение памяти EEPROM (для всех типов микроконтроллеров базовой серии семейства AVR) ... 67
Запись в память EEPROM в микроконтроллерах AT90S2313, AT90S4414 и AT90S8515 ... 67
Запись в память EEPROM в микроконтроллере AT90S1200 ... 70
РАЗЛИЧНЫЕ СПОСОБЫ АДРЕСАЦИИ КОМАНД И ДАННЫХ ... 72
Прямая адресация одного регистра ... 72
Прямая адресация двух регистров Rd и Rr ... 73
Прямая адресация области ввода/вывода ... 73
Прямая адресация памяти данных (SRAM) ... 74
Косвенная адресация памяти данных (SRAM) ... 75
Косвенная адресация регистров в микроконтроллере AT90S1200 ... 76
Косвенная адресация памяти данных с последующим приращением адреса ... 77
Косвенная адресация памяти данных с предварительным уменьшением адреса ... 77
Относительная адресация памяти данных ... 78
Адресация констант в памяти программ ... 79
Прямая адресация памяти программ (команды `jmp` и `call`) ... 80
Косвенная адресация памяти программ (команды `ijmp` и `icall`) ... 81
Относительная адресация памяти программ (команды `rjmp` и `rcall`) ... 82
Время доступа к памяти и время выполнения команд ... 83
СБРОС И ОБРАБОТКА ПРЕРЫВАНИЙ ... 84
Сброс по включению питания ... 87
Внешний сброс ... 90
Сброс от сторожевого таймера ... 91
Обработка прерываний ... 91
Время ответа на прерывание ... 92
Внешние прерывания ... 92
Регистр GIMSK ... 93
Регистр GIFR ... 93
Регистр TIMSK ... 94
Регистр TIFR ... 96

"Спящие" режимы центрального процессора 97

4. Таймеры/счетчики микроконтроллеров базовой серии семейства AVR 100

Предварительный делитель частоты и схема управления таймером 100

Восьмиразрядный таймер/счетчик T/C0 102

Регистр управления TCCR0 таймера/счетчика T/C0 104

Счетный регистр TCNT0 таймера/счетчика T/C0 104

Области применения таймера/счетчика T/C0 104

Шестнадцатиразрядный таймер/счетчик T/C1 104

Счетный регистр TCNT1 107

Регистр управления TCCR1A 108

Регистр управления TCCR1B 109

Регистр захвата на входе ICR1 113

Регистры сравнения на выходе OCR1A и OCR1B 113

Таймер/счетчик T/C1 как широтно-импульсный модулятор 114

Области применения таймера/счетчика T/C1 118

5. Сторожевой таймер 120

Регистр управления WDTCR 121

6. Асинхронная передача данных через приемопередатчик UART 123

Распространенные стандарты асинхронной передачи данных 123

Формат передачи по асинхронному интерфейсу 127

Физическое устройство приемопередатчика UART 129

Передающий элемент UART (трансмиттер) 130

Принимающий элемент приемопередатчика UART (ресивер) 131

Регистры UART 134

Регистр ввода/вывода данных UDR 134

Регистр состояния USR 134

Регистр управления UCR 136

Регистр скорости передачи данных UBRR 137

7. Синхронная передача данных через последовательный интерфейс (SPI) 139

Входы и выходы интерфейса SPI 141

Протокол передачи 144

Системные конфликты SPI 147

Регистр управления SPCR 148

Регистр состояния SPSR 149

Регистр данных SPDR 149

8. Последовательная передача данных по шине I^2C 151

Принцип действия шины I^2C 152

Линия SCL 152

Линия SDA 153

Режимы работы блоков, подсоединенных с помощью шины I^2C 154

Электрические свойства ... 155
Протокол шины ... 155
Занятость шины ... *155*
Условие начала передачи ... *156*
Процесс передачи данных ... *156*
Бит подтверждения ... 156
Освобождение шины передачи данных ... *157*
Условие завершения передачи ... 158
Адресация ведомых устройств ... 158
Адрес ведомого устройства ... *158*
Постоянная часть адреса ведомого устройства ... 159
Переменная часть адреса ведомого устройства ... 159
Разряд направления передачи данных ... 159
Особые случаи ... 161
Повторение условия начала передачи ... *161*
Чтение–модификация–запись ... *161*
Задержка такта ... *162*
Синхронизация такта ... *162*
Арбитраж шины при работе с несколькими ведущими устройствами. 164
Синхронизация шины I^2C ... 165
Обращение микроконтроллеров семейства AVR к шине I^2C ... 165

9. ИНТЕГРИРОВАННЫЙ АНАЛОГОВЫЙ КОМПАРАТОР ... 166
Регистр управления и состояния ACSR ... 167

10. ПОРТЫ ВВОДА/ВЫВОДА ... 169
Порт A ... 172
Регистр данных порта A ... *172*
Регистр направления передачи данных DDRA ... *172*
Адреса порта A — выводы PINA ... *172*
Порт B ... 173
Регистр данных порта B ... *173*
Регистр направления передачи данных DDRB ... *175*
Адреса порта B — выводы PINB ... *175*
Порт C ... 175
Регистр данных порта C ... *175*
Регистр направления передачи данных DDRC ... *176*
Адреса порта C — выводы PINC ... *176*
Порт D ... 176
Регистр данных порта D ... *176*
Регистр направления передачи данных DDRD ... *177*
Адреса порта D — выводы PIND ... *178*
Выход с открытым коллектором ... 178

11. ПРОГРАММИРОВАНИЕ ПАМЯТИ ... 179
Разряды блокировки памяти программ LB1 и LB2 ... 179
Разряды предохранения RCEN, FSTRT и SPIEN ... 179

Байты сигнатуры ... 180
Процесс программирования ... 180
Параллельный режим программирования ... *181*
Считывание запрограммированных данных из флэш-памяти ... 187
Программирование памяти EEPROM ... 188
Чтение памяти EEPROM ... 188
Программирование разрядов предохранения ... 190
Программирование разрядов блокировки ... 190
Считывание разрядов предохранения и блокировки ... 190
Считывание байтов сигнатурны ... 191
Параметры параллельного режима программирования ... 191
Последовательный режим программирования ... *192*
Алгоритм последовательного программирования ... 194
Опрос для установления окончания процесса программирования ... 197
Параметры последовательного режима программирования ... 197

12. СИСТЕМА КОМАНД ... 199
Разряды условий (флаги) микроконтроллеров AVR ... 200
Разряд 0 — C (флаг переноса) ... *201*
Разряд 1 — Z (нулевой флаг) ... *203*
Разряд 2 — N (флаг отрицательного результата) ... *203*
Разряд 3 — V (переполнение при вычислениях в дополнительных кодах) ... *203*
Разряд 4 — S (флаг знака) ... *204*
Разряд 5 — H (флаг половинного переноса) ... *204*
Разряд 6 — T (флаг копирования) ... *205*
Разряд 7 — I (общее разрешение прерываний) ... *205*
Условный переход в программе ... *205*
Обзор команд микроконтроллеров AVR ... 206
Описание команд микроконтроллеров AVR ... 222
ADC ... *222*
ADD ... *223*
ADIW ... *223*
AND ... *224*
ANDI ... *224*
ASR ... *225*
BCLR ... *225*
BLD ... *226*
BRBC ... *226*
BRBS ... *227*
BRCC ... *227*
BRCS ... *228*
BREQ ... *228*
BRGE ... *229*
BRHC ... *229*
BRHS ... *230*
BRID ... *230*
BRIE ... *231*

BRLO 231
BRLT 232
BRMI 232
BRNE 233
BRPL 233
BRSH 234
BRTC 234
BRTS 235
BRVC 235
BRVS 236
BSET 236
BST 237
CALL 237
CBI 238
CBR 238
CLC 239
CLH 239
CLI 239
CLN 240
CLR 240
CLS 240
CLT 241
CLV 241
CLZ 241
COM 242
CP 242
CPC 243
CPI 243
CPSE 244
DEC 245
EOR 245
ICALL 246
IJMP 246
IN 247
INC 247
JMP 248
LD (LDD) 248
Косвенная загрузка из SRAM в регистр с помощью указателя X 248
Косвенная загрузка из SRAM в регистр с помощью указателя Y 249
Косвенная загрузка из SRAM в регистр с помощью указателя Z 250
LDI 251
LDS 251
LPM 252
LSL 252
LSR 253
MOV 253

MUL ... 254
NEG ... 254
NOP ... 255
OR ... 255
ORI ... 256
OUT ... 256
POP ... 257
PUSH ... 257
RCALL ... 258
RET ... 258
RETI ... 259
RJMP ... 259
ROL ... 260
ROR ... 260
SBC ... 261
SBCI ... 261
SBI ... 262
SBIC ... 262
SBIS ... 263
SBIW ... 263
SBR ... 264
SBRC ... 264
SBRS ... 265
SEC ... 265
SEH ... 266
SEI ... 266
SEN ... 266
SER ... 267
SES ... 267
SET ... 267
SEV ... 268
SEZ ... 268
SLEEP ... 268
ST (STD) ... 269
Косвенное сохранение содержимого регистра в SRAM с помощью указателя X ..269
Косвенное сохранение содержимого регистра в SRAM с помощью указателя Y ..270
Косвенное сохранение содержимого регистра в SRAM с помощью указателя Z...271
STS ... 272
SUB ... 272
SUBI ... 273
SWAP ... 273
TST ... 274
WDR ... 274

13. АССЕМБЛЕР 275
Установка AVR-ассемблера 279
Синтаксис ассемблера 280
Работа с *AVR*-ассемблером 281
Кнопки панели инструментов 281
Меню AVR-ассемблера WAVRASM 282
Меню Window 283
Поиск ошибок 283
Директивы ассемблера 284
Директива .BYTE 285
Директива .CSEG 285
Директива .DB 286
Директива .DEF 286
Директива .DEVICE 287
Директива .DSEG 287
Директива .DW 288
Директива .ENDMACRO 288
Директива .EQU 288
Директива .ESEG 289
Директива .EXIT 289
Директива .INCLUDE 289
Директива .LIST 290
Директива .LISTMAC 290
Директива .MACRO 291
Директива .NOLIST 291
Директива .ORG 291
Директива .SET 292
Выражения 293
Операнды 293
Функции 293
Операторы 294

14. ОТЛАДКА ПРОГРАММ В СРЕДЕ AVR-STUDIO 298
Установка AVR-Studio 298
Обзор 299
Окна AVR-Studio 300
Окно исходного кода 300
Окно Registers 301
Окно Processor 302
Окна Memory 303
Окно IO 305
Свойства окна IO 307
Окно Watch 307
Окно Trace 307
Окно Message 308

Меню AVR-Studio 308
Меню File *308*
Меню правки Edit *309*
Меню Debug для управления ходом программы *309*
Меню Breakpoint *310*
Меню Trace & Triggers *310*
Меню Watch *310*
Меню Options *311*
Меню Views *311*
Меню Window и Help *312*
Модули ввода/вывода симулятора 312

15. НАБОР STK200 ДЛЯ ТЕСТИРОВАНИЯ И ЗАПИСИ В ПАМЯТЬ СОБСТВЕННЫХ ПРОГРАММ 314
Описание аппаратной части STK200 315
Организация питания STK200 ..... *316*
Обнаружение провалов напряжения в STK200 ..... *316*
Функции перемычек STK200 ..... *317*
Подключения портов STK200 ..... *318*
Интерфейс ISP STK200 ..... *319*
RS232-интерфейс STK200 ..... *319*
ЖКИ-интерфейс STK200 ..... *320*
Встроенный кварцевый осциллятор STK200 ..... *322*
Восемь индикаторных светодиодов на STK200 ..... *322*
Восемь кнопочных выключателей на STK200 ..... *323*
Расширение памяти в STK200 ..... *323*
Программное обеспечение для STK200 323
Меню программного обеспечения ISP *324*
Панель инструментов среды ISP *326*

16. ПРИМЕНЕНИЕ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА AVR 327
Средства двоично-десятичной арифметики 327
Преобразование шестнадцатеричного числа в BCD-число *332*
Преобразование простой двоичной дроби в BCD-дробь *332*
Преобразование BCD-числа в шестнадцатеричное число *332*
Сложение BCD-чисел *333*
Вычитание BCD-чисел *334*
Базовые операции ввода/вывода 334
Подключение ЖК-модулей 341
Подключение модуля HD44780 ..... *341*
Регистры модуля HD44780 ..... *342*
Память модуля HD44780 ..... *343*
Знакогенератор модуля HD44780 ..... *344*
Разработка собственных символов *344*
Сброс табло *345*
Система команд HD44780 ..... *347*

Пример для подключения двухстрочного табло по 16 знаков в строке. 350
Описание подпрограмм 356
Описание главной программы 357
ФОРМИРОВАНИЕ ИМПУЛЬСОВ ОПРЕДЕЛЕННОЙ ДЛИНЫ С ПОМОЩЬЮ T/C0... 357
Описание программы 358
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АВТОМАТИЧЕСКОЙ ПЕРЕЗАГРУЗКИ T/C0 360
Описание подпрограммы обработки прерывания 362
Время выполнения подпрограммы обработки прерывания таймера 0.. 363
Описание главной программы 363
ВЫРАБОТКА С ПОМОЩЬЮ T/C1 ИМПУЛЬСОВ С ЧАСТОТОЙ 50 ГЦ И КОЭФФИЦИЕНТОМ ЗАПОЛНЕНИЯ 0,025 364
Описание подпрограммы обработки прерывания 367
Описание главной программы 367
ТРЕХКАНАЛЬНЫЙ ЦАП С РАЗРЕШЕНИЕМ 10 РАЗРЯДОВ 369
Описание подпрограммы обработки прерывания от T/C0 378
Время выполнения подпрограммы обработки прерывания от таймера 0. 379
Описание главной программы 380
Измерения 380
ЧЕТЫРЕХКАНАЛЬНЫЙ АЦП С ДВОЙНЫМ ИНТЕГРИРОВАНИЕМ И РАЗРЕШЕНИЕМ 11 РАЗРЯДОВ 381
Описание подпрограмм 388
Описание главной программы 388
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЕМОПЕРЕДАТЧИКА UART ДЛЯ МИКРОКОНТРОЛЛЕРА AT90S1200 392
ПОДКЛЮЧЕНИЕ К МИКРОКОНТРОЛЛЕРУ AT90S8515 МИКРОСХЕМЫ ЦАП MAX5154 ЧЕРЕЗ ИНТЕРФЕЙС SPI 399
РАСШИРЕНИЕ ПОРТОВ ВВОДА/ВЫВОДА МИКРОКОНТРОЛЛЕРА AT90S4414 С ПОМОЩЬЮ ИНТЕРФЕЙСА SPI 407
Описание подпрограммы 411
Описание главной программы 412
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИНТЕРФЕЙСА SPI ДЛЯ МОДЕЛИ AT90S1200 ПРИ ПОДКЛЮЧЕНИИ АЦП 413
Описание программы 418
ПОДКЛЮЧЕНИЕ К ШИНЕ I^2C ДАТЧИКА ТЕМПЕРАТУРЫ LM75 419
ПОДКЛЮЧЕНИЕ К ШИНЕ I^2C СХЕМЫ SAA1064 УПРАВЛЕНИЯ ЧЕТЫРЕХРАЗРЯДНЫМ СВЕТОДИОДНЫМ ТАБЛО 424
ИСПОЛЬЗОВАНИЕ МИКРОКОНТРОЛЛЕРА AVR В КАЧЕСТВЕ ВЕДУЩЕГО УСТРОЙСТВА I^2C 429
Описание подпрограмм 438
Описание главной программы 441

17. ПРИЛОЖЕНИЕ 443
ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА AVR 443
ДОКУМЕНТИРОВАННЫЕ КОМПАНИЕЙ ATMEL ОШИБКИ И ПУТИ ИХ УСТРАНЕНИЯ 443

18. СОДЕРЖИМОЕ ПРИЛАГАЕМОГО К КНИГЕ КОМПАКТ-ДИСКА 445
ПАПКА DATEN 445
ПАПКА ATMEL 446
Подпапка Sheets *446*
Подпапка Applicat *446*
Подпапка Software *447*
Подпапка Tools *448*
ПАПКА PROGRAMM 449
INTERNET-ССЫЛКИ 450
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 451

Предисловие

Со времени выхода на рынок первого широко применяющегося в промышленности микропроцессора 8080 компании Intel развитие последующих поколений пошло по двум главным направлениям. С одной стороны находятся такие высокопроизводительные процессоры как Pentium компании Intel или семейство процессоров 680x0 компании Motorola, которые находят применение в персональных компьютерах, рабочих станциях и научных вычислительных установках, а также реализуются в архитектуре компьютеров со сложным набором команд CISC (CISC — Complex Instruction Set Computer). С другой стороны быстро обозначилась большая потребность в более простых микроконтроллерах, для которых компактность важнее способности обработать огромный поток команд.

Компания Intel еще в 1976 году вывела на рынок однокристальный микроконтроллер 8048, у которого в одном кристалле интегрирован набор команд 1 Кбайт (ROM), 64-байтное ОЗУ (RAM), 27 портов ввода/вывода, восьмиразрядный таймер, схема прерываний, а также кварцевый осциллятор.

Для высокопроизводительных процессоров, наподобие Pentium (которые, впрочем, нацелены на другой сегмент рынка), необходимо большое число таких внешних компонентов как загрузочное ПЗУ (ROM), RAM в качестве рабочей памяти, жесткий диск в качестве устройства для хранения программ, таймер, схема прерываний и т.п., которые должны быть размещены в одном корпусе. В случае же однокристального микроконтроллера для работы не требуются никакие другие элементы.

По цене также наметилась существенная разница. В то время как процессоры, наподобие Pentium, вследствие сложного устройства и дорогостоящего процесса изготовления, предлагаются по цене в несколько сотен долларов, стоимость однокристального микроконтроллера находится в пределах от одного до 25 долларов.

Однокристальные микроконтроллеры находят широкое применение в самых разнообразных сферах: от измерительных приборов, фотоаппаратов и видеокамер, принтеров, сканнеров и копировальных аппаратов до изделий электронных развлечений и всевозможной домашней техники, в результате чего они получили нарицательное название "процессор для стиральных машин".

Учитывая обширную область применения и, соответственно, большие объемы продаж однокристальных микроконтроллеров, нет ничего удивительного в том, что практически каждый изготовитель полупроводников предлагает на рынке свой собственный "тип". Тем не менее, в восьмидесятые годы на рынок смог продвинуться главным образом восьмиразрядный процессор 8051 компании Intel со всеми его вариациями. Даже несмотря на то, что со временем на рынке появился конкурирующий 16-разрядный микроконтроллер, i8051 по-прежнему удерживает за собой пальму первенства, хотя в последнее время его и потеснили восьмиразрядные RISC-контроллеры нового поколения (RISC — Reduced Instruction Set Computer — компьютер с сокращенным набором команд). К таким RISC-микроконтроллерам, к примеру, относится семейство PIC компании-изготовителя Microchip,

а также семейство AVR компании Atmel, которые будут подробно рассмотрены в данной книге.

Можно заранее отметить, что компания Atmel, благодаря комбинации прогрессивной архитектуры RISC и программируемой флэш-памяти EPROM (**E**rasable **P**rogrammable **R**ead **O**nly **M**emory — стираемое программируемое постоянное запоминающее устройство), разработала очень эффективное решение для всех мыслимых задач внутриплатного управления и установила новые масштабы в соотношении цены и производительности.

Эта книга предназначена для проектировщиков, студентов, инженеров и ученых, для тех, кто обучается по электротехническим специальностям, а также для опытных любителей электроники — короче говоря, для всех, кто уже обладает определенными основными познаниями о построении и функционировании микроконтроллеров, желает понять и изучить однокристальные микроконтроллеры AVR и успешно претворить в жизнь собственные идеи, чтобы не отстать от "поезда", который движется в технологическое будущее.

Хотел бы выразить признательность за дружескую поддержку Бобу Хендерсону (Bob Henderson) и Гюнтеру Кисигу (Gunter Kiessig), сотрудникам компании Atmel и, прежде всего, Инго Швинду (Ingo Schwind) из немецкой компании Ventec — дистрибьютора Atmel. Они с большой заинтересованностью всеми силами помогали мне при написании этой книги. Также благодарю всех, кто поспособствовал появлению этого издания, и кого я не имею возможности перечислить поименно.

Выражаю особую признательность Гюнтеру Валю (Gunther Wahl) из издательства Francis' за оказанное мне доверие и доброе сотрудничество.

Буду благодарен читателям за новые идеи и предложения по улучшению моей книги. Если когда-нибудь перед кем-либо из вас возникнет проблема, для решения которой можно было бы использовать микроконтроллеры AVR, то сообщите мне. В том случае, если предоставленная информация будет представлять интерес также и для других читателей, возможно, в следующей моей книге по микроконтроллерам AVR вы найдете предложение по решению указанной проблемы. При этом, однако, вы должны понимать, что я не имею возможности ответить на каждый запрос, поскольку в этом случае я не смогу заниматься ничем другим!

Желаю моим читателям больших успехов при работе с этой книгой и последующего претворения в жизнь собственных идей.

Вольфганг Трамперт
Адрес электронной почты: W.Trampert@web.de
Домашняя страница: http://Trampert-AVR.de.vu

Условные обозначения

В этой книге приняты некоторые условные обозначения. Так, например, в литературе принята различная запись шестнадцатеричных чисел. В программах, написанных на языках Pascal и Basic, они обозначаются с помощью префикса в виде знака доллара (например, `$A3`). В программах, написанных на языке C или на ассемблере, которые применяются в семействе микроконтроллеров AVR компании Atmel, то же самое шестнадцатеричное число обозначается как `0xA3`, а в тексте часто представлено с индексом "h" ($A3_h$). В настоящей книге, в зависимости от случая применения, будут использоваться все перечисленные варианты.

В представленной ниже таблице сведены другие условные обозначения, используемые в книге.

Вид обозначения	*Написание*	*Примеры*
Шестнадцатеричные числа	Со знаком "$" перед числом	`$AF, $10, $FEDC`
	"0x" + шестнадцатеричное число	`0xAF, 0x10, 0xFEDC`
	С нижним индексом "h"	AF_h, 10_h, $FEDC_h$
Двоичные числа	"0b" + двоичное число	`0b10100101`
	С верхним индексом "b"	10100101_b
Десятичные числа	Без приставок	`15, 37, 1023`
	С нижним индексом "d"	15_d, 27_d, 1023_d
Команды ассемблера	Готический шрифт	`rjmp, clr, breq`
Инвертирование	Знак "/" впереди	/Rd, /Rr

В книге принято, что бит "установлен", если имеет значение логической "1" (лог. 1) и, соответственно, высокий уровень; "сброшен", если имеет значение логического "0" (лог. 0) и, соответственно, низкий уровень.

Для выделения определенных мест в тексте применяются следующие знаки:

Это знак важного примечания

Этим знаком обозначается дополнительная информация и полезные советы

1 Введение

Семейство микроконтроллеров AVR компании Atmel на момент написания настоящей книги состоит из четырех представителей: так называемой "базовой серии" (Basic Line), которые, как показано в табл. 1.1, отличаются по производительности (а следовательно, — и по стоимости). Благодаря этому, в зависимости от случая применения, в распоряжении всегда имеется самый оптимальный по производительности и стоимости микроконтроллер, что соответствует лозунгу "Не надо платить за ненужное!". Тот факт, что на сегодняшний день уже определены команды, не задействованные в первых четырех микроконтроллерах AVR, позволяет сделать вывод о том, что компания Atmel также и в будущем будет постоянно расширять семейство микроконтроллеров AVR за счет разработки более производительных образцов.

Архитектура RISC

Все микроконтроллеры AVR спроектированы для работы в составе очень производительной маломощной архитектуры RISC. Но по какой же причине это новое поколение компьютеров стало таким опасным конкурентом для уже утвердившихся на рынке?

Все микропроцессоры первого поколения обладали жестко "прошитой" логикой декодирования команд. В то время технология создания запоминающих устройств сильно "хромала" по сравнению с технологией изготовления процессоров, то есть, более скоростным процессорам приходилось долго ожидать считывания следующей команды для декодирования из медленнодействующего запоминающего устройства. Поэтому, с целью эффективного использования этого периода ожидания, возникла идея заполнить время до поступления следующей команды. В результате были разработаны сложные и комплексные машинные команды.

Возникли команды, которые выполняли последовательно несколько внутрипроцессорных операций до тех пор, пока не поступала следующая внешняя команда. Так появились процессоры CISC (**C**omplex **I**nstruction **S**et **C**omputer — компьютер со сложным набором команд). Типичными представителями этой архитектуры процессоров являются семейства 80x86 и процессор Pentium компании Intel или семейства 680x0 компании Motorola.

Практически все процессоры CISC работают по принципу микропрограммирования, то есть, каждая машинная команда последовательно обрабатывается микропрограммой, которая выполняется внутри кристалла процессора.

Однако если взглянуть на существующие программы через призму статистики, то обнаружится, что из большого набора команд процессоров CISC (у некоторых типов — свыше 300) используется только небольшая, постоянно повторяющаяся часть в размере около 20 процентов.

Представители базовой серии AVR

В табл. 1.1 перечислены характеристики представителей базовой серии микроконтроллеров семейства AVR. Сравнив эти характеристики, можно быстро определить элементную базу для каждого конкретного случая применения.

Таблица 1.1. Отличительные признаки четырех представителей базовой серии микроконтроллеров AVR

Характеристика	*AT90S8515*	*AT90S4414*	*AT90S2313*	*AT90S1200*
Количество команд	118	118	118	89
Флэш-EPROM	8 Кбайт	4 Кбайта	2 Кбайта	1 Кбайт
Программирование через интерфейс SPI	да	да	да	да
EEPROM	512 байт	256 байт	128 байт	64 байта
SRAM	512 байт	256 байт	128 байт	только стек
Рабочие регистры	32	32	32	32
Контакты ввода/ вывода	32	32	15	15
Приемопередатчик UART	да	да	да	нет
Интерфейс SPI	да	да	нет	нет
Сторожевой таймер	да	да	да	да
8-разрядный таймер/счетчик с предделителем частоты	да	да	да	да
16-разрядный таймер/ счетчик с предделителем частоты	да	да	да	нет
ШИМ-модулятор	2	2	1	нет
Источники прерывания	12	12	10	3
Аналоговый компаратор	да	да	да	да
Ждущий режим	да	да	да	да
Режим пониженного энергопотребления	да	да	да	да
Рабочее напряжение	2,7...6 В	2,7...6 В	2,7...6 В	2,7...6 В
Тактовая частота при Vcc = + 5 В	0...8 МГц	0...8 МГц	0...10 МГц	0...12 МГц
Тактовая частота при Vcc = + 3 В	0...4 МГц	0...4 МГц	0...4 МГц	0...4 МГц
Корпус	PDIP40 PLCC44	PDIP40 PLCC44	PDIP20 SOIC20	PDIP40 SOIC20 SSOP20

2 Обзор

Основные характеристики семейства микроконтроллеров AVR

В принципе, все микроконтроллеры построены по одной схеме. Система управления, состоящая из счетчика команд и схемы декодирования, берет на себя считывание и декодирование команд из памяти программ, а операционное устройство отвечает за выполнение арифметических и логических операций; интерфейс ввода/вывода позволяет обмениваться данными с периферийными устройствами; и, наконец, необходимо иметь запоминающее устройство для хранения программ и данных (рис. 2.1).

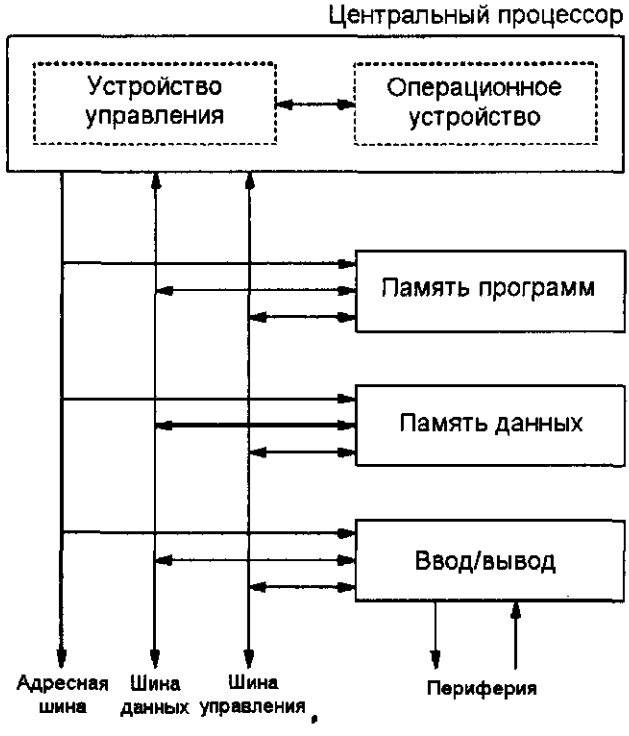


Рис. 2.1. Базовая структура микроконтроллера

Операционное устройство, как правило, состоит из арифметико-логического устройства (АЛУ), накапливающего сумматора и нескольких вспомогательных регистров. В классической программе почти половина всех команд — это команды пересылки (`move`) для передачи данных от вспомогательных регистров к накапливающему сумматору и обратно. В семействе микроконтроллеров AVR накапливающий сумматор, представляющий собой "тонкое место", становится не столь критически важным, благодаря применению 32-х рабочих регистров, связанных

с блоком АЛУ. В результате арифметические и логические операции могут быть выполнены в течение единственного такта.

Арифметико-логическое устройство

Операции АЛУ можно разделить на три основные категории: арифметические, логические и поразрядные. Каждой из этих категорий соответствуют мощные команды. Некоторые (новые) микроконтроллеры семейства AVR имеют также аппаратные умножители в арифметическом блоке АЛУ.

Структура команд

Несмотря на RISC-архитектуру, запас команд не так уж и ограничен: как никак, AT90S8515, "парадная лошадь" базовой серии, имеет 118 команд, в высшей степени оптимизированных с точки зрения целесообразности и эффективности, а самый "слабый" представитель семейства — AT90S1200 — едва ли сильно "отстает от лидера" со своими 89 командами. В микроконтроллерах AVR базовой серии все команды имеют одинаковую ширину слова 16 бит (то есть, 2 байта). Исключением являются только две команды для прямой адресации статической памяти данных `lds` и `sts`, состоящие из двух слов и, соответственно, из 4 байт.

За немногими исключениями (команда перехода, операции с непосредственной адресацией и пересылки в память/из памяти) все команды обрабатываются в течение одного такта системной синхронизации.

Время выполнения команды

На рис. 2.2 показано сопоставление количества необходимых тактовых импульсов осциллятора для выполнения двух последовательных команд процессорами 68HC05, 80C51, а также семействами микроконтроллеров PIC и AVR.

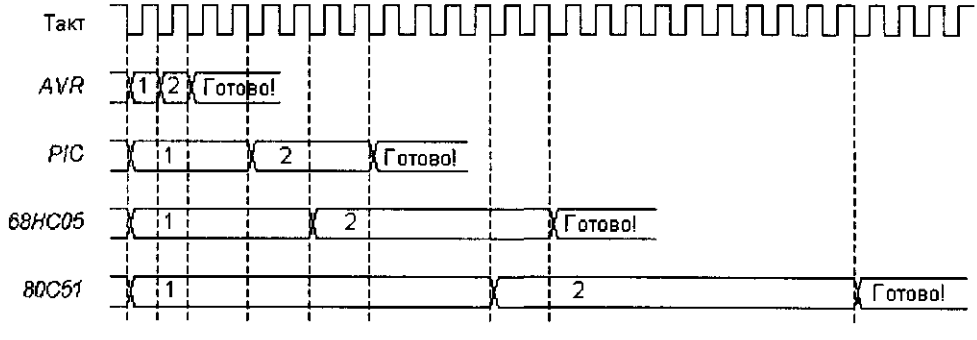


Рис. 2.2. Сравнение времени выполнения команд различными процессорами

Архитектура памяти

В запоминающих устройствах, соответствующих классической концепции фон Неймана, данные и команды хранятся в одной памяти. В противоположность этому, память по Гарвардской архитектуре, используемой в микроконтроллерах AVR, состоит из нескольких компонентов. В данном случае разделены память ко-

манд и память данных, то есть, обращение к командам осуществляется независимо от доступа к данным.

В микроконтроллерах AVR отдельные сегменты памяти устроены по-другому также и физически.

- Память программ реализована на основе программируемой и электрически стираемой флэш-технологии. Во всех микроконтроллерах AVR память 16-разрядная (двухбайтовая), и всегда находится "на кристалле". Расширение памяти программ с помощью блоков EPROM или флэш невозможно.
- Внутренняя память для энергозависимых данных (то есть, данных, которые будут потеряны после отключения рабочего напряжения), представляет собой статическую память RAM (SRAM). Преимущество этого заключается в том, что отпадает всякая необходимость во внутренней регенерации как в случае с некоторыми другими процессорами, которые используют динамическую память. По этой причине микроконтроллеры AVR могут работать с тактами вплоть до 0 Гц. В дополнение к этому, некоторые микроконтроллеры AVR для увеличения объема обрабатываемых данных могут работать с подключаемой внешней памятью SRAM. Для этого, однако, приходится жертвовать драгоценными контактами ввода/вывода портов A и C.
- Для данных, которые должны сохраниться после отключения рабочего напряжения, в распоряжении имеется микросхема EEPROM (**E**lectrically **EPROM** — электрически стираемое ППЗУ). В память EEPROM можно записывать данные во время нормального выполнения программы. Для области EEPROM также нет обязательной необходимости в программирующем устройстве.

Область ввода/вывода

В микроконтроллерах AVR область ввода/вывода занимает 64 байта и находится в блоке статической памяти SRAM по адресам от $20 до $5F. В микроконтроллерах AVR функции ввода/вывода для взаимодействия со внешним миром применимы не только для портов (от A до D), но также и для всех регистров состояния и управления таких "встроенных в кристалл" периферийных функций как таймер, устройство UART, сторожевой таймер, доступ на запись и чтение к памяти EEPROM, интерфейс SPI и т.д.

Прерывания и подпрограммы

Техника подпрограмм и прерываний в микроконтроллерах AVR может быть применена обычным образом. Переход к частям программы, которые в процессе ее выполнения обрабатываются многократно или должны содержаться отдельно по причине структурирования, осуществляется с помощью команды `rcall`. Это — команда перехода, которая помещает в стек следующую команду нормального выполнения программы в качестве адреса возврата. После выполнения подпрограммы адрес извлекается из стека с помощью команды `ret`, и программа продолжается.

Стек для адресов возврата находится в памяти SRAM. Исключение составляет только микроконтроллер AT90S1200, который применяет для этой цели аппарат-

ный стек по принципу LIFO (**L**ast **I**n, **F**irst **O**ut — "последним вошел, первым вышел").

В примере на рис. 2.3 указатель стека инициализируется, начиная с метки `Start`, после чего следует метка `Call1` первого вызова подпрограммы `UP1`. В качестве адреса возврата в стек помещается адрес следующей команды, то есть, `$78A`. Сама подпрограмма `UP1` в представленном примере не выполняет никаких действий, и происходит немедленный возврат к главной программе. Для этого из стека извлекается адрес, и выполнение программы продолжается с команды `rjmp Call2`. С метки `Call2` происходит повторный вызов подпрограммы `UP1`. В этом случае в стек помещается адрес возврата `$78D`.

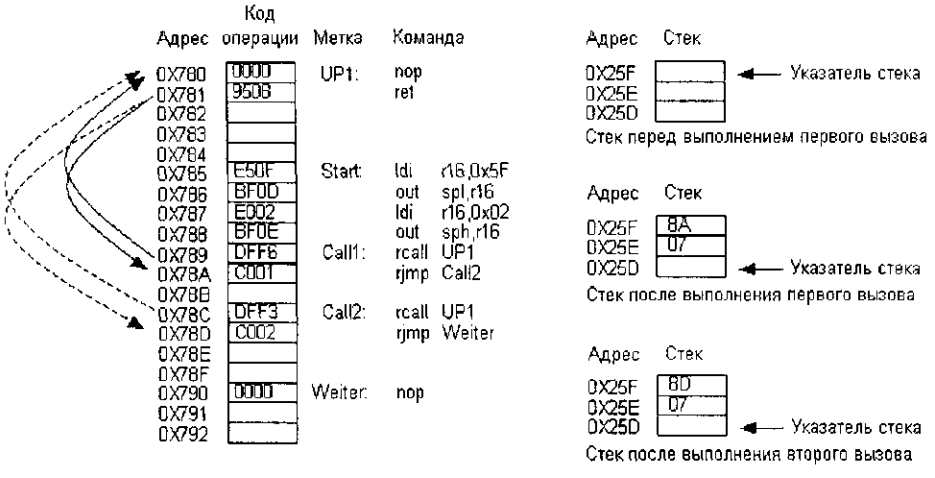


Рис. 2.3. Два вызова подпрограммы

В правой части рис. 2.3 показан фрагмент статической памяти SRAM, отведенной для стека. Указано содержимое стека перед выполнением первого и после выполнения первого и второго вызовов. Подпрограммы также могут быть вложенными (то есть, из подпрограммы можно вызвать другую программу).

Особой формой подпрограмм являются подпрограммы обработки прерываний, с помощью которых реализованы заранее не запланированные обращения из программ. Другими словами, выполняющаяся программа асинхронна по отношению к событию, вызывающему прерывание. Здесь, например, речь может идти о работе таймера, приеме байта через приемопередатчик UART или запросе на прерывание от внешнего устройства.

В зависимости от построения микроконтроллера, в распоряжении базовой серии AVR имеется до 12 векторов прерывания (адресов входа).

Периферийные функции

С точки зрения аппаратной части, в микроконтроллерах AVR реализовано множество периферийных функций. Так, например, "парадная лошадь" AT90S8515 обладает следующими особенностями:

- 32 программируемые линии ввода/вывода;
- программируемое устройство UART;

- синхронный интерфейс SPI;
- 8-разрядный таймер/счетчик;
- 16-разрядный таймер/счетчик с функциями сравнения и захвата и возможностью реализации двух выходов ШИМ (широтно-импульсный модулятор);
- интегрированный сторожевой таймер;
- аналоговый компаратор.

Блок-схема микроконтроллеров AT90S1200 и AT90S8515

Архитектура семейства микроконтроллеров AVR показана на примере ее наиболее простого в настоящее время представителя AT90S1200 (рис. 2.4) и самого производительного микроконтроллера AT90S8515 (рис. 2.5).

Два других представителя семейства — микроконтроллеры AT90S2313 и AT90S4414 — оптимизированы с точки зрения размера элементов и/или цены (см. табл. 1.1) и предоставляют в распоряжение то же количество функций и модулей, что и микроконтроллер AT90S8515.

Конструктивное исполнение корпусов и расположение выводов

Расположение выводов четырех представителей базовой серии микроконтроллеров AVR показано на рис. 2.6 и рис. 2.7. Рассмотрим назначение выводов подробнее.

VCC

Подвод питающего напряжения

GND

Заземление

Port A (PA0...PA7)

Порт A представляет собой двунаправленный порт ввода/вывода с пропускной способностью 8 бит. Буфер вывода порта A в режиме приема данных в состоянии принимать ток силой до 20 мА и, благодаря этому, напрямую питать, например, светодиоды. Каждый вывод порта может быть сконфигурирован индивидуально как вход или выход, а при выполнении функции ввода к нему, при желании, можно подключать подтягивающее сопротивление.

В качестве особой функции через порт A работает демультиплексированная шина передачи данных и адресов, если к микроконтроллеру AVR должна быть подключена внешняя память RAM.

Port B (PB0...PB7)

Порт B представляет собой двунаправленный порт ввода/вывода (I/O) с пропускной способностью 8 бит. Буфер вывода порта B в режиме приема данных

в состоянии принимать ток силой до 20 мА и, благодаря этому, напрямую питать, например, светодиоды. Каждый вывод порта может быть сконфигурирован индивидуально как вход или выход, а при выполнении функции ввода к нему, при желании, можно подключать подтягивающее сопротивление.

Альтернативно, через порт B могут выполняться также различные специальные функции (таймер, подключение входов аналогового компаратора, интерфейс SPI).

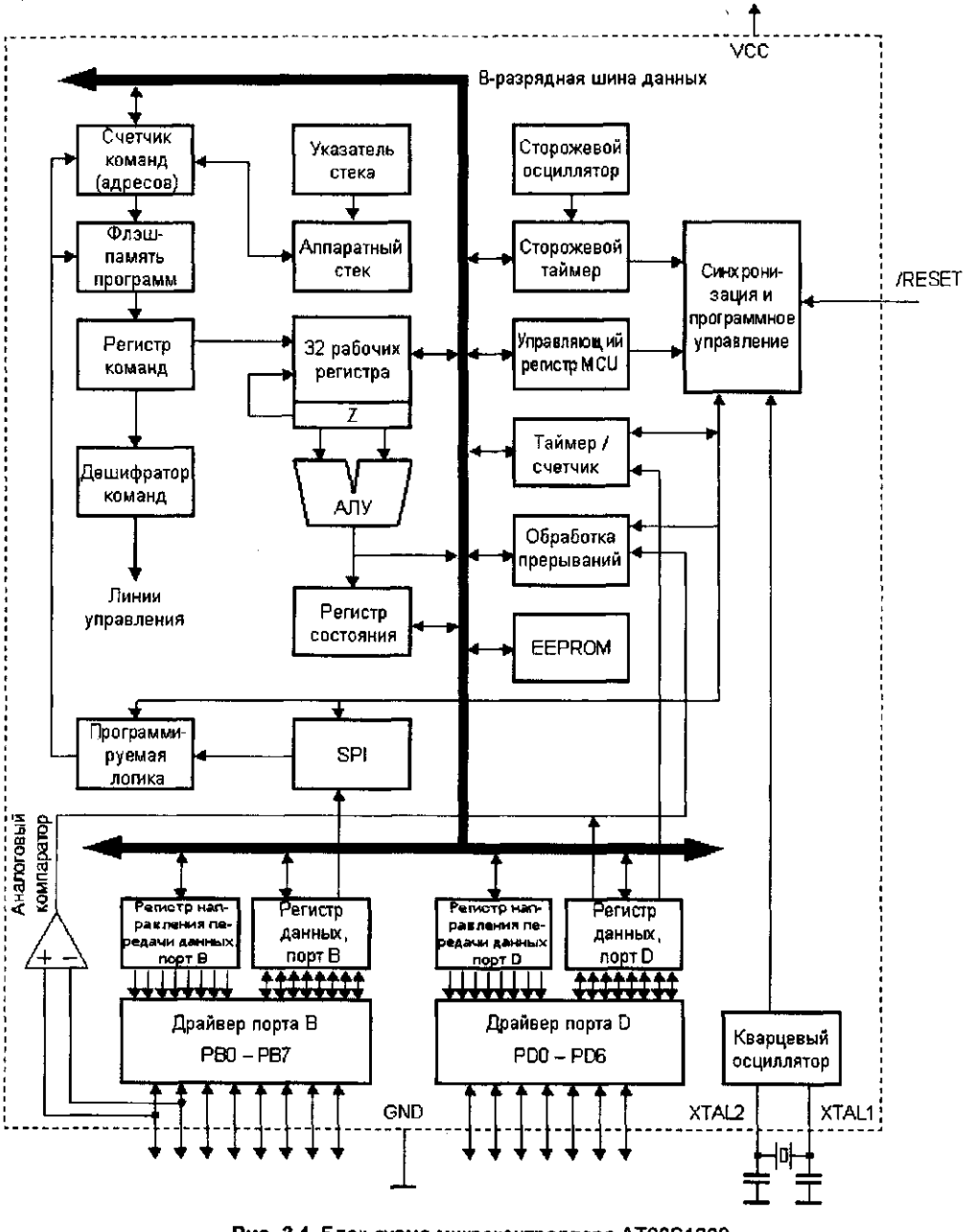


Рис. 2.4. Блок-схема микроконтроллера AT90S1200

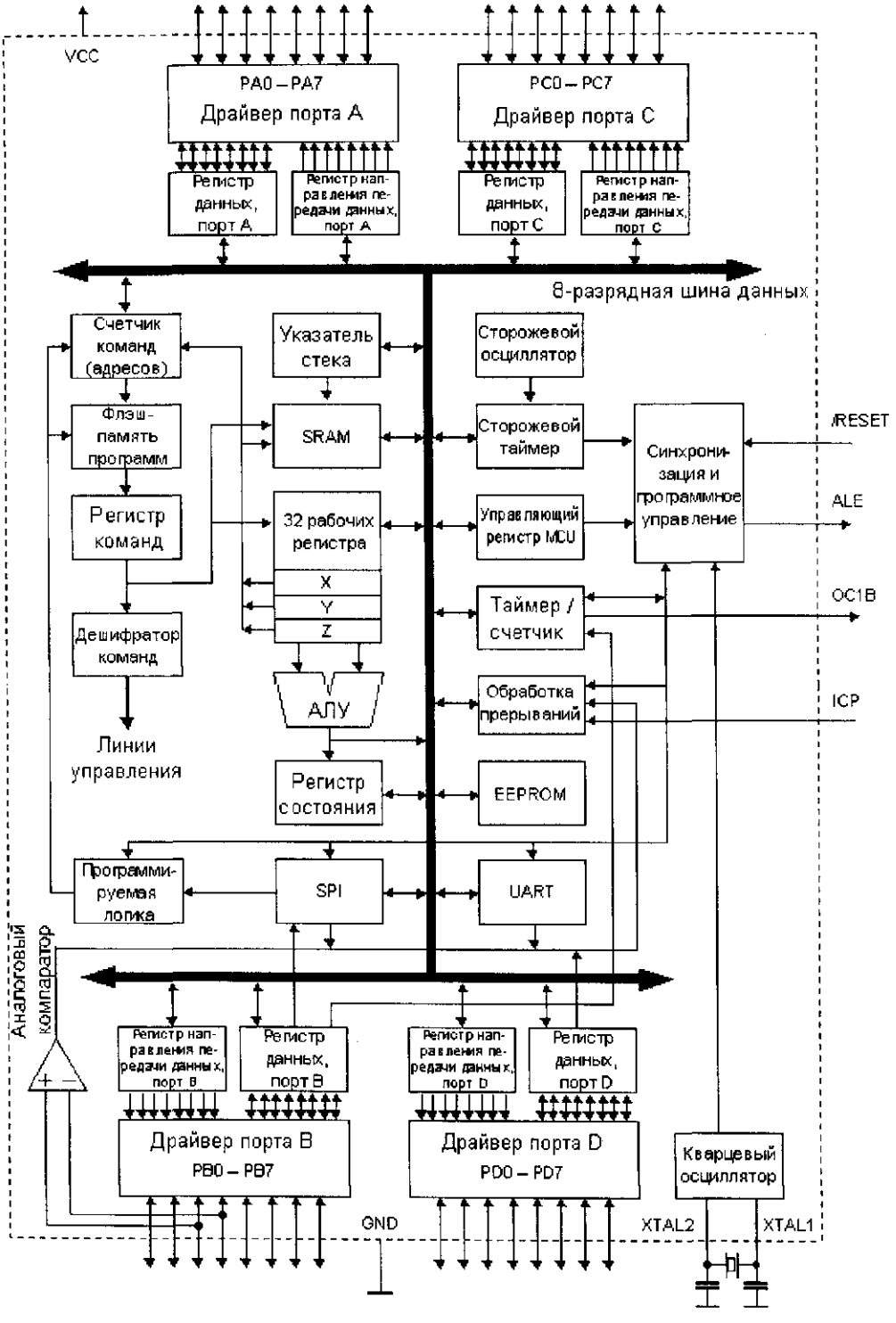


Рис. 2.5. Блок-схема микроконтроллера AT90S8515

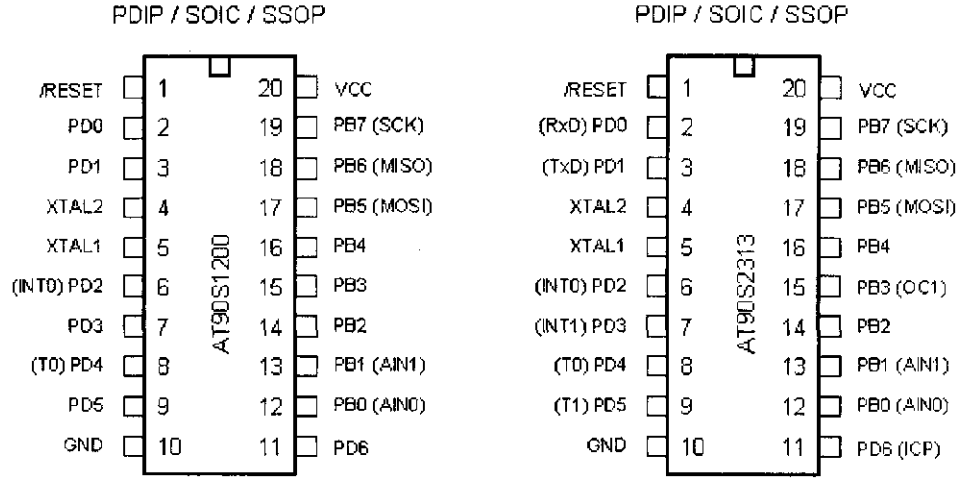


Рис. 2.6. Расположение выводов микроконтроллеров AT90S1200 и AT90S2313 (вид сверху)

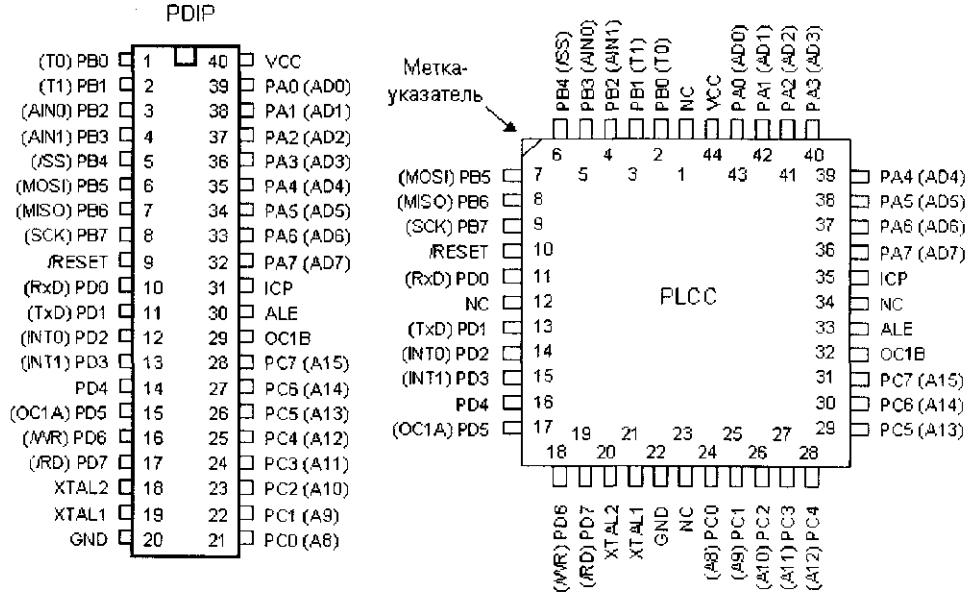


Рис. 2.7. Конструктивное исполнение корпусов AT90S4414 и AT90S8515 (вид сверху)

Port C (PC0...PC7)

Порт С представляет собой двунаправленный порт ввода/вывода (I/O) с пропускной способностью 8 бит. Буфер вывода порта С в режиме приема данных в состоянии принимать ток силой до 20 мА и, благодаря этому, напрямую питать, например, светодиоды. Каждый вывод порта может быть сконфигурирован индивидуально как вход или выход, а при выполнении функции ввода к нему, при желании, можно подключать подтягивающее сопротивление.

В качестве особой функции через порт C выводится старший байт адресной шины, если к микроконтроллеру AVR должна быть подключена внешняя память RAM.

Port D (PD0...PD7)

Порт D представляет собой двунаправленный порт ввода/вывода с пропускной способностью 8 бит. Буфер вывода порта D в режиме приема данных в состоянии принимать ток силой до 20 мА и, благодаря этому, напрямую питать, например, светодиоды. Каждый вывод порта может быть сконфигурирован индивидуально как вход или выход, а при выполнении функции ввода к нему, при желании, можно подключать подтягивающее сопротивление.

Альтернативно, через порт D могут выполняться также различные дополнительные функции (например, поступление запросов на прерывание, передача выходных данных таймера, интерфейс с устройством UART).

/RESET

Вывод для подачи сигнала сброса. Уровень лог. 0 на этом выводе на протяжении минимум двух циклов системного такта **Ф** при активном осцилляторе переводит микроконтроллер в исходное состояние.

ICP

Вывод функции "Захват" (Capture) интегрированного таймера/счетчика T/C1.

OC1B

Вывод функции "Сравнение" (Compare) интегрированного таймера/счетчика T/C1.

ALE

Вывод для подачи импульса при записи младшего адресного байта с демультиплексированной шины данных/адреса через порт A во внешний фиксирующий регистр, когда к микроконтроллеру AVR подключена внешняя память RAM (см. рис. 3.6, рис. 3.7 и рис. 3.8). Байт данных передается на втором шаге обращения к памяти RAM через порт A.

XTAL1

Вход интегрированного осциллятора для выработки такта системной синхронизации **Ф** и, равным образом, вход для внешнего тактового сигнала, если внутренний осциллятор не применяется.

XTAL2

Выход интегрированного осциллятора для выработки такта системной синхронизации **Ф.**

Генерирование такта системной синхронизации в микроконтроллерах AVR

Для генерирования тактов системной синхронизации в микроконтроллерах семейства AVR используется интегрированный осциллятор (выводы XTAL1 и XTAL2), вырабатывающий такты Ф. В качестве альтернативного варианта может также использоваться внешний тактовый сигнал. В случае микроконтроллера AT90S1200 возможен еще один вариант, поскольку в этом микроконтроллере для генерирования такта системной синхронизации может быть применен интегрированный RC-осциллятор, вырабатывающий тактовый сигнал для сторожевого таймера (см. главу 5, "Сторожевой таймер") и схемы сброса (см. пункт "Сброс и обработка прерываний" главы 3).

Интегрированный кварцевый осциллятор базовой серии микроконтроллеров AVR

Интегрированный кварцевый осциллятор используется всеми представителями базовой серии микроконтроллеров AVR.

XTAL1 и XTAL2 — это входы/выходы инвертирующего усилителя, который может быть применен как встроенный осциллятор для генерирования такта системной синхронизации Ф. Генерирование колебаний может осуществляться кварцевым или керамическим резонатором. Емкости конденсаторов C1 и C2 на рис. 2.8, которые вместе с кварцем и внутренним инвертором образуют генератор Пирса, обычно составляют 22 пФ.

Если микроконтроллер AVR должен работать при тактировании от внешнего источника, то сигнал подводится на вход XTAL1, а выход XTAL2 встроенного осциллятора в этом случае остается открытым (рис. 2.9).

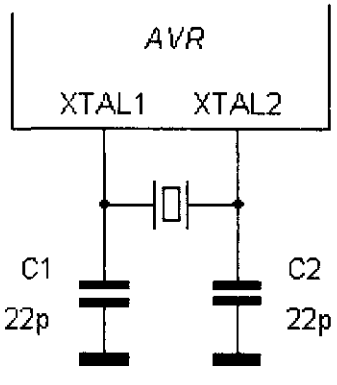


Рис. 2.8. Выработка такта системной синхронизации Ф с помощью встроенного осциллятора

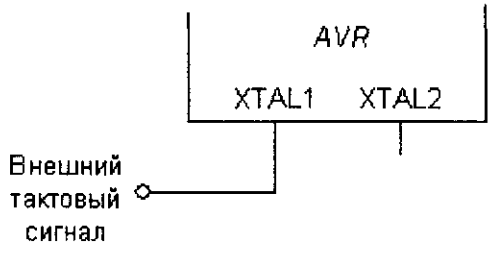


Рис. 2.9. Применение внешнего тактового сигнала в качестве такта системной синхронизации Ф

Генерирование такта системной синхронизации с помощью контура RC-осциллятора

Генерирование такта системной синхронизации с помощью контура RC-осциллятора реализовано только в модели AT90S1200. Для случаев применения, особенно чувствительных к быстродействию, когда не предъявляются большие требования к точности такта системной синхронизации, в качестве тактового источника микроконтроллера AT90S1200 вместо кварцевого осциллятора может быть использован RC-осциллятор, присутствующий на кристалле всех микроконтроллеров AVR.

Этот осциллятор служит в первую очередь для подачи тактов на сторожевой таймер (см. главу 5, "Сторожевой таймер") и схему сброса (см. раздел "Сброс и обработка прерываний"). Он колеблется с постоянной частотой около 1,1 МГц при рабочем напряжении Vcc = 5 В. Частоты для других значений рабочего напряжения могут быть взяты из рис. 2.10.

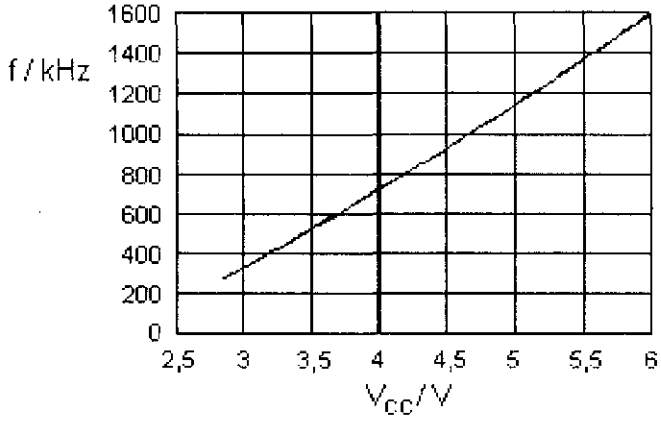


Рис. 2.10. Частота встроенного RC-осциллятора в зависимости от питающего напряжения Vcc при температуре окружающей среды T_A =25°C

Само собой разумеется, точность RC-осциллятора находится в рамках обычных электрических схем осцилляторов типа RC, и зависит от температуры. Для работы микроконтроллера AT90S1200 совместно с RC-осциллятором нет необходимости в каких-либо внешних компонентах, вывод XTAL2 кварцевого осциллятора может оставаться неподключенным, а вход XTAL1 должен быть установлен в лог. 0 или лог. 1.

Осциллятор типа RC в случае микроконтроллера AT90S1200 выбирается в качестве тактового генератора с помощью управляющего разряда RCEN (**RC En**able — "RC-осциллятор активен") во флэш-памяти. Если разряд RCEN запрограммирован (то есть, находится в состоянии лог. 0), то выбирается встроенный в кристалл RC-осциллятор; а в том случае, если RCEN = лог. 1, такт системной синхронизации вырабатывается кварцевым осциллятором.

По молчанию, разряд RCEN микроконтроллера AT90S1200 при поставке не запрограммирован (лог. 1), хотя можно приобрести микроконтроллер AT90S1200A и с запрограммированным разрядом RCEN (лог. 0).

Пользователь может изменить разряд RCEN только в параллельном режиме программирования. Если микроконтроллер AT90S1200 должен работать с RC-осциллятором, а данные для памяти флэш и EEPROM должны быть запрограммированы в последовательном режиме, то разряд RCEN должен быть заранее запрограммирован в параллельном режиме.

Программирование для AVR на языке высокого уровня C

В связи с тем, что микроконтроллеры AVR были полностью новой разработкой, и отпала необходимость учитывать совместимость с какими-либо предшественниками, появилась возможность воспользоваться самыми современными достижениями. Так, например, с самого начала в проектирование и разработку структур микроконтроллеров AVR были вовлечены специалисты по языкам высокого уровня — прежде всего, языку программирования C. В результате была получена архитектура процессора, специально рассчитанная на составление программ на языке C.

Ниже на примере двух коротких фрагментов программ показано, как эффективно элементы языка C могут быть превращены в коды ассемблера для микроконтроллеров AVR.

Исходный код на языке C	*Сгенерированные коды AVR*
`unsigned char *var1;`	`ld r16,-X`
`unsigned char *var2;`	`st Z+, r16`
`*var1++ = *var2;`	
`void routine (void)`	`...`
`{`	`cp r0, r4 ;n1-n2 (byte 0)`
`long n1, n2;`	`cpc r1, r5 ;n1-n2-C (byte 1)`
`int n3`	`cpc r2, r6 ;n1-n2-C (byte 2)`
`...`	`cpc r3, r7 ;n1-n2-C (byte 3)`
`if (n1 != n2) n3 += 5;`	`breq EQUAL ; Переход, если равно`
`...`	`subi r16, Low (0xfffb) ;n3+5, младший байт`
`}`	`sbci r17, High (0xfffb) ;n3+C, старший байт`
	`EQUAL:`
	`...`

В первом примере показана типичная операция с указателем в том виде, в котором она очень часто встречается в языке программирования C.

Во втором примере `n1` и `n2` — целые числа типа Long Integer, то есть, оба имеют длину 4 байта. `n3` — это целочисленная переменная длиной 2 байта. Значения `n1`, `n2` и `n3` определены внутри подпрограммы как локальные переменные.

В коде, сгенерированном на ассемблере, число `n1` размещается в регистрах `r3–r0`, `n2` — в регистрах `r7–r4`, а `n3` — в регистрах `r17–r16`.

Кроме того, во втором примере показано, что отсутствующие в наборе AVR-команд команды `addi` (add immediate — "добавить непосредственное значение") и `adci` (add immediate with carry — "добавить непосредственное значение с учетом

переноса") для вычисления выражения n3+5 могут быть заменены вычитанием второго компонента из 5.

Стендовые испытания для сравнения микроконтроллеров AVR с главными конкурентами

То, что применение языка высокого уровня C при составлении программ для AVR дает в результате чрезвычайно сжатый код на ассемблере (а значит, и машинный код), доказывают также и стендовые испытания базовой серии микроконтроллеров AVR. При этом результаты сопоставлялись с главными конкурентами (80C51, 68HC11, 80196 ...), а критерием для оценки был объем кода, сгенерированного для выполнения одной и той же задачи различными процессорами.

Данные по сравнительным испытаниям микроконтроллеров семейства AVR с распространенными на рынке 8- и 16-разрядными микроконтроллерами, предоставлены компанией Atmel. Во время испытаний решались девять распространенных практических задач из различных областей применения процессоров. Для получения сравнимых результатов испытаний, программа для всех процессоров была написана на языке высокого уровня C.

Для трансляции кода почти во всех случаях был применен компилятор фирмы IAR Systems Ltd (на момент проведения испытаний компилятора этой фирмы не существовало только для процессора ARM7/ARM Thumb). Благодаря этому, результаты испытаний зависели не от возможностей оптимизации различных изготовителей компиляторов, а в значительной степени от архитектурных отличий процессоров. Тем временем уже появились более новые варианты некоторых компиляторов, вследствие чего представленные здесь результаты могли бы выглядеть еще лучше. Это относится также и к компилятору AVR-C, который является относительно "молодым" и с появлением каждой новой версии заметно улучшается в плане создания результирующего кода.

Само собой разумеется, стендовые испытания этого вида можно назвать объективными только условно, и потому их наглядность ограничена, поскольку каждый процессор имеет свои особые сильные и слабые стороны.

Эффективность кода микроконтроллера сильно зависит от постановки задачи — нет никаких универсальных "лучших микроконтроллеров" для всех случаев применения.

С помощью стендовых испытаний (как и с помощью статистики) можно доказать почти все, что угодно, поскольку каждый микроконтроллер имеет свои "сильные стороны", и если его сконструировать для особых случаев реализации этих специфических сильных сторон, то он может стать "эффективным с точки зрения кода".

Таким образом, на основании этого сопоставления мы не будем объявлять микроконтроллеры AVR самыми лучшими из имеющихся на рынке. Цель данных испытаний — показать, что архитектура отдельных представителей семейства AVR в высшей степени оптимизирована с точки зрения объема программного кода.

Ниже на рис. 2.11–2.19 представлены результаты отдельных тестов, а также общий результат испытаний.

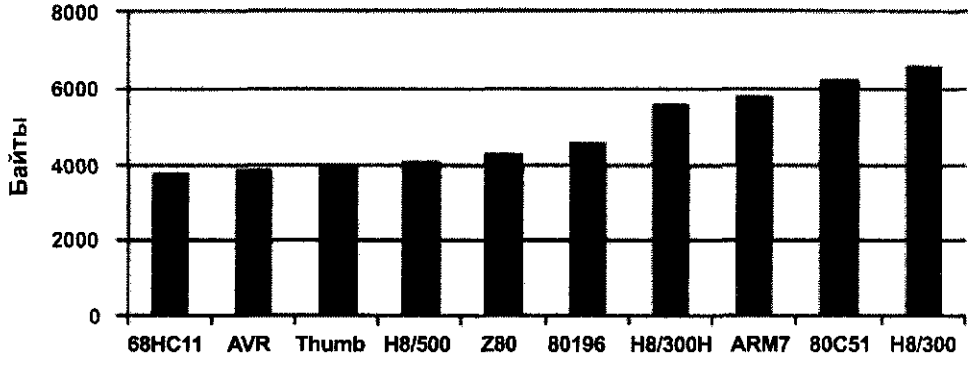


Рис. 2.11. Протокол пейджера с тремя уровнями, включая драйвер

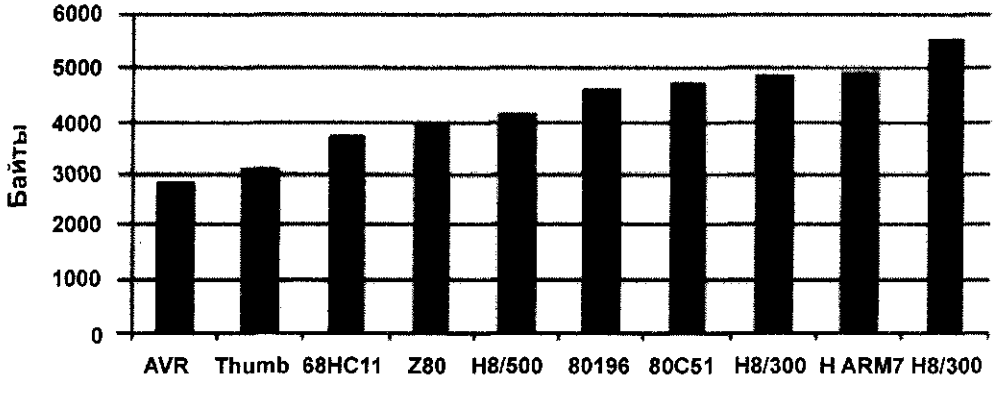


Рис. 2.12. Аналоговый телефонный аппарат 1 с интерфейсом SIM и управлением дисплеем

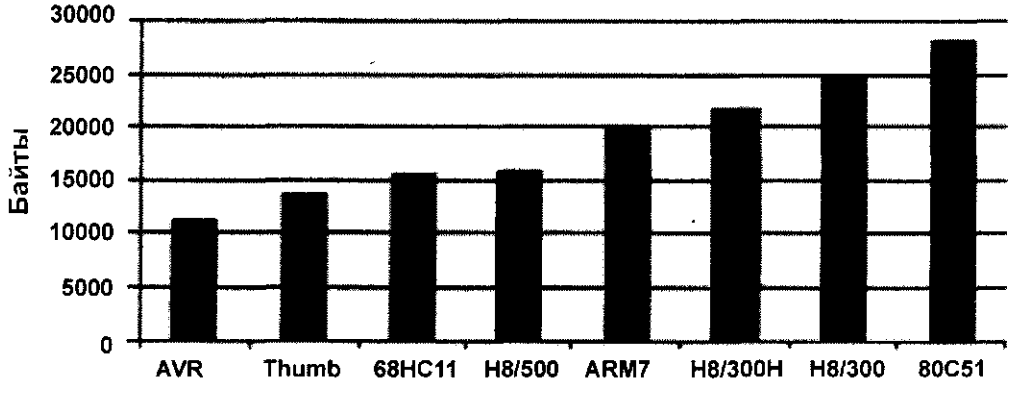


Рис. 2.13. Аналоговый телефонный аппарат 2 на базе конечных автоматов

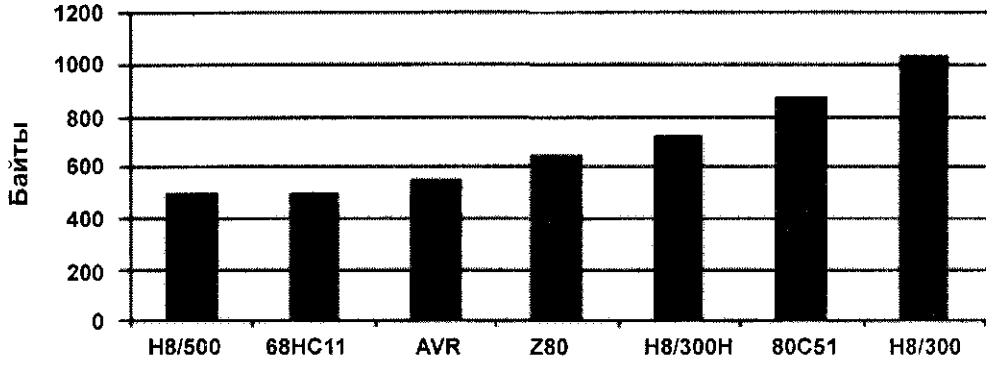


Рис. 2.14. Аналоговый телефонный аппарат 3: собраны типичные подпрограммы

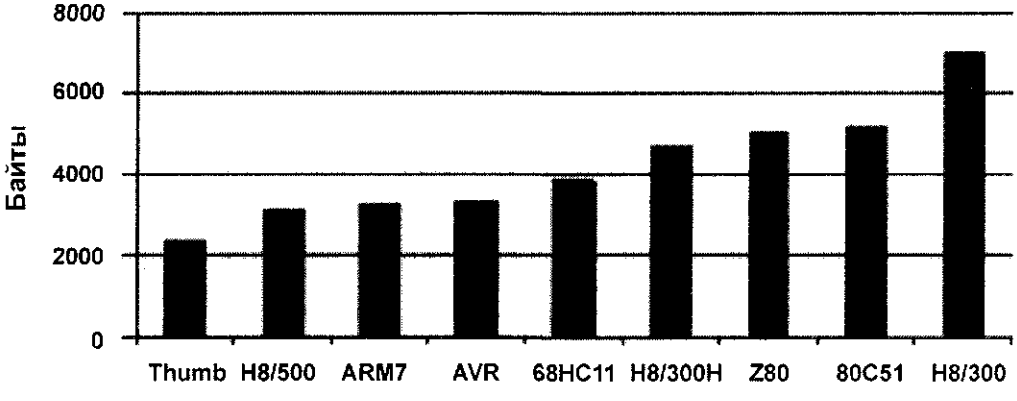


Рис. 2.15. Шифратор/дешифратор Рида-Соломона (алгоритм коррекции ошибок)

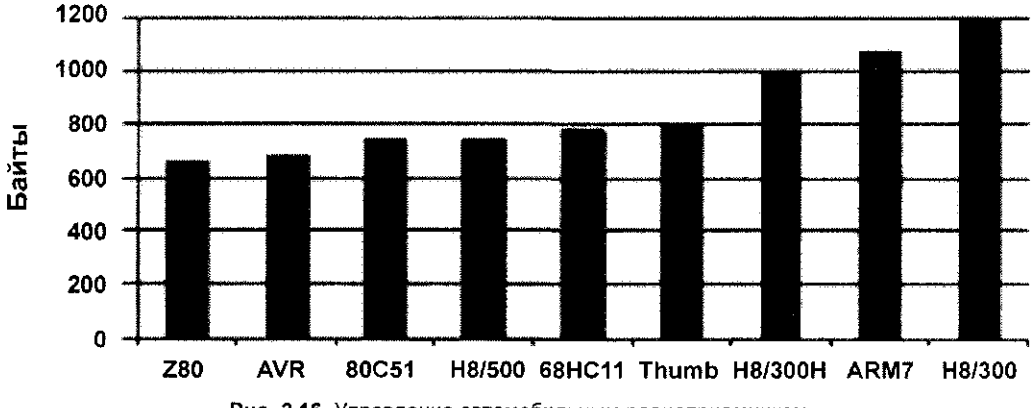


Рис. 2.16. Управление автомобильным радиоприемником

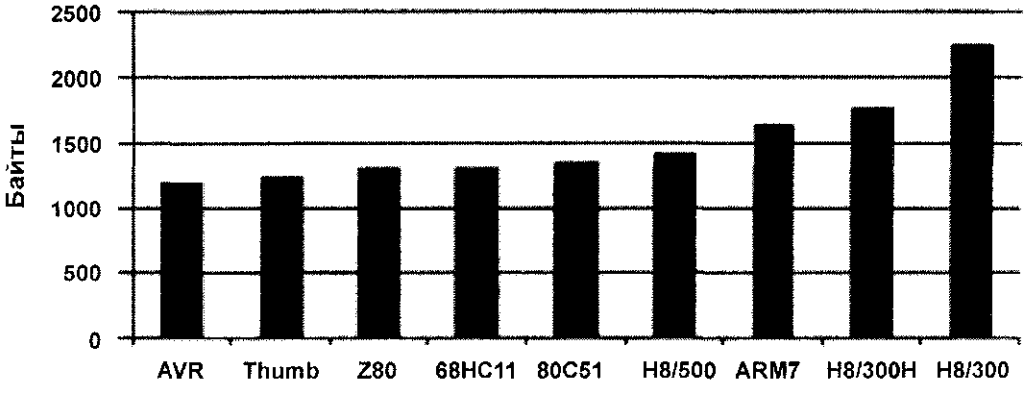


Рис. 2.17. Обработка переменных Битфельда различной величины

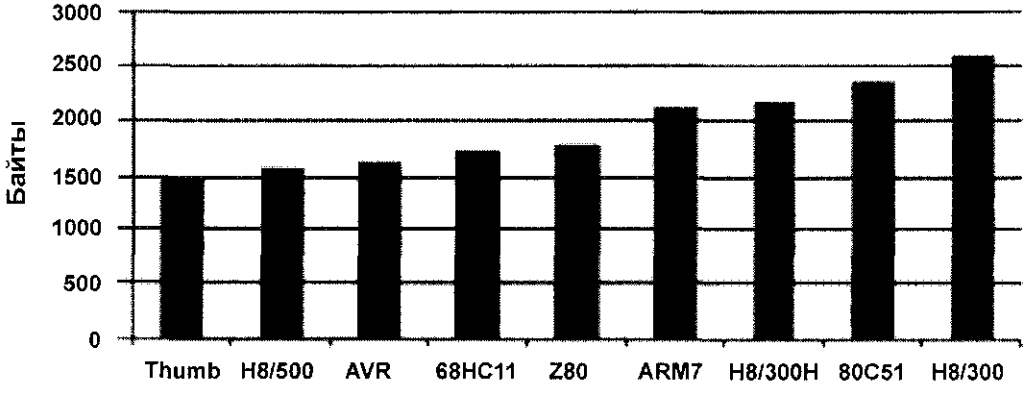


Рис. 2.18. Алгоритм DES для кодирования и декодирования данных

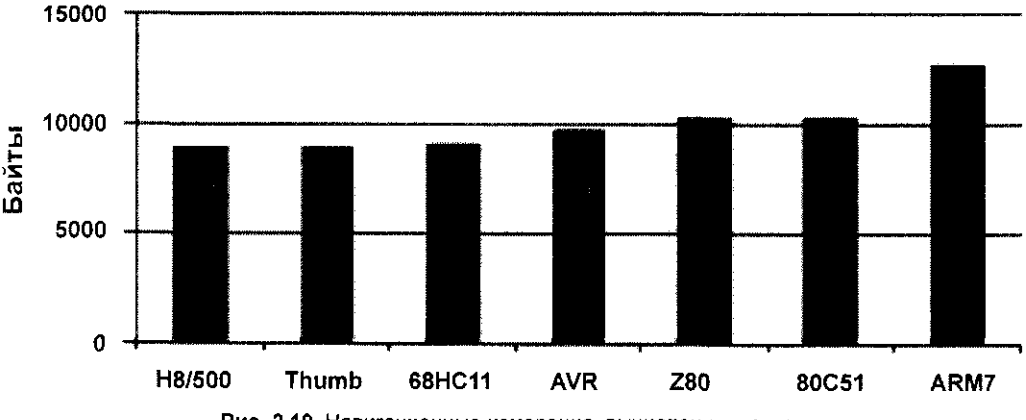


Рис. 2.19. Навигационные измерения, вычисления и связь

Результаты

В семи из девяти тестов микроконтроллер AVR был среди трех лучших и, как показывает общая оценка, он резко отличается в лучшую сторону от всех протестированных контроллеров.

На рис. 2.20 представлены результаты отдельных тестов, просуммированные и нормализованные на основе результатов микроконтроллера AVR. Например, для ARM7 объем кода получился в среднем в полтора раза больше, чем для AVR.

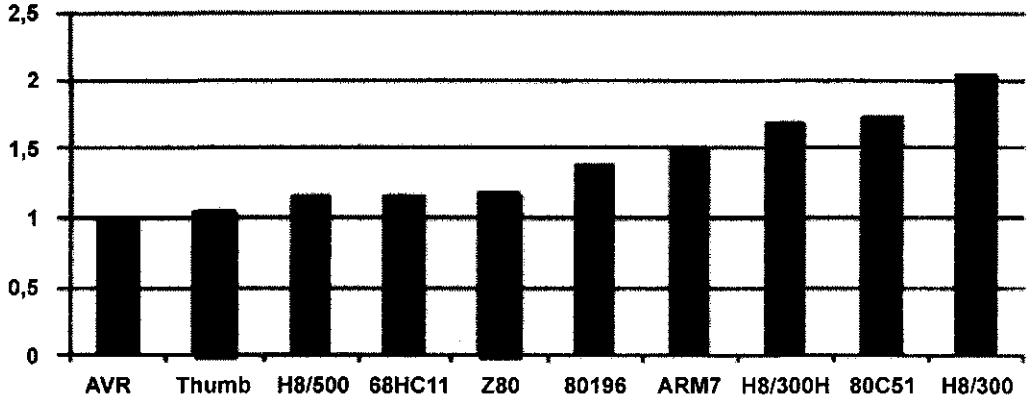


Рис. 2.20. Суммарный результат (нормализованный)

Ведущим производителем С-компиляторов можно считать компанию IAR Systems Ltd.. За их прекрасные "ноу-хау" в области компиляторов действительно стоит заплатить высокую цену. На рынке уже существуют намного более дешевые С-компиляторы SmallC AVR (естественно, при ограниченной производительности), стоимость которых ниже 50 долларов США.

Для любителей других языков программирования на рынке имеются следующие компиляторы AVR:

- для языка Pascal — компания E-LAB, Германия, адрес в Internet: `http://www.e-lab.de/`;
- для языка Basic — компания Silicon Studio, адрес в Internet: `http://www.sistudio.com/`.

3 Центральный процессор и внутренняя память

Микроконтроллер AVR построен и работает по модульному принципу, как показано в табл. 1.1, однако для многих случаев применения нет необходимости в полной комплектации, представленной в микроконтроллере AT90S8515. По этой причине, наряду с "парадной лошадью" базовой серии микроконтроллеров AVR — AT90S8515 — можно приобрести еще три члена семейства AVR, в которых с целью снижения цены или уменьшения размеров корпуса нет того или иного функционального блока (см. табл. 1.1).

В настоящей главе будут подробно рассмотрены все основные системы центрального процессора и несколько внутренних блоков памяти.

Система управления и АЛУ

Система управления регулирует процесс выполнения программы и контролирует взаимодействие отдельных встроенных в кристалл модулей, как это показано на блок-схемах на рис. 2.1 и рис. 2.5. Система координирует выполнение всех действий, необходимых для обработки команды от декодирования до выполнения (например, в случае арифметических операций).

Все арифметические и логические операции выполняются в арифметико-логическом устройстве (АЛУ) и 32-х рабочих регистрах. АЛУ базовой серии микроконтроллеров AVR может выполнять операции сложения, вычитания, смещения, а также логические операции "И", "ИЛИ" и "Исключающее ИЛИ". Готовятся к выпуску микроконтроллеры AVR, арифметико-логическое устройство которых будет в состоянии выполнять также и операции умножения. АЛУ микроконтроллеров AVR является настолько мощным, что в течение единственного системного такта может извлечь из регистров два операнда, выполнить над ними операции и сохранить результат в регистре назначения.

Статическая память RAM (SRAM)

Оперативная память, допускающая запись и считывание переменных величин и данных, которые не должны сохраняться после отключения рабочего напряжения, в семействе микроконтроллеров AVR реализована в виде статической памяти RAM (SRAM). В противоположность динамической памяти RAM, эта технология не нуждается в регулярной регенерации, поэтому для некоторых случаев применения существует возможность уменьшить тактовую частоту вплоть до 0 Гц.

Память SRAM состоит из отдельных RS-триггеров, как это символически показано на рис. 3.1. Ячейки памяти расположены в форме матрицы, а декодирование выполняется по столбцам и строкам. Память триггерного типа, как это симво-

лически показано на рис. 3.1, состоит из двух КМОП-инверторов, образованных транзисторами V1, V2 и, соответственно, V3, V4. При записи бита данных сигнал через драйвер G1 попадает на разрядную шину B, преобразуется драйвером G2 и попадает на дополнительную разрядную шину /B.

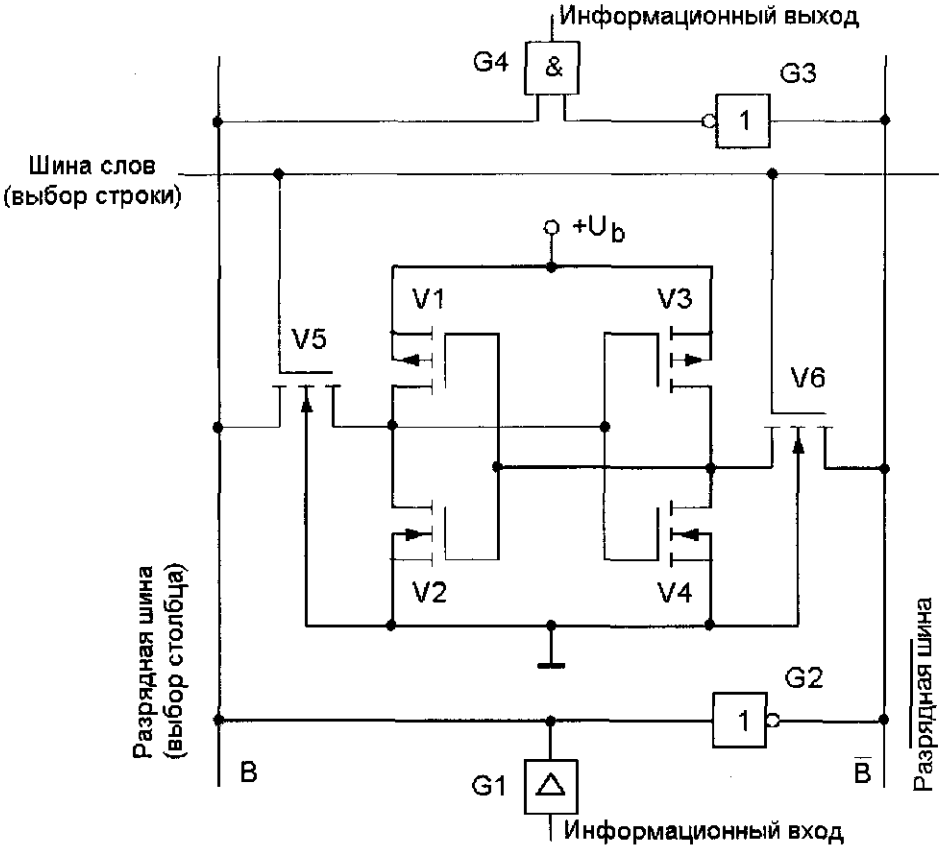


Рис. 3.1. Символическое представление ячейки статической памяти КМОП-RAM

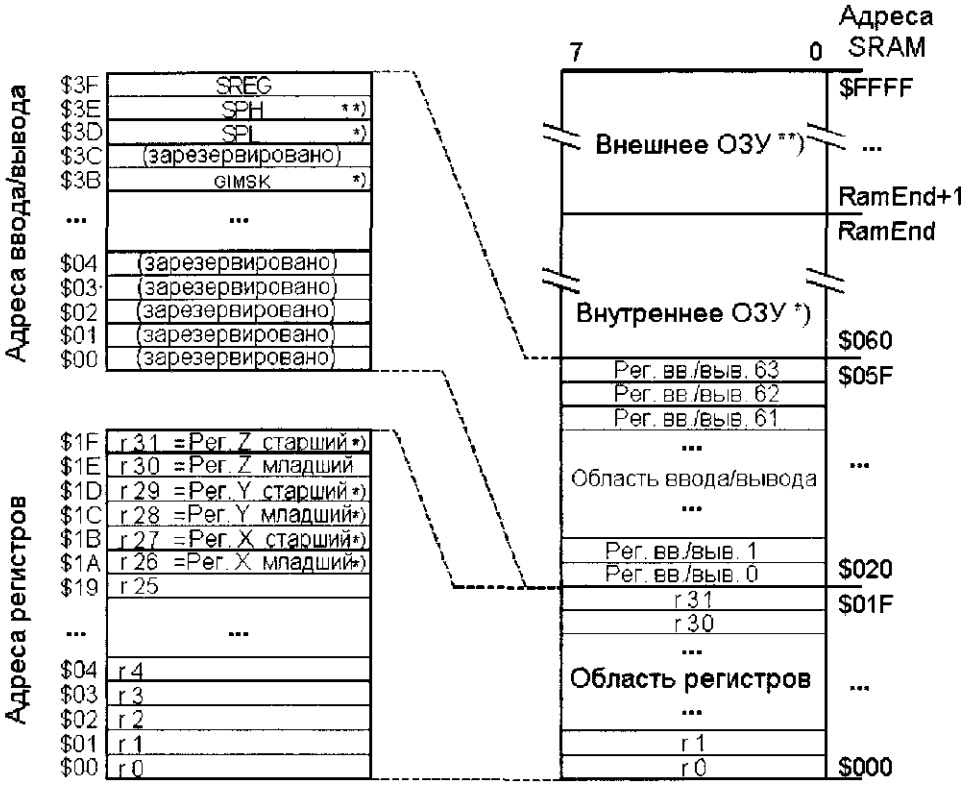
Если в процессе записи соответствующая ячейка памяти будет выбрана через лог. 1 на шине слов, то управление осуществляют два МОП-транзистора: V5 и V6. Если теперь, к примеру, необходимо записать в ячейку лог. 1, то эта "1" через V5 будет передана на вентили V3 и V4. Одновременно через V6 на вентилях V1 и V2 находится лог. 0 инвертированной разрядной шины, поэтому выход инверторов V1, V2 переходит в состояние лог. 1, и, аналогично, выход инверторов V3, V4 — в состояние лог. 0. Когда шина слов опять возвращается в состояние лог. 0, и строка больше не выбрана, то информация будет сохранена на соответствующем противоположном входе с помощью обратной связи выходов инвертора.

Для считывания записанной информации ячейка должна быть опять выбрана через лог. 1 на шине слов. Затем через подключенные МОП-транзисторы V5 и V6 содержание ячейки будет перенесено на разрядные шины B и /B. Если, например, в память записан лог. 0, то шина B содержит уровень 0, а шина /B — уровень 1, поэтому на выходе вентиля "И" G4, объединяющего обе шины, будет установлен

лог. 0. Если в память записана лог. 1, то, соответственно, шина B содержит уровень 1, а шина /B — уровень 0. В этом случае, в результате инвертирования уровня шины /B через G3, на оба входа вентиля AND G4 подается уровень лог. 1, а на его выходе появляется лог. 1.

Организация статической памяти SRAM семейства микроконтроллеров AVR

Статическое ОЗУ (SRAM) семейства микроконтроллеров AVR, как это показано на рис. 3.2, представляет собой блок со сквозной адресацией, состоящий из трех (на микроконтроллере AT90S1200 только двух) подобластей.



*) Нет в микроконтроллере AT90S1200

**) Нет в микроконтроллерах AT90S1200 и AT90S2313

Рис. 3.2. Организация модуля статического ОЗУ

Самый верхний адрес внутренней памяти SRAM, обозначенный на рис. 3.2 как "RamEnd", зависит от построения ОЗУ соответствующего микроконтроллера. В расположенной ниже таблице показаны наибольшие адреса памяти SRAM микроконтроллеров AVR базовой серии.

Микроконтроллер	AT90S1200	AT90S2313	AT90S4414	AT90S8515
Адрес RamEnd	$005F	$00DF	$015F	$025F

В микроконтроллерах AT90S4414 и AT90S8515 можно увеличивать пространство памяти SRAM посредством подключения внешних блоков памяти вплоть до 64 Кбайт, однако для этого приходится пожертвовать портами вывода A и C, которые в этом случае применяются в качестве шин передачи данных и адресов.

Все четыре представителя базовой серии микроконтроллеров AVR имеют область регистров (регистровый файл) и область ввода/вывода, однако в микроконтроллере AT90S1200 внутреннее статическое ОЗУ отсутствует, поэтому для запоминания переменных, применяемых при выполнении программы в AT90S1200, необходимо обращаться к 32-м рабочим регистрам.

Регистровый файл

Самая нижняя область памяти SRAM образует регистровый файл с 32 рабочими регистрами, которые все связаны с АЛУ и доступ к которым может быть выполнен в течение единственного такта системной синхронизации. Это означает, что за время такта в арифметико-логическое устройство вводятся два операнда из регистрового файла, выполняется операция, а результат запоминается в регистре назначения (рис. 3.3).

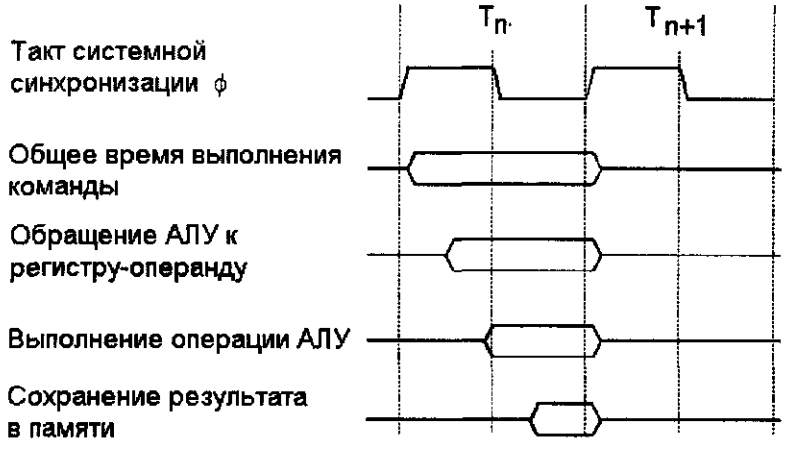


Рис. 3.3. Диаграмма выполнения однотактной операции

Несмотря на то, что рабочие регистры, с физической точки зрения, не являются ячейками памяти, им, как это показано на рис. 3.2, поставлены в соответствие 32 самых нижних адреса от $00 до $1F в памяти SRAM, поэтому их можно адресовать как обычные ячейки.

Регистры двойной длины X, Y и Z

Шесть рабочих регистров от R26 до R31 могут применяться в качестве регистров двойной длины X, Y и Z шириной 16 бит с возможностью доступа к обеим половинам (рис. 3.4).

Благодаря специальным командам, их можно использовать в качестве указателей, что при косвенной адресации позволяет очень эффективно обращаться к ячейкам памяти SRAM. Дополнительно к этому, с помощью регистра Z из памяти команд можно извлекать восьмиразрядные константы.

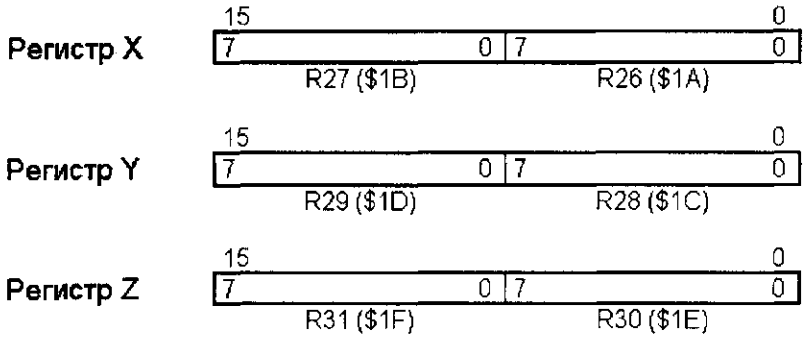


Рис. 3.4. Структура регистров двойной длины X, Y и Z

Применение регистров двойной длины X, Y и Z с возможностью доступа к обеим половинам подробно описано в пункте "Различные способы адресации команд и данных", а также в главе 12, "Система команд".

В связи с тем, что в микроконтроллерах AT90S1200 нет внутреннего статического ОЗУ, исключены также и регистры двойной длины X и Y. Здесь в распоряжении имеется только регистр R30, который выполняет роль восьмиразрядного указателя Z для косвенной адресации регистров (см. ниже соответствующий пункт). Регистры от R26 до R29, а также R31, естественно, могут применяться в качестве обычных рабочих регистров.

Область ввода/вывода

В этой области памяти SRAM размещены все регистры для программирования, управления и сигнализации о всех периферийных функциях микроконтроллеров AVR. В табл. 3.1 перечислены регистры ввода/вывода базовой серии микроконтроллеров AVR.

Таблица 3.1. Область ввода/вывода базовой серии микроконтроллеров AVR (зарезервированные или не использующиеся адреса не указаны)

Адрес	*Название*	*Функция*	*Начальн.*	*1200*	*2313*	*4414*	*8515*
$3F ($5F)	SREG	Регистр состояния	$00	+	+	+	+
$3E ($5E)	SPH	Указатель стека, старший байт	$00	—	—	+	+
$3D ($5D)	SPL	Указатель стека, младший байт	$00	—	+	+	+
$3B ($5B)	GIMSK	Общий регистр маски прерываний	$00	+	+	+	+
$3A ($5A)	GIFR	Общий регистр флагов прерываний	$00	—	+	+	+
$39 ($59)	TIMSK	Регистр маски прерываний таймера/счетчика	$00	+	+	+	+
$38 ($58)	TIFR	Регистр флагов прерываний от таймеров/счетчиков	$00	+	+	+	+
$35 ($55)	MCUCR	Регистр общего управления микроконтроллером	$00	+	+	+	+

Таблица 3.1. Продолжение

Адрес	Название	Функция	Начальн.	1200	2313	4414	8515
$33 ($53)	TCCR0	Регистр управления таймером/ счетчиком T/C0	$00	+	+	+	+
$32 ($52)	TCNT0	Счетный регистр таймера/счетчика T/C0 (8 бит)	$00	+	+	+	+
$2F ($4F)	TCCR1A	Регистр управления A таймера/счетчика T/C1	$00	—	+	+	+
$2E ($4E)	TCCR1B	Регистр управления B таймера/счетчика T/C1	$00	—	+	+	+
$2D($4D)	TCNT1H	Счетный регистр T/C1, старший байт	$00	—	+	+	+
$2C ($4C)	TCNT1L	Счетный регистр T/C1, младший байт	$00	—	+	+	+
$2B($4B)	OCR1AH	Регистр сравнения A таймера/счетчика T/C1, старший байт	$00	—	+	+	+
$2A ($4A)	OCR1AL	Регистр сравнения A таймера/счетчика T/C1, младший байт	$00	—	+	+	+
$29 ($49)	OCR1BH	Регистр сравнения B таймера/счетчика T/C1, старший байт	$00	—	—	+	+
$28 ($48)	OCR1BL	Регистр сравнения B таймера/счетчика T/C1, младший байт	$00	—	—	+	+
$25 ($45)	ICR1H	Регистр захвата таймера/ счетчика T/C1, старший байт	$00	—	+	+	+
$24 ($44)	ICR1L	Регистр захвата таймера/ счетчика T/C1, младший байт	$00	—	+	+	+
$21 ($41)	WDTCR	Регистр управления сторожевым таймером	$00	+	+	+	+
$1F ($3F)	EEARH	Регистр адреса EEPROM, старший байт	$00	—	—	—	+
$1E ($3E)	EEAR(L)	Регистр адреса EEPROM, младший байт	$00	+	+	+	+
$1D ($3D)	EEDR	Регистр данных EEPROM	$00	+	+	+	+
$1C ($3C)	EECR	Регистр управления EEPROM	$00	+	+	+	+
$1B ($3B)	PORTA	Регистр данных порта A	$00	—	—	+	+
$1A ($3A)	DDRA	Регистр направления передачи данных порта A	$00	—	—	+	+
$19 ($39)	PINA	Выводы порта A	Hi-Z	—	—	+	+
$18 ($38)	PORTB	Регистр данных порта B	$00	+	+	+	+
$17 ($37)	DDRB	Регистр направления передачи данных порта B	$00	+	+	+	+
$16 ($36)	PINB	Выводы порта B	Hi-Z	+	+	+	+

Таблица 3.1. Окончание

Адрес	*Название*	*Функция*	*Начальн.*	*1200*	*2313*	*4414*	*8515*
$15($35)	PORTC	Регистр данных порта C	$00	—	—	+	+
$14 ($34)	DDRC	Регистр направления передачи данных порта C	$00	—	—	+	+
$13 ($33)	PINC	Выводы порта C	Hi-Z	—	—	+	+
$12 ($32)	PORTD	Регистр данных порта D	$00	+	+	+	+
$11($31)	DDRD	Регистр направления передачи данных порта D	$00	+	+	+	+
$10 ($30)	PIND	Выводы порта D	Hi-Z	+	+	+	+
$0P ($2F)	SPDR	Регистр ввода/вывода данных SPI	$00	—	—	+	+
$0E ($2E)	SPSR	Регистр состояния SPI	$00	—	—	+	+
$0D ($2D)	SPCR	Регистр управления SPI	$04	—	—	+	+
$0C ($2C)	UDR	Регистр данных UART	$00	—	+	+	+
$0B ($2B)	USR	Регистр состояния UART	$20	—	+	+	+
$0A ($2A)	UCR	Регистр управления UART	$00	—	+	+	+
$09 ($29)	UBRR	Регистр скорости передачи данных UART	$00	—	+	+	+
$08 ($28)	ACSR	Регистр управления и состояния аналогового компаратора	$00	+	+	+	+

Столбец "Начальн." в табл. 3.1 указывает на значение, которое будет записано в соответствующий регистр после поступления сигнала сброса.

К регистрам ввода/вывода относятся регистры разрешения/запрета отдельных прерываний, а также указатель стека, регистры состояния для указания результатов арифметических и логических операций, регистры управления работой многочисленных компонентов аппаратного обеспечения и периферии (порты, таймер, UART, интерфейс SPI, аналоговый компаратор, сторожевой таймер, режимы пониженного энергопотребления), а также регистры для обращения к памяти EEPROM.

Для доступа к регистрам ввода/вывода лучше всего использовать команды ввода/вывода AVR `in` и `out`. Команда вывода `out` выдает один байт из одного из 32-х рабочих регистров в регистр ввода/вывода; команда ввода `in` считывает байт из одного регистра ввода/вывода в один из 32-х рабочих регистров.

Области ввода/вывода назначены адреса с $00 по $3F, которые должны применяться в случае использования специальных команд `in`, `out`, `sbi`, `cbi`, `sbis` и `sbic`.

Если регистр ввода/вывода адресуется как область в памяти SRAM, то к адресу ввода/вывода необходимо добавить $20. Все адреса регистров ввода/вывода в настоящей книге указываются вместе с соответствующими адресами в памяти SRAM в скобках (см. табл. 3.1).

Таким образом, при использовании двух последовательностей команд, представленных ниже, результат будет одинаковым.

Первая последовательность команд:

```
Адрес   Код     Команда
000000  ea0a    ldi r16, 0xAA ; Загрузить $AA в r16
000001  bb02    out 0x12, r16 ; Вывести <r16> в порт D
                              ; по адресу ввода/вывода
```

Вторая последовательность команд:

```
Адрес   Код         Команда
000000  ea0a        ldi r16, 0xAA  ; Загрузить $AA в r16
000001  9300 0032   sts 0x32, r16  ; Вывести <r16> в порт
                                   ; D по адресу SRAM
```

В обоих случаях в порт D выводится число $AA, поскольку порт D имеет адрес ввода/вывода $12 и адрес SRAM $32 (сравните с табл. 3.1).

У регистров ввода/вывода в области с $00 по $1F ($20 ... $3F) отдельные разряды могут быть изменены напрямую с помощью команд `sbi` и `cbi`. Кроме того, содержимое отдельных разрядов этих регистров может быть напрямую опрошено с помощью команд `sbis` и `sbic`. Более подробно об этом можно узнать из главы 12, "Система команд". Различные регистры ввода/вывода и периферийные регистры управления будут подробно рассмотрены в соответствующих главах.

Регистр состояния (SREG)

Регистр состояния содержит биты условий (флаги) микроконтроллеров семейства AVR и располагается в области ввода/вывода по адресу $3F ($5F). Он доступен для чтения и записи и после подачи сигнала сброса инициализируется нулями.

Разряд	7	6	5	4	3	2	1	0	
$3F($5F)	I	T	H	S	V	N	Z	C	SREG

В микроконтроллерах семейства AVR для обозначения результата выполнения операций используются восемь различных битов условий. Все они могут быть проверены с помощью команд, определяющих последующий ход выполнения программы (например, `brcs`, `brhc`, `brtc`, `brie` и т.д.).

При входе в подпрограмму обработки прерывания рекомендуется сохранить регистр состояния и снова его восстановить при выходе для того, чтобы после возврата в прерванную программу работать с корректными битами условий.

Значения отдельных битов условий (флагов) в микроконтроллерах семейства AVR:

- разряд **0** — **C** — флаг переноса (Carry) — указывает на переполнение (перенос) после выполнения арифметической или логической операции;
- разряд **1** — **Z** — нулевой флаг (Zero) — всегда устанавливается, если результат арифметической или логической операции равен нулю; сбрасывается, если результат операции не равен нулю;
- разряд **2** — **N** — флаг отрицательного результата (Negative) — указывает на отрицательный результат после выполнения арифметической или логической операции;

- разряд **3** — **V** — флаг переполнения при вычислениях в дополнительных кодах (Two's complement Overflow) — поддерживает арифметику дополнительных кодов (арифметика кодов с дополнением до двух); устанавливается, если при выполнении соответствующей операции происходит переполнение, в противном случае — сбрасывается;
- разряд **4** — **S** — флаг знака (Sign) — S = N⊕V — связь флагов N и V с помощью операции "Исключающее ИЛИ"; флаг знака может применяться для определения фактического результата арифметической операции;
- разряд **5** — **H** — флаг половинного переноса (Half Carry) — указывает на переполнение в младшем полубайте (биты 0...3 байта данных); устанавливается, когда происходит перенос из младшего полубайта в старший, в противном случае — сбрасывается;
- разряд **6** — **T** — флаг копирования (Transfer or Copy) — предназначен для свободного применения программистом (например, в качестве буфера);
- разряд **7** — **I** — общее разрешение прерываний (Global Interrupt) — если прерывания, как таковые, должны быть разрешены, то должен быть установлен разряд 7 регистра состояния (в лог. 1).

Правила установки флагов подробно рассматриваются в главе 12, "Система команд".

Регистр управления MCU (MCUCR)

Регистр управления MCU (**M**icrocontroller **U**nit — модуль микроконтроллера) содержит разряды управления общими функциями MCU. Наряду с разрешением доступа к внешней памяти RAM, регистр MCUCR также управляет "спящим" режимом и характером срабатывания внешних прерываний.

Регистр MCUCR находится в области ввода/вывода по адресу $35 ($55) и доступен для чтения и записи. После подачи сигнала сброса он инициализируется нулями.

Разряд	7	6	5	4	3	2	1	0	
$35($55)	SRE **)	SRW **)	SE	SM	ISC11 *)	ISC10 *)	ISC01	ISC00	MCUCRR

*) — в AT90S1200 отсутствует

**) — в AT90S1200 и AT90S2313 отсутствует

Если разряд **SRE** (External **S**RAM **E**nable) установлен в лог. 1, то разрешен доступ к внешней памяти RAM; а порты A, C (а также выводы PD6 и PD7 порта D) будут переключены в их альтернативные функции AD0...7, A8...15, /WR и /RD. Разряды направления передачи данных, соответствующие выводам регистров направления передачи данных, с помощью разряда SRE будут обновлены согласно их новым функциям. Подробности работы внешней памяти SRAM описаны в разделе "Внешняя память SRAM". Если разряд SRE содержит лог. 0, то внешняя память SRAM недоступна, и все порты и разряды направления передачи данных выполняют свои обычные функции.

Если в разряд **SRW** (External **S**RAM **W**ait State) записана лог. 1, то в последовательность обращения к внешней памяти SRAM будет добавлено

состояние ожидания (Wait State) продолжительностью один системный такт. Если разряд SRW содержит лог. 0, то доступ к внешней памяти RAM выполняется за два обычных тактовых цикла. Эксплуатация внешней памяти SRAM с состоянием ожидания и без него подробно описана в разделе "Внешняя память SRAM".

Разряд **SE** (**S**leep **E**nable) должен быть установлен в лог. 1 для того, чтобы сделать эффективной команду `sleep`. Если этот разряд содержит лог. 0, то после выполнения команды `sleep` центральный процессор (ЦП) останется активным. Во избежание непреднамеренного перевода процессора в ждущий режим или в режим пониженного энергопотребления непосредственно перед выполнением команды `sleep` рекомендуется установить разряд SE.

С помощью разряда **SM** (**S**leep **M**ode — "Спящий режим") осуществляется выбор между двумя возможными состояниями пониженного энергопотребления: "Idle" и "Power Down". Если разряд SM содержит лог. 0, то ЦП микроконтроллера после выполнения команды `sleep` переходит в ждущий режим ("Idle"), если же разряд SM содержит лог. 1, то ЦП микроконтроллера перейдет, соответственно, в режим пониженного энергопотребления ("Power Down"). Подробнее оба режима описаны в разделе "'Спящие' режимы центрального процессора".

Разряды **ISC11** и **ISC10** (**I**nterrupt **S**ense **C**ontrol **1**, разряды 1 и 0) определяют способ активизации обработки внешнего прерывания 1, когда в регистре SREG установлен разряд I, а в регистре GIMSK — разряд INT1: по нарастающему или ниспадающему фронту сигнала, или же по уровню лог. 0 на выводе INT1. В табл. 3.2. показаны возможные комбинации ICS11 и ICS10

Таблица 3.2. Выбор способа активизации прерывания INT1

ISC11	*ISC10*	*Описание*
0	0	Прерывание INT1 вызывается по уровню лог. 0
0	1	Зарезервировано
1	0	Прерывание INT1 вызывается по нарастающему фронту сигнала
1	1	Прерывание INT1 вызывается по ниспадающему фронту сигнала

Если разряды ISC11 и ISC10 должны быть изменены, то сначала необходимо запретить прерывание 1 посредством сброса разряда INT1 в регистре GIMSK, в противном случае, непреднамеренное изменение этих разрядов может привести к вызову внешнего прерывания 1.

Разряды ISC01 и ISC00 (**I**nterrupt **S**ense **C**ontrol **0**, разряды 1 и 0) устанавливают способ активизации внешнего прерывания 0, когда в регистре SREG будет установлен разряд I, а в регистре GIMSK — разряд INT0: по нарастающему или ниспадающему фронту сигнала, или же по уровню лог. 0 на выводе INT0.

В табл. 3.3. показаны возможные комбинации ICS01 и ICS00.

Таблица 3.3. Выбор способа активизации прерывания INT0

ISC01	*ISC00*	*Описание*
0	0	Прерывание INT0 вызывается по уровню лог. 0
0	1	Зарезервировано
1	0	Прерывание INT0 вызывается по ниспадающему фронту сигнала
1	1	Прерывание INT1 вызывается по нарастающему фронту сигнала

Если разряды ISC01 и ISC00 должны быть изменены, то сначала необходимо запретить прерывание 0 посредством сброса разряда INT0 в регистре GIMSK, в противном случае, непреднамеренное изменение этих разрядов может привести к вызову внешнего прерывания 0.

Внутренняя память SRAM

Данные и переменные, используемые в программах, как правило, хранятся во внутренней памяти SRAM микроконтроллеров AVR. К этим данным можно обращаться из программы прямо или косвенно (см. раздел "Различные способы адресации команд и данных"). В памяти SRAM также может быть размещен стек — исключение здесь составляет только микроконтроллер ATS90S1200 со своим аппаратным стеком (см. пункт "Стек базовой серии микроконтроллеров AVR"). На рис. 3.5 символически показаны графики синхронизации при обращении ко внутренней памяти SRAM.

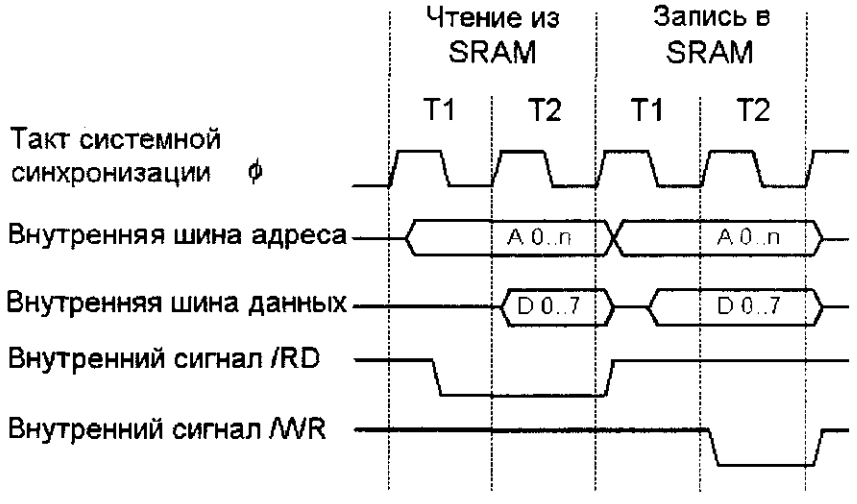


Рис. 3.5. Синхронизация при обращении к внутренней памяти SRAM

При считывании байта данных из памяти SRAM в рабочий регистр сначала через встроенную шину адреса вводится желаемый адрес SRAM. Количество *n* адресных разрядов зависит от построения запоминающего устройства того или иного микроконтроллера. После этого подается внутренний сигнал чтения (/RD = лог. 0). Затем содержимое ячейки памяти SRAM D0...D7, появившееся на внутренней шине данных, переносится по нарастающему фронту сигнала /RD в желаемый рабочий регистр.

Если содержимое рабочего регистра должно быть записано в память SRAM, то сначала желаемый адрес SRAM подается на встроенную шину адреса, а содержимое рабочего регистра — на встроенную шину данных. Вслед за этим подается внутренний сигнал записи (/WR = лог. 0), по нарастающему фронту которого байт с шины данных записывается в желаемую ячейку памяти.

Внешняя память SRAM

Если объема внутренней памяти SRAM недостаточно, то в микроконтроллерах типов AT90S4414 и AT90S8515 его можно увеличить до 64 Кбайт посредством подключения внешних блоков памяти. Для этого в регистре MCUCR (адрес в области ввода/вывода — $35, адрес в SRAM — $55) следует установить разряд SRE (разряд 7). После установки этого разряда порты PA (AD 0...7) и PC (A8...15) будут выступать в качестве шины адреса и шины данных, а выводы 7 и 6 порта PD — в качестве управляющих сигналов чтения /RD- и, соответственно, записи /WR для внешней памяти SRAM (см. рис. 3.6), независимо от того, какие направления передачи данных установлены для этих портов в соответствующих регистрах направления передачи данных. Дальнейшее обращение к внешней статической памяти может осуществляться с помощью тех же команд (`ld`, `st`, `lds`, `sts`, `ldd`, `std`, `push`, `pop`), что и при обращении к внутренней памяти.

При обращении к адресам SRAM, которые находятся в области внутренней статической памяти (то есть, по адресам с $000 по RamEnd на рис. 3.2), передача данных выполняется автоматически как во внутреннюю память SRAM, так и из нее, даже при установленном разряде SRE. В этом случае управляющие сигналы чтения /RD и записи /WR остаются неактивными на протяжении всего цикла. Обращение к внешней памяти будет выполнено только тогда, когда адрес будет больше, чем RamEnd, а также если в регистре управления MCUCR будет установлен разряд SRE.

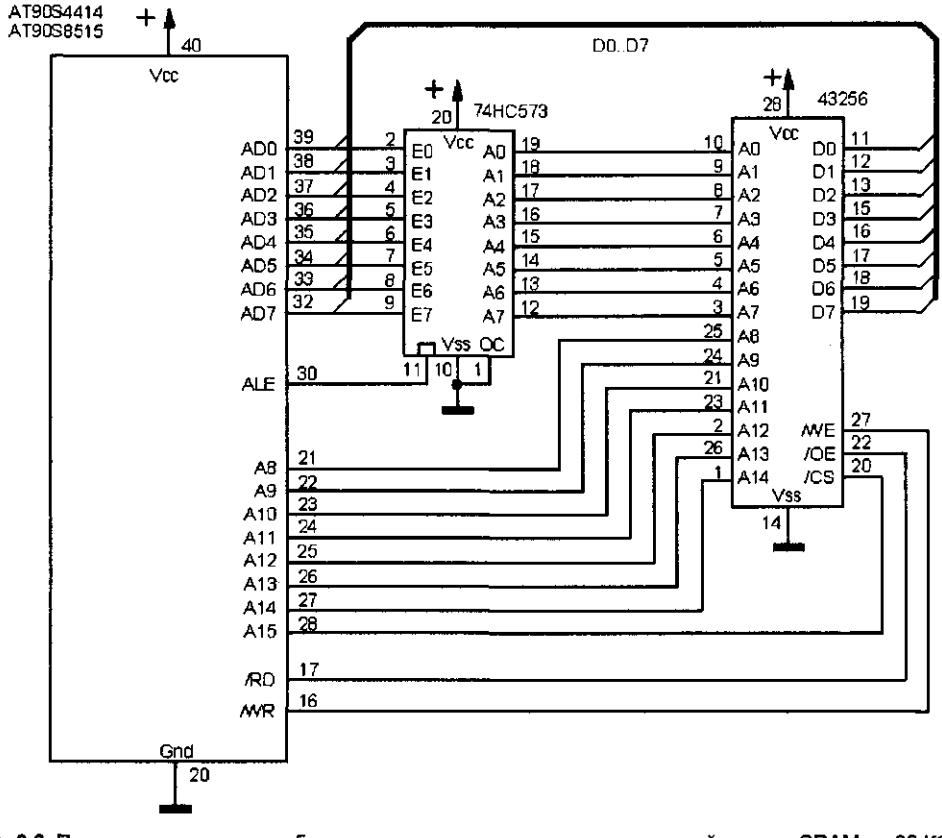


Рис. 3.6. Подключение внешнего блока памяти для расширения статической памяти SRAM до 32 Кбайт

С целью упрощения, на рис. 3.6 не показаны все детали, необходимые для расширения статической памяти SRAM.

Для начала нового цикла передачи на порт A, который функционирует в качестве мультиплексированной шины адреса/данных, передаются восемь младших разрядов адреса A0..A7. По ниспадающему фронту сигнала ALE (Address Latch Enable) они сохраняются в 8-разрядном фиксирующем регистре 74HCT573. Параллельно, через порт C выдаются восемь старших разрядов адреса A8..A15 (см. рис. 3.6 и рис. 3.7).

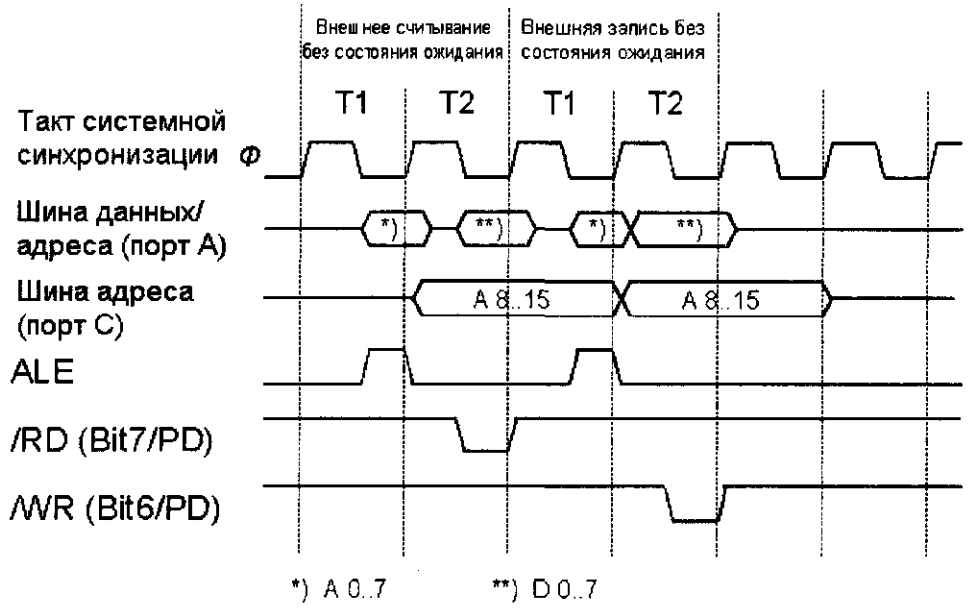


Рис. 3.7. Обращение к внешней памяти SRAM без состояния ожидания (разряд 6 в регистре управления MCUCR по адресу ввода/вывода $35 ($55) сброшен)

Для чтения с учетом определенного времени обращения, пока адрес будет устойчиво находиться на входах внешнего блока памяти 43256, активизируется сигнал чтения /RD, после чего блок памяти SRAM выдает адресуемый байт на порт A шины данных. По нарастающему фронту сигнала /RD байт затем записывается в рабочий регистр, определенный в ассемблерной команде.

Если содержимое рабочего регистра необходимо скопировать в SRAM, то после вывода мультиплексированного младшего адресного байта содержание этого регистра выводится на шину данных порта A. Когда после короткой задержки этот байт данных стабильно установится на входах данных SRAM, подается сигнал записи (/WR = лог. 0), и по его нарастающему фронту происходит запись байта с шины данных в адресуемую ячейку памяти.

На рис. 3.8 символически показана синхронизация при выполнении команды чтения, следующей сразу же после команды записи. Последовательность команд могла быть, например, такой:

```
ld  r0,x
st  Y,r1
```

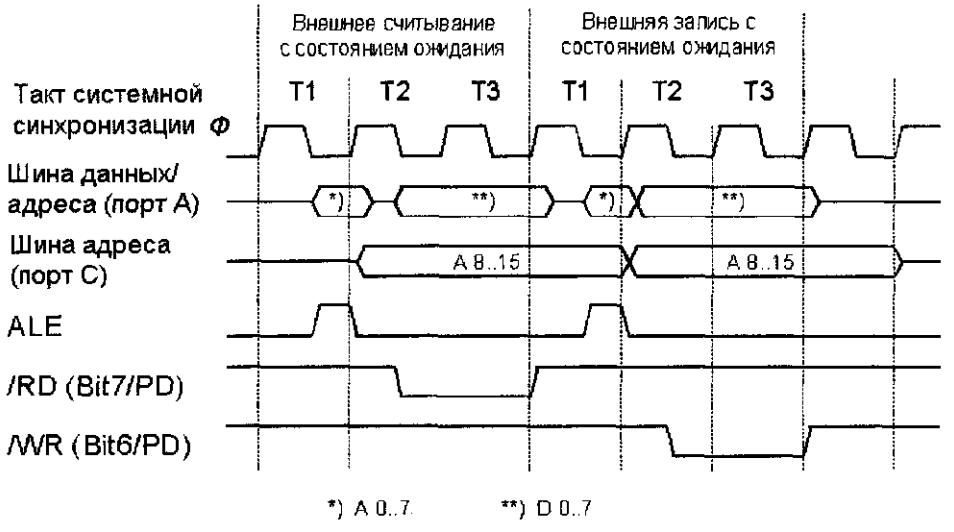


Рис. 3.8. Обращение к внешней памяти SRAM с состоянием ожидания (в регистре управления MCUCR по адресу ввода/вывода $35 ($55) установлен разряд 6)

На тот случай, если используемые внешние блоки статической памяти окажутся недостаточно быстродействующими для циклов записи/чтения микроконтроллера AVR, установив разряд SRW (разряд 6) в регистре MCUCR можно активизировать состояние ожидания.

Благодаря введению этого дополнительного состояния продолжительностью в один импульс такта системной синхронизации, длительность сигналов передачи данных и команд /RD и, соответственно, /WR увеличивается для того, чтобы предоставить достаточно времени для передачи данных блокам памяти, работающим медленнее.

Благодаря чрезвычайно коротким маршрутам прохождения сигналов внутри кристалла, обращение к внутренней памяти RAM в значительной степени более свободно от сбоев, чем обращение к внешней памяти через шину. Если при этом принять во внимание, что из-за подсоединения внешнего запоминающего устройства теряются 18 выводов, то следует тщательно взвешивать: действительно ли подсоединение внешней памяти — это наилучшее решение для данного случая применения.

Стек базовой серии микроконтроллеров AVR

Стек служит для хранения адресов возврата при вызове подпрограмм, а также для передачи параметров в подпрограммы. Кроме того, зачастую в стеке, как в буфере, временно хранятся переменные и промежуточные результаты.

В случае микроконтроллеров AT90S4414 и AT90S8515, стек по выбору может быть расположен во внутренней или внешней памяти RAM, а в микроконтроллере AT90S2313 он всегда находится во внутренней памяти RAM. В микроконтроллере AT90S1200, который не имеет явной памяти данных, применяется трехуровневый аппаратный стек (действует по принципу "последним вошел — первым вышел"), предназначенный для вызова подпрограмм и обработки прерываний.

Указатель стека

Прежде всего рассмотрим особый случай микроконтроллера AT90S1200. Здесь по причине аппаратного стека программа пользователя не управляет указателем стека. Аппаратный стек, как и счетчик команд микроконтроллера AT90S1200, имеет ширину 9 разрядов. При выполнении команды `rcall` или запросе на прерывание все адреса, введенные в аппаратный стек, смещаются на один уровень вниз (адрес, помещенный до этого на уровень 2, теряется, а находящийся на уровне 1 перемещается на уровень 2; адрес, ранее записанный на уровне 0, будет перемещен на уровень 1). После этого соответствующий адрес возврата помещается на уровень 0 аппаратного стека.

Поясним это на примере следующей программы:

```
.device AT90S1200

000000 d13f      Start:   rcall  UP1
000001 cffe               rjmp   Start

.org 0x140
000140 d00f      UP1:     rcall    UP2
000141 9508               ret

.org 0x150
000150 d00f      UP2:     rcall    UP3
000151 9508               ret

.org 0x160
000160 9508      UP3:     ret
```

Аппаратный стек перед выполнением команды `rcall UP1`:

Уровень 0	???
Уровень 1	???
Уровень 2	???

Аппаратный стек после выполнения команды `rcall UP1`:

Уровень 0	$001
Уровень 1	???
Уровень 2	???

Аппаратный стек после выполнения команды `rcall UP2`:

Уровень 0	$141
Уровень 1	$001
Уровень 2	???

Аппаратный стек после выполнения команды `rcall UP3`:

Уровень 0	$151
Уровень 1	$141
Уровень 2	$001

При выходе из обычной подпрограммы или подпрограммы обработки прерывания адрес, находящийся на нулевом уровне, извлекается в счетчик команд как адрес возврата, и все уровни аппаратного стека перемещаются на одну позицию вверх (уровень 1 — на уровень 0, уровень2 — на уровень 1).

Аппаратный стек перед выполнением команды `ret` в `UP3`:

Уровень 0	$151
Уровень 1	$141
Уровень 2	$001

Аппаратный стек после выполнения команды `ret` в `UP3`:

Уровень 0	$141
Уровень 1	$001
Уровень 2	???

Аппаратный стек после выполнения команды `ret` в `UP2`:

Уровень 0	$001
Уровень 1	???
Уровень 2	???

Аппаратный стек после выполнения команды `ret` в `UP1`:

Уровень 0	???
Уровень 1	???
Уровень 2	???

При глубине вложения вызовов 4 (и более) подпрограмм необходимо соблюдать осторожность, поскольку в этом случае теряется адрес возврата, введенный в память первым, вследствие чего возврат становится невозможным!

Микроконтроллеры моделей AT90S4414 и AT90S8515, которые могут адресовать до 64 Кбайт адресов внешней памяти, имеют 16-разрядный указатель стека, состоящий из двух отдельных 8-разрядных регистров SPH и SPL в области ввода/вывода. Старший байт SPH (разряды SP8...SP15) находится в памяти по адресу $3E ($5E), а младший байт SPL (разряды SP0...SP7) — по адресу $3D ($5D). Указатель стека после подачи сигнала сброса устанавливается в "0". Он доступен для чтения и записи.

Разряд	7	6	5	4	3	2	1	0	
$3E ($5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
$3D ($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL

Для микроконтроллера AT90S2313 достаточно указателя стека шириной 8 разрядов — ячейка по адресу $3D ($5D) в области ввода/вывода — поскольку этот тип отображает свой стек исключительно во внутренней памяти SRAM объемом 128 байт с адреса $60 по $DF. Указатель стека после подачи сигнала сброса переходит в нулевое состояние. Он также доступен для чтения и записи.

Разряд	7	6	5	4	3	2	1	0	
$3D ($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL

Указатель стека позволяет обращаться к сегменту данных в области, зарезервированной для стека. Во всех микроконтроллерах AVR, за исключением AT90S1200, он должен быть инициализирован до начала выполнения программы прежде, чем будет выполнена подпрограмма или будет разрешено прерывание, поскольку указатель стека после подачи сигнала сброса будет снова установлен в "0". В микроконтроллерах семейства AVR стек заполняется "сверху вниз", то есть, при выполнении команды push соответствующий байт будет скопирован по адресу, на который в настоящее время ссылается указатель стека, после чего указатель стека уменьшается на 1 (декрементируется). Это целесообразно потому, что при инициализации до начала выполнения программы указатель стека позволяет ссылаться на ячейку памяти с наибольшим адресом.

Пример инициализации указателя стека в микроконтроллере AT90S8515 (наибольший адрес в области данных — $25F):

```
.equ    SPH     = $3E          ; Определения в файле
.equ    SPL     = $30          ; 8515def.inc
.equ    RAMEND  = $25F

.cseg                          ; Сегмент кода
.org 80h                       ; Начало области программы
init:   ldi r16, low(RAMEND)
out SPL, r16
ldi r16, high(RAMEND)
out SPH, r16                   ; Указатель стека инициализирован
```

Когда байт с помощью команды push помещается в стек, то он записывается по адресу, на который сейчас ссылается указатель стека, после чего указатель стека декрементируется на 1.

Когда байт с помощью команды pop извлекается из стека, то сначала указатель стека инкрементируется на 1, а затем байт, извлеченный по соответствующему адресу, копируется в регистр назначения.

Рассмотрим следующий пример:

```
Адрес   Код    Метка   Команда
000000  e50f   Start:  ldi    r16, 0x5F
000001  bf0d           out    SPL, r16
000002  e002           ldi    r16, 0x02
000003  bf0e           out    SPH, r16  ; SP указывает на
                                         ; адрес $025F
000004  ee07           ldi    r16, 0xE7 ; r16 изменен
000005  930f           push   r16       ; r16 помещается в стек
000006  e30c           ldi    r16, 0x3C ; r16 символически изменен
000007  910f           pop    r16       ; r16 извлекается из стека
000008  cff7           rjmp   Start     ; Цикл завершен
```

В этом примере указатель стека инициализируется в командах с адресами от $000 до $003 включительно. После этого в r16 загружается значение $E7. После выполнения команды push (адрес $005) ячейка памяти $025F в сегменте данных содержит $E7. Указатель стека декрементируется на 1 до $025E. С помощью следующей команды содержимое r16 символически (вместо подлежащей выполнению последовательности команд) изменяется на $3C. Затем команда pop (адрес

$007) увеличивает (инкрементирует) на 1 указатель стека, после чего в r16 записывается содержимое адреса $025F в сегменте данных, по которому ссылается указатель стека — теперь r16 опять содержит $E7.

В табл. 3.4 показано содержимое отдельных регистров, а также ячеек памяти после обработки соответствующей команды.

Таблица 3.4. Содержимое регистров **после** выполнения команд из рассмотренной выше программы

Команда	*PC*	*SP*	*r16*	*Адрес $025F*
ldi r16, 0x5F	001	$????	$5F	$??
out SPL, r16	002	$??5F	$5F	$??
ldl r16, 0x02	003	$??5F	$02	$??
out SPH, r16	004	$025F	$02	$??
ldl r16. 0xE7	005	$025F	$E7	$??
push r16	006	$025E	$E7	$E7
ldl r16, 0x3C	007	$025E	$3C	$E7
pop r16	008	$025F	$E7	$E7
rjmp start	000	$025F	$E7	$E7

При прерывании или вызове подпрограммы (call) в качестве адреса возврата в стек будет помещен адрес следующей команды из выполняемой программы.

В представленном ниже примере показан соответствующий процесс выполнения программы:

```
'.cseg
.org 0x480

Адрес    Код    Метка  Команда
000480   e50f   Start: ldi    r16, 0x5F
000481   bf0d          out    SPL, r16
000482   e002          ldi    r16, 0x02
000483   bf0e          out    SPH, r16  ; SP указывает на адрес $025F
000484   d07b          rcall  UP1       ; Обращение к UP1
000485   cffa          rjmp   Start     ; Завершить цикл
                                        ;
.org     0x500
000500   9508   UP1:   ret              ; Сразу же возврат
```

Команды с адресами с $480 по $483 включительно инициализируют указатель стека ($025F). Следующая команда rcall (адрес $484) записывает младший байт адреса возврата ($85) по адресу в памяти данных, на который в этот момент ссылается указатель стека, то есть, — по адресу $025F. Указатель стека декрементируется на 1 — теперь он указывает на адрес $025E. Теперь в эту ячейку памяти записывается старший байт адреса возврата ($04). Указатель стека еще раз декрементируется на 1, теперь он указывает на адрес $025D. В завершение команда rcall устанавливает счетчик команд на адрес подпрограммы UP1 ($500).

В этом примере из подпрограммы UP1 с помощью команды ret сразу же происходит возврат к вызывающей программе. Команда ret увеличивает указатель стека на 1. Теперь он ссылается на адрес $025E в области данных. Хранящийся по этому адресу байт ($04) копируется в старший байт счетчика команд, после чего указатель стека увеличивается на 1. Теперь он указывает на адрес $025F. Байт

($85), хранящийся по этому адресу, копируется в младший байт счетчика команд. Итак, после выполнения команды `ret` ход программы будет продолжен, начиная с адреса $485, то есть, — с адреса команды `rjmp`, следующей после команды `rcall`.

Команда `reti` выполняет ту же функцию, что и команда `ret`, с тем отличием, что она применяется для возврата из подпрограммы обработки прерывания и дополнительно к этому устанавливает флаг I (разряд 7) в регистре состояния SREG для того, чтобы снова разрешить дальнейшие прерывания.

Память команд (технология Flash-EPROM)

Отличительной особенностью микроконтроллеров семейства AVR является то, что у них в качестве памяти программ используется одна и та же память типа Flash-EPROM, которая в зависимости от обстоятельств может быть разного объема (см. табл. 1.1), свободно программироваться пользователем и опять вытираться электрическим способом.

Такие первые представители однокристальных микроконтроллеров как 8048 и 8051 компании Intel с их производными можно было приобрести или с памятью типа MROM (программно-маскируемая ROM), или со встроенной памятью EPROM. Альтернативно, они могли работать также и с внешней памятью EPROM. Модели с памятью типа MROM имели один недостаток: их мог программировать только завод-изготовитель в рамках производственного процесса, а вытереть данные было невозможно. Микроконтроллеры с памятью EPROM имели в корпусе окошко с кварцевым стеклом для вытирания информации, однако их отрицательной стороной была высокая цена.

В микроконтроллерах с отдельной памятью EPROM, наряду с собственно памятью, как правило, применена также 8-разрядная адресный фиксатор. Не следует также забывать и о том, что в этом случае требуются драгоценные 18 контактов ввода/вывода для адресной шины и шины данных, а также для управляющих сигналов /RD и /WR.

Микроконтроллеры семейства AVR со встроенной флэш-памятью EPROM экономят не только место на печатной плате, но также предоставляют в распоряжение пользователя все контакты ввода/вывода микроконтроллера. Также может отпасть необходимость и в колодке для внешней памяти EPROM.

Наряду с возможностью программирования всех модулей микроконтроллеров семейства AVR в параллельном режиме, компания Atmel предоставляет в распоряжение пользователей очень эффективную возможность последовательного программирования через последовательный интерфейс SPI. В результате программа пользователя может быть "записана" во флэш-память и снова вытерта непосредственно в составе схемы, в которой будет работать микроконтроллер. Схема программирования уже интегрирована в кристалл. Вспомогательное напряжение V_{pp} = +12 В необходимо только при программировании в параллельном режиме. Так или иначе, это напряжение используется в большинстве электрических схем (например, для последовательного интерфейса).

Физическая организация флэш-памяти базовой серии микроконтроллеров семейства AVR

Память команд в микроконтроллерах базовой серии семейства AVR представляет собой флэш-память EPROM и во всех четырех моделях состоит из *n* 16-разрядных (2 байта) слов. В самом мощном из микроконтроллеров — AT90S8515 — счетчик команд имеет ширину 12 разрядов и, таким образом, может обрабатывать *n* = 4096 ($1000) — 4 Кбайт командных 16-разрядных слов. В микроконтроллере AT90S4414 он имеет ширину 11 разрядов, что соответствует *n* = 2048 ($800) командных слов. Микроконтроллер AT90S2313 обладает счетчиком команд шириной 10 разрядов (*n* = 1024 и, соответственно, $400), а микроконтроллер AT90S1200 со своим счетчиком команд шириной 9 разрядов может адресовать 512 ($200) команд.

Распределение памяти команд в четырех основных типах микроконтроллеров AVR показано на рис. 3.9.

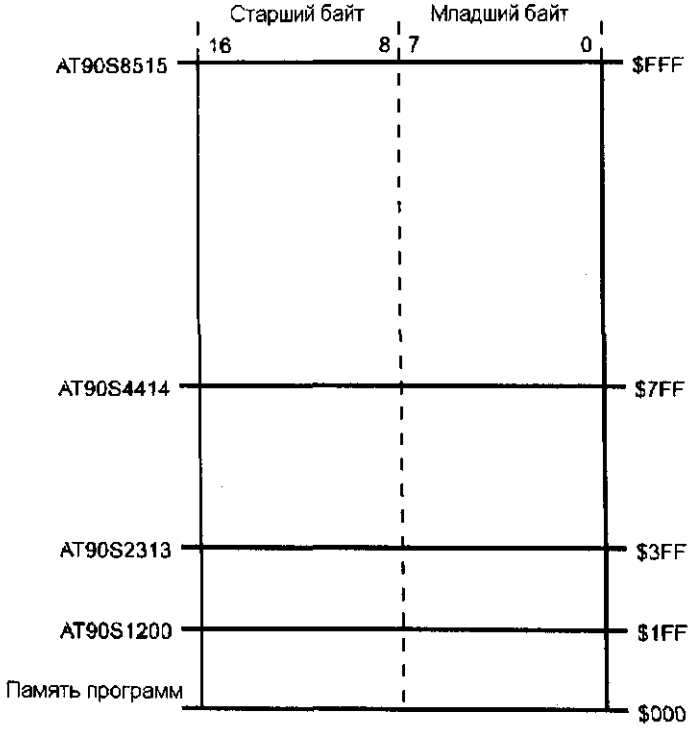


Рис. 3.9. Размер и организация памяти команд по технологии флэш-памяти EPROM

Технологии памяти Flash-EPROM

Чтобы лучше понимать процессы при программировании флэш-памяти микроконтроллеров семейства AVR, необходимо сделать небольшое введение в технологию различных имеющихся в распоряжении типов памяти EPROM.

Как и в "обычных" EPROM (**E**rasable **PROM** — вытираемая память PROM), которую пользователь может не только программировать, но и опять вытирать

пучком ультрафиолетового света, во флэш-памяти EPROM также применяется запоминающий элемент на МОП-транзисторе с дополнительным "плавающим затвором" (Floating Gate). При программировании он заряжается отрицательными зарядами и, благодаря этому, смещает пороговое напряжение МОП-транзистора (рис. 3.10).

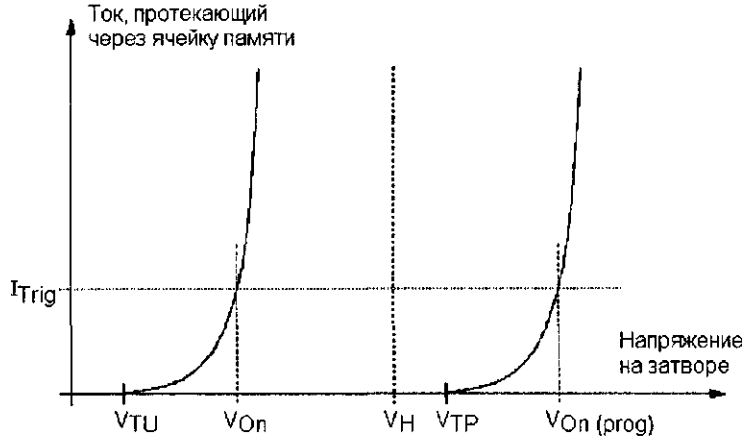


Рис. 3.10. Программирование ячейки запоминающего устройства с флэш-памятью EPROM

В противоположность памяти EPROM, у которой заряд может быть снова вытерт посредством облучения пучком ультрафиолетовых лучей через кварцевое стекло в корпусе в течение около 20 минут, в Flash-EPROM возможно, в зависимости от организации запоминающего устройства, электрическим методом "молниеносно" стереть весь кристалл или его отдельные блоки в течение нескольких миллисекунд (отсюда и название "flash" — "молния").

Процесс программирования

В случае ячейки флэш-памяти EPROM речь идет о МОП-транзисторе, у которого между затвором и каналом установлен еще один дополнительный "плавающий затвор" (Floating Gate) — рис. 3.11.

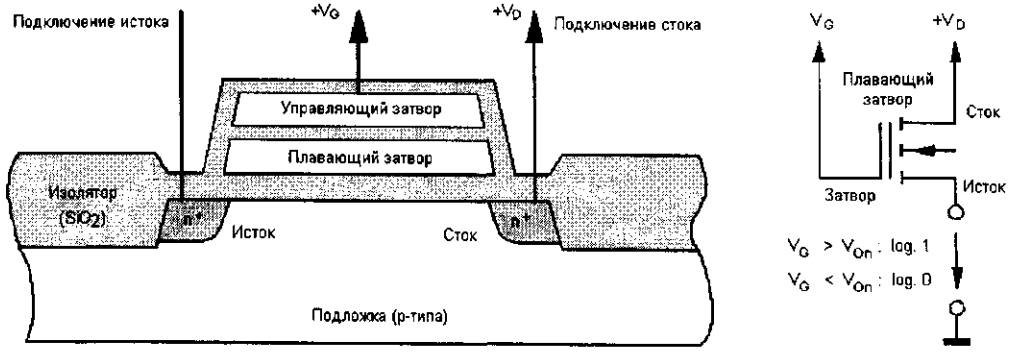


Рис. 3.11. Построение ячейки запоминающего устройства Flash-EPROM (МОП-транзистор с плавающим затвором): справа — работа ячейки памяти с незаряженным плавающим затвором

В вытертом состоянии этот плавающий затвор не содержит никакого электрического заряда. Как управляющий затвор, так и плавающий затвор помещены в

очень сильно изолированные слои из окиси кремния, поэтому они отделены друг от друга и от канала.

После подачи на управляющий затвор положительного напряжения, которое больше напряжения V_{TU} (см. рис. 3.10), возникает электрический ток через ячейку запоминающего устройства. Если этот ток превышает пороговое значение I_{Trig}, то содержимое ячейки запоминающего устройства интерпретируется как лог. 1. Напряжение V_{on} при этом является, по меньшей мере, тем напряжением, которое необходимо для МОП-транзистора в условиях нормальной эксплуатации, то есть, при незаряженном плавающем затворе можно переключиться в лог. 1.

Итак, незапрограммированная или вытертая ячейка флэш-памяти EPROM при подводе к затвору считывающего напряжения, превышающего напряжение V_{on}, всегда считывается как лог. 1. Для того чтобы ячейку памяти EPROM запрограммировать в лог. 0, необходимо предотвратить возможность ее отключения напряжением на управляющем затворе. Для этого на плавающий затвор подается отрицательное напряжение, которое смещает потенциал, необходимый для включения МОП-транзистора, до значения, превышающего V_{TP} (см. рис. 3.10). Теперь для включения МОП-транзистора потребуется считывающее напряжение больше, чем $V_{on\ (prog)}$. Поскольку при нормальной эксплуатации на управляющий затвор напряжение больше V_H быть подано не может, то в ячейку памяти на продолжительное время записано значение лог. 0.

Для программирования была разработана технология инжекции горячих электронов (hot electron injection). Если исток подключить на землю (как это показано на рис. 3.12), а на сток и управляющий затвор подать повышенное напряжение программирования V_{pp} примерно 12 В, то через канал МОП-транзистора будет протекать относительно сильный электрический ток. Некоторые электроны при этом становятся настолько "горячими" (то есть, насыщенными энергией), что могут преодолеть барьер, образованный слоем окиси кремния, и попасть на плавающий затвор.

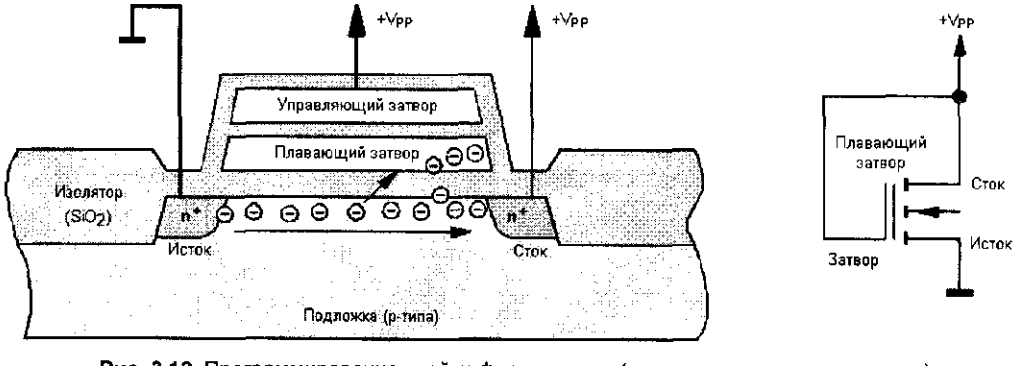


Рис. 3.12. Программирование ячейки флэш-памяти (инжекция горячих электронов)

Благодаря окружающему изоляционному слою, этот отрицательный заряд будет сохраняться на плавающем затворе также и по окончанию процесса программирования. Благодаря этому, включение транзистора запоминающего устройства через управляющий затвор будет заблокировано на длительное время.

Процесс стирания

Стирание данных на обычных запоминающих устройствах типа EPROM выполняют посредством облучения пучком ультрафиолетового света, вследствие чего заряд на плавающем затворе исчезает. Для этого в корпусе над кристаллом имеется кварцевое стекло.

Фотоны в пучке ультрафиолетового света поглощаются электронами в плавающем затворе, благодаря чему они становятся настолько насыщенными энергией, что преодолевают барьер окружающего их изоляционного слоя и могут взаимодействовать с электронами в управляющем затворе или в подложке. В отличие от этого, для стирания флэш-памяти EPROM применяется метод туннелирования Файлера-Нордхайма, который заключается в следующем: поскольку во флэш-памяти, в отличие от EPROM, изоляционный слой между истоком и плавающим затвором тоньше, можно добиться проникновения через него электронов в результате создания электрического поля.

Для этого ячейка запоминающего устройства подключается в соответствии с рис. 3.13 (на исток подается напряжение V_{pp}, управляющий затвор — на землю, контакт истока открыт). После этого плавающий затвор разряжается, и МОП-транзистор опять может быть приведен в состояние проводимости через управляющий затвор.

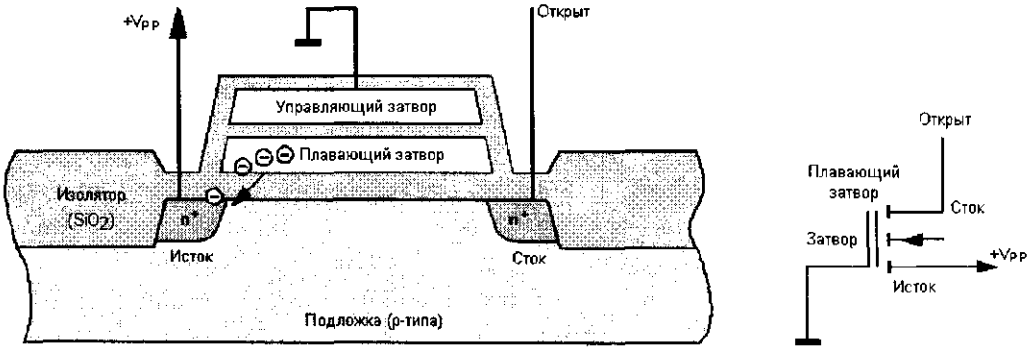


Рис. 3.13. Стирание информации в ячейке памяти типа Flash-EPROM (использование туннелирования)

Для полноты изложения следует также отметить, что процесс флэш-программирования основан на архитектуре "НЕ-ИЛИ". Это означает такое расположение полей запоминающего устройства, при котором чтение каждого транзистора может происходить отдельно. Наряду с этим, были разработаны другие виды архитектуры запоминающего устройства типа Flash-EPROM, например, с размещением ячеек типа "НЕ-И". Здесь как программирование, так и стирание выполняется с применением туннелирования, позволяющего сохранять полупроводниковый материал.

Сравнение технологических свойств, методов программирования и стирания при различных вариантах памяти EPROM представлено в табл. 3.5.

Предписания и инструкции по программированию для флэш-памяти микроконтроллеров AVR подробно представлены в главе 11, "Программирование памяти".

Таблица 3.5. Сравнение технологических свойств, методов программирования и стирания при различных вариантах памяти EPROM

	EPROM	*Flash-EPROM (технология "НЕ-ИЛИ")*	*EEPROM*
Величина ячейки относительно ячейки Flash-EPROM	0,8 ... 1	1	2,5... 3
МОП-транзисторов на одну ячейку памяти	1	1	минимум 2
Программирование	Спецприбор	В схеме	В схеме
Механизм программирования	Инжекция горячих электронов	Инжекция горячих электронов	Туннелирование
Разрешение	байт	байт	байт
Время программирования одного байта	Менее 100 мс	0,1...2 мс	1...10 мс
Стирание	Прибор с ультрафиолетовым светом	В схеме	В схеме
Механизм стирания	Ультрафиолетовый свет	Туннелирование	Туннелирование
Разрешение	Кристалл	Кристалл или блок	Байт
Время стирания	Около 20 минут	10...1000 мс	1...10 мс

Память для энергонезависимых данных (технология EEPROM)

EEPROM (E^2PROM)	AT90S1200	AT90S2313	AT90S4414	AT90S8515
	64 байта	128 байт	256 байт	512 байт

Память данных типа EEPROM

Микроконтроллеры семейства AVR имеют память типа EEPROM объемом от 64 до 512 байт, которая организована в виде отдельного запоминающее устройства. Запись и чтение могут выполняться по одному байту. Память AVR-EEPROM имеет срок службы минимум 100000 циклов записи/чтения.

Наряду со встроенной в кристалл флэш-памятью команд, все представители базовой серии микроконтроллеров AVR также имеют встроенную память типа EEPROM (различного объема) для запоминания энергонезависимых данных и констант. Это очень удобно, к примеру, при калибровке измерительных приборов, работающих под управлением микроконтроллеров AVR, у которых в памяти EEPROM в процессе настройки сохраняются параметры корректировки. Благодаря этому, в большинстве случаев полностью отпадает необходимость в настроечных потенциометрах и триммерах.

Технология памяти EEPROM

Под памятью типа EEPROM (**E**lectrically **E**rasable **PROM** — электрически стираемая память PROM) понимают такую память PROM, которую, как и Flash-EPROM, можно электрически программировать и снова стирать. При этом в слу-

чае памяти EEPROM за один раз стирается не вся информация на кристалле, благодаря чему возможно стирать только отдельные байты (то есть, во все ячейки строки, соответствующей байту, записывается лог. 0). Как программирование, так и стирание памяти EEPROM осуществляется с использованием туннелирования (см. раздел "Память команд (технология Flash-EPROM)").

В микроконтроллерах AVR преобразователи напряжения, используемые для выработки напряжения программирования и питания таймера, определяющего длительность программирования, интегрированы в кристалл. Кроме того, реализовано распознавание провалов напряжения для предотвращения записи при понижении рабочего напряжения ниже минимально допустимого значения. Поэтому для программирования одного байта достаточно всего лишь записать в соответствующий регистр адрес и байт данных, а затем активизировать процесс записи через регистр управления. Дальнейший процесс происходит на кристалле в автономном режиме, поэтому пользователю не о чем беспокоиться. Сначала будет стерт старый байт, а затем выполнено программирование нового байта. Этот процесс контролируется внутренне и его продолжительность, в зависимости от подведенного рабочего напряжения, составляет 2,5...4 мс.

Тем не менее, несмотря на такие простые и относительно быстрые процессы стирания и записи, память типа EEPROM нельзя использовать в качестве ОЗУ (RAM), поскольку количество возможных циклов записи ограничено: один байт не может быть записан более 10^6 раз. Продолжительность программирования в среднем составляет 3 мс, следовательно ресурс ячейки памяти, если ее программировать беспрерывно, можно исчерпать в течение всего лишь 50 минут.

По этой причине некоторые заводы-изготовители комбинируют память типа EEPROM с RAM. При таком варианте содержимое памяти переносится при прекращении подачи рабочего напряжения в память EEPROM. Благодаря этому, при нормальных условиях эксплуатации достигают короткого цикла записи, не приводящего к износу.

Доступ ЦП к памяти EEPROM на запись/чтение

Для программирования памяти EEPROM микроконтроллеров AVR нет необходимости применять внешнее программирующее устройство. Каждая ячейка памяти EEPROM может быть запрограммирована непосредственно во время выполнения пользовательской программы.

Если во время обращения к памяти EEPROM с целью записи происходит системный сброс, то результат будет неопределенный, поскольку регистр адреса EEAR будет обнулен. По этой причине могут содержаться ошибочные данные по адресу, подлежащему программированию, или в байте с адресом 0 памяти EEPROM!

Для программирования используются три регистра памяти EEPROM: регистр адреса EEAR, регистр данных EEDR и регистр управления EECR. Во всех трех случаях речь идет о 8-разрядном регистре, за исключением регистра EEAR микроконтроллера AT90S8515. Поскольку для этого типа в распоряжении имеется 512 байт памяти EEPROM, и, таким образом, для адресации необходимо 9 разрядов, то здесь регистр EEAR 16-разрядный, разделенный на две части: EEARH (старший байт) и EEARL (младший байт).

Рассмотрим вначале регистры EEAR, EEDR и EECR, используемые при обращении к памяти EEPROM.

Регистр адреса EEAR памяти EEPROM

Регистр адреса EEAR памяти EEPROM имеет длину в один байт (микроконтроллеры AT90S1200, AT90S2313, AT90S4414) или в два байта (микроконтроллер AT90S8515). Он расположен в области ввода/вывода по адресу $1E (RAM:$3E) а, в случае микроконтроллера AT90S8515 занимает байты с адресами $1E (RAM: $3E — младший байт) и $1F (RAM: $3F — старший байт). Эти байты доступны для чтения и записи. После подачи сигнала сброса они инициализируются нулями.

Разряд	15	14	13	12	11	10	9	8	
$1F ($3F)	–	–	–	–	–	–	–	EEAR8	EEARH [3]
$1E ($3E)	EEAR7 [2]	EEAR6 [1]	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEAR(L)
Разряд	7	6	5	4	3	2	1	0	

1) В микроконтроллере AT90S1200 отсутствует
2) В микроконтроллерах AT90S1200, AT90S2313 отсутствует
3) В микроконтроллерах AT90S1200, AT90S2313, AT90S4414 отсутствует

Для программирования или чтения байта данных памяти EEPROM в регистр адреса EEAR должен быть записан соответствующий адрес.

Регистр данных EEDR памяти EEPROM

Регистр данных EEDR памяти EEPROM находится в области ввода/вывода по адресу $1D (RAM: $3D). После подачи сигнала сброса он инициализируется нулями. Байт EEDR доступен для чтения и записи.

Разряд	7	6	5	4	3	2	1	0	
$1D ($3D)	MSB							LSB	EEDR

В процессе записи в память EEPROM подлежащий программированию байт загружается в регистр EEDR. В процессе чтения из памяти EEPROM в регистр EEDR записывается содержимое соответствующей строки EEPROM. Адрес в памяти EEPROM в обоих случаях определяется по содержимому регистра EEAR.

Регистр управления EECR памяти EEPROM

Регистр управления EECR памяти EEPROM находится в области ввода/вывода по адресу $1C (RAM: $3C). После подачи сигнала сброса он инициализируется нулями. В микроконтроллерах базовой серии семейства AVR используются только разряды 0...1 (AT90S1200) или же 0...2 регистра EECR (доступны для чтения и записи). Остальные разряды компания Atmel зарезервировала, и они доступны только для чтения (всегда содержат лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$1C ($3C)	–	–	–	–	–	EEMWE *)	EEWE	EERE	EECR

*) В AT90S1200 отсутствует

Чтение памяти EEPROM (для всех типов микроконтроллеров базовой серии семейства AVR)

Для управления процессом чтения используется разряд **EERE** (**EE**PROM **R**ead **E**nable — память EEPROM готова к чтению). После записи корректного адреса в регистр EEAR процесс чтения может быть активизирован установкой разряда EERE в регистре управления.

В микроконтроллерах AT90S1200, особенно в вариантах для высоких тактовых частот (окончание "–12" в обозначении модуля) разряд EERE необходимо устанавливать подряд два раза, чтобы предоставить аппаратной части достаточно времени для чтения ячеек памяти EEPROM. Это показано на примере рассмотренной ниже программы.

По окончанию считывания разряда EERE аппаратное обеспечение считывает требуемый байт в регистр EEDR, после чего уже нет необходимости вновь опрашивать разряд EERE, поскольку считывание длится только один цикл такта системной синхронизации.

После того как разряд EERE установлен в лог. 1, центральный процессор будет задержан на два такта системной синхронизации прежде, чем будет выполнена следующая команда (это не отображается при тестировании в AVR-Studio вплоть до версии 1.50).

Перед началом операции чтения программа пользователя должна постоянно опрашивать разряд EEWE и ждать появления лог. 0. Если во время программирования памяти EEPROM в соответствующий регистр ввода/вывода памяти EEPROM будут записаны новые адреса или данные, то еще продолжающийся процесс программирования будет прерван и результат будет неопределенным!

Запись в память EEPROM в микроконтроллерах AT90S2313, AT90S4414 и AT90S8515

Разряд **EEWE** (**EE**PROM **W**rite **E**nable — память EEPROM готова к записи) — это разряд управления процессом записи. Для записи байта в память EEPROM разряд EEWE необходимо установить в лог. 1, если в регистре EEAR находятся корректный адрес памяти EEPROM, а в регистре EEDR — байт данных, подлежащий программированию. Для предотвращения непреднамеренной записи в память EEPROM разряд EEWE может быть установлен только в том случае, если установлен также и разряд EEMWE.

Для программирования памяти EEPROM должны быть выполнены следующие действия.

1. Дождаться окончания процесса программирования памяти EEPROM (если он активен), то есть, пока разряд EEWE возвратится в состояние лог. 0.
2. Записать новый адрес в регистр EEAR памяти EEPROM (1 байт или 2 байта в случае микроконтроллера AT90S8515).
3. Записать требуемый байт данных в регистр EEDR памяти EEPROM.
4. Установить разряд EEMWE в лог. 1.
5. На протяжении следующих четырех периодов системного такта после установки разряда EEMWE в разряд EEWE должна быть записана лог. 1. Тем самым будет запущен процесс программирования.

По окончанию цикла программирования (типичная продолжительность — 2,5 мс при рабочем напряжении V_{CC} = 5 В и, соответственно, 4 мс при напряжении V_{CC} = 2,7 В), разряд EEWE аппаратно автоматически сбрасывается в лог. 0. Программа пользователя должна непрерывно опрашивать этот разряд, ожидая появления лог. 0, прежде чем приступить к программированию следующего байта.

После установки разряда EEWE в лог. 1 центральный процессор будет задержан на два такта системной синхронизации, после чего он может выполнять следующую команду (это не будет показано в результатах тестирования в AVR-Studio вплоть до версии 1.50; также не полностью учтена и продолжительность процесса программирования).

При записи одного байта в память EEPROM должен быть также установлен в лог. 1 разряд **EEMWE** (**EE**PROM **M**aster **W**rite **E**nable — общее разрешение записи в память EEPROM). После того как разряд EEMWE установлен, уровень лог. 1 в нем сохраняется в течение четырех периодов такта системной синхронизации, а затем будет выполнен аппаратный сброс в лог. 0. Программа пользователя может программировать байт посредством записи лог. 1 в разряд EEWE только на протяжении этих четырех тактов системной синхронизации. Если будет установлен разряд EEWE, но без установки также и разряда EEMWE, то процесс программирования начат не будет.

Рассмотрим в качестве примера короткую программу, которая показывает процесс программирования и чтения памяти EEPROM микроконтроллера AT90S8515.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\EEPR8515.asm`

```
.include "8515def.inc"
.equ AdrWr = $100     ; Адрес, по которому будет выполнено программирование
.equ AdrRd = $101     ; Адрес, по которому должно быть выполнено чтение
.def EEdrd = r0       ; Байт, считанный из памяти EEPROM
.def EEdwr = r16      ; Байт, подлежащий программированию в память EEPROM
.def Temp = r17       ; Вспомогательный регистр

RESET:
000000 c013    rjmp Initial           ; Переход к части инициализации

EEWrite:                              ; ***** Подпрограмма "Запись в EEPROM"
000001 99e1    sbic EECR,EEWE         ; Если EEWE не лог. 0, то
000002 cffe    rjmp EEWrite           ;   ожидать дальше
000003 e011    ldi  Temp,High(AdrWr)  ; Старший байт адреса записи в EEPROM -
000004 bb1f    out  EEARH,Temp        ;   в регистр адреса
000005 e010    ldi  Temp,Low(AdrWr)   ; Младший байт адреса записи в EEPROM -
000006 bb1e    out  EEARL,Temp        ;   в регистр адреса
000007 bb0d    out  EEDR,EEdwr        ; Байт данных - в регистр данных
000008 9ae2    sbi  EECR,EEMWE        ; Разряд EEMWE разрешает программирование
000009 9ae1    sbi  EECR,EEWE     ; Разряд EEWE установлен: начало программир-я
                                      ; Команда выполняется в течение 4-х
                                      ; тактов, поскольку задержка
                                      ; ЦП составляет 2 такта
00000a 9508 ret

EERead:                               ; ***** Подпрограмма "Чтение EEPROM"
00000b 99e1    sbic EECR, EEWE        ; Если EEWE не лог. 0, то
00000c cffe    rjmp EERead            ;   ждать дальше
```

```
00000d e011     ldi   Temp,High(AdrRd) ; Старший байт адреса чтения EEPROM -
00000e bb1f     out   EEARH, Temp       ;   в регистр адреса
00000f e011     ldi   Temp,Low(AdrRd)  ; Младший байт адреса чтения EEPROM -
000010 bb1e     out   EEARL, Temp       ;   в регистр адреса
000011 9ae0     sbi   EECR, EERE        ; Установить разряд EERE - начать
                                        ; процесс чтения. Команда выполняется 4
                                        ; такта, поскольку ЦП задержан
                                        ; на 2 такта
000012 b20d     in    EEdrd, EEDR       ; Чтение байта данных
000013 9508 ret

Initial:
000014 e51f     ldi   Temp, Low(RAMEND)
000015 bf1d     out   SPL, Temp
000016 e012     ldi   Temp, High(RAMEND)
000017 bf1e     out   SPH, Temp         ; Установить указатель стека
000018 ef1f     ldi   Temp, $ff         ; Направление передачи данных - вывод
000019 bb1a     out   DDRA, Temp        ; В регистр направления передачи данных
00001a e010     ldi   Temp, $00         ; Направление передачи данных - ввод
00001b bb14     out   DDRC, Temp        ; В регистр направления передачи данных
00001c bb15     out   PortC, Temp       ; Выбрать вход с тремя состояниями
00001d b303     in    EEdwr, PinC       ; Загрузить байт данных из порта C
00001e dfe2     rcall EEWrite           ; Программируем байт по адресу $100
00001f dfeb     rcall EERead            ; Чтение байта данных по адресу $101
000020 ba0b     out   PortA, EEdrd      ; Считанный байт передать в порт A
Endlos:
000021 cfff     rjmp Endlos             ; Символически для продолжения программы
```

Подпрограмма `EEWrite` с помощью циклического опроса разряда EEWE ожидает готовности памяти EEPROM к новому программированию. Как только разряд EEWE переходит в состояние лог. 0, в регистры EEARH, EEARL и EEDR будут записаны соответственно два байта адреса EEPROM и байт данных, подлежащий программированию. Посредством установки разряда EEMWE будет разрешено программирование. Теперь у программы пользователя есть время продолжительностью 4 периода такта системной синхронизации для запуска процесса программирования посредством установки разряда EEWE. В рассмотренном выше примере это происходит уже при выполнении следующей команды. Если выход из подпрограммы осуществляется по команде `ret`, то процесс программирования продолжается дальше.

Подпрограмма `EERead` с помощью циклического опроса разряда EEWE ожидает окончания процесса программирования (если он в данный момент активен). Как только разряд EEWE перейдет в состояние лог. 0, в регистры EEARH и EEARL будут записаны два байта адреса памяти EEPROM, подлежащие чтению. После этого для начала процесса чтения будет установлен разряд EERE. В связи с тем, что такт системной синхронизации в микроконтроллере AT90S8515 должен составлять максимум 8 МГц, здесь нет необходимости устанавливать разряд EERE два раза, как это происходит в микроконтроллере AT90S1200 для того, чтобы предоставить в распоряжение аппаратной части достаточно времени для чтения ячейки памяти EEPROM. Требуемый байт находится в области ввода/вывода, соответствующей регистру EEDR, и переписывается для дальнейшей обработки в рабочий регистр EEdrd.

Основная программа вначале инициализирует указатель стека и используемые порты ввода/вывода A и C. Вслед за этим считывается байт, присутствующий на выводах порта C, и записывается с помощью подпрограммы `EEWrite` по адресу $100 памяти EEPROM. Затем подпрограмма `EERead` считывает содержимое ячейки памяти EEPROM по адресу $101 в рабочий регистр EEdrd. Этот байт в дальнейшем передается в порт A. Последующее выполнение программы показано в виде символического бесконечного цикла.

Запись в память EEPROM в микроконтроллере AT90S1200

Разряд **EEWE** (**EE**PROM **W**rite **E**nable) управляет процессом записи. Для того чтобы записать байт в память EEPROM, разряд EEWE должен быть программно установлен в лог. 1, если в регистре EEAR находится корректный адрес EEPROM, а в регистре EEDR — байт данных, подлежащий программированию. Разряд EEMWE, который в других микроконтроллерах базовой серии семейства AVR предотвращает непреднамеренную запись в память EEPROM, в микроконтроллере AT90S1200 отсутствует.

Для программирования памяти EEPROM в микроконтроллере AT90S1200 должны быть выполнены следующие действия:

1. Дождаться окончания процесса программирования памяти EEPROM (если он активен), то есть, пока разряд EEWE установится в лог. 0.
2. Записать в регистр адреса памяти EEPROM новый адрес (1 байт).
3. Записать требуемый байт данных в регистр данных EEPROM.
4. Установить разряд EEWE в лог. 1. Тем самым будет начат процесс программирования.

По окончанию цикла программирования (типичная продолжительность — 2,5 мс при рабочем напряжении V_{CC} = 5 В и, соответственно, 4 мс при V_{CC} = 2,7 В) аппаратная часть автоматически устанавливает разряд EEWE в лог. 0. Программа пользователя должна непрерывно опрашивать этот разряд и ожидать появления уровня лог. 0, прежде чем снова обратиться к памяти EEPROM.

После установки разряда EEWE в лог. 1 центральный процессор задерживается на два такта системной синхронизации, после чего может выполнять следующую команду (это не будет показано в результатах тестирования в AVR-Studio вплоть до версии 1.50; также не полностью учтена и продолжительность процесса программирования).

Рассмотрим в качестве примера короткую программу, которая показывает процесс программирования и чтения памяти EEPROM в микроконтроллере AT90S1200.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\EEPR1200.asm`

```
.include "1200def.inc"

.def EEard = r18    ; Адрес, по которому должно быть выполнено чтение
.def EEawr = r17    ; Адрес, по которому должно быть выполнено программирование
.def EEdwr = r16    ; Байт, подлежащий программированию в памяти EEPROM
.def EEdrd = r0     ; Байт, считанный из памяти EEPROM
```

```
000000 c00d     rjmp RESET          ; К основной программе после сброса

EEWrite:                            ; ***** Подпрограмма "Запись в EEPROM"
000001 99e1     sbic EECR, EEWE     ; Если EEWE - не лог. 0, то
000002 cffe     rjmp EEWrite        ;   ожидать дальше
000003 bb1e     out  EEAR, EEawr    ; Запись адреса в регистр адреса
000004 bb0d     out  EEDR, EEdwr    ; Запись байта данных в регистр данных
000005 9ae1     sbi  EECR, EEWE     ; Установка разряда EEWE - начало процесса
                                    ; программирования. Команда выполняется за
                                    ; 4 такта, поскольку ЦП
                                    ; задерживается на 2 такта
000006 9508 ret

EERead:                             ; ***** Подпрограмма "Чтение EEPROM"
000007 99e1     sbic EECR, EEWE     ; Если EEWE - не лог. 0, то
000008 cffe     rjmp EERead         ;   ожидать дальше
000009 bb2e     out  EEAR, EEard    ; Запись адреса в регистр адреса
00000a 9ae0     sbi  EECR, EERE     ; Установка разряда EERE - начало процесса
                                    ; чтения. Команда выполняется за 4 такта,
                                    ; поскольку ЦП задерживает на 2 такта
00000b 9ae0     sbi  EECR, EERE     ; Повторная установка разряда EERE. Команда
                                    ; выполняется за 4 такта, поскольку
                                    ; ЦП задерживается на 2 такта
00000c b20d     in   EEdrd, EEDR    ; Читать байт данных
00000d 9508 ret

RESET:                              ; ***** Основная программа
00000e ea0a     ldi  EEdwr, $aa     ; Установка регистра данных памяти EEPROM
00000f e110     ldi  EEawr, $10     ; Установка регистра адреса памяти EEPROM
000010 dff0     rcall EEWrite       ; Записываем $aa по адресу $10 EEPROM
000011 e101     ldi  EEard, $11     ; Установка регистра адреса памяти EEPROM
000012 dff4     rcall EERead        ; Считываем байт данных по адресу $11
Endlos:
000013 cfff     rjmp Endlos         ; Символически для продолжения программы
```

Подпрограмма `EEWrite` с помощью циклического опроса разряда EEWE ожидает готовности памяти EEPROM к новому программированию. Как только разряд EEWE переходит в состояние лог. 0, в регистры EEAR и EEDR будут соответственно записаны адрес EEPROM и байт данных, подлежащий программированию. После этого будет установлен разряд EEWE для начала процесса программирования.

Подпрограмма `EERead` с помощью циклического опроса разряда EEWE ожидает окончания процесса программирования (если он активен). Как только разряд EEWE переходит в состояние лог. 0, в регистр EEAR будет записан подлежащий чтению адрес памяти EEPROM. После этого для начала процесса чтения дважды последовательно будет установлен разряд EERE. Если микроконтроллер работает с тактом системной синхронизации 12 МГц, то особенно важно, чтобы этот разряд был установлен два раза, чтобы предоставить в распоряжение аппаратной части достаточно времени для чтения ячейки памяти EEPROM. После этого требуемый байт будет записан в область ввода/вывода, соответствующую регистру EEDR, и переписан для дальнейшей обработки в рабочий регистр EEdrd.

Основная программа вначале вызывает подпрограмму `EEWrite` для программирования байта $AA по адресу $10 памяти EEPROM. Затем подпрограмма `EERead` выполняет чтение содержимого ячейки памяти EEPROM по адресу $11 в рабочий регистр EEdrd. Дальнейшее выполнение программы показано в виде символического бесконечного цикла.

Различные способы адресации команд и данных

В микроконтроллерах семейства AVR для доступа к памяти команд (флэш-память) и данных (регистры, порты ввода/вывода и память SRAM) используются очень удобные и эффективные режимы адресации.

В настоящем разделе описаны различные виды адресации, которые поддерживаются архитектурой микроконтроллеров семейства AVR. Далее по тексту сокращение "OP" означает "Operations-Code" ("код операции"), то есть, неизменную часть командного слова, на основании которой дешифратор распознает команды.

Некоторые из описанных ниже режимов адресации используются не во всех микроконтроллерах семейства AVR!

Прямая адресация одного регистра

Такой режим адресации (рис. 3.14) используется во всех микроконтроллерах базовой серии семейства AVR.

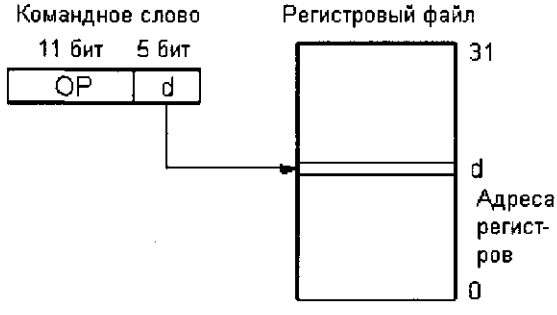


Рис. 3.14. Прямая адресация одного регистра:

Регистр с адресом d содержит операнды, к которым обращается команда. Примеры к рис. 3.14:

16-разрядное командное слово

com Rd	1001	010d	dddd	0000
inc Rd	1001	010d	dddd	0011

```
Код   Команда     Комментарий
9500   com r16    ; Дополнение до единицы содержимого r16
94f3   inc r15    ; Увеличить на 1 содержимое r15
```

Прямая адресация двух регистров Rd и Rr

Такой режим адресации (рис. 3.15) используется во всех микроконтроллерах базовой серии семейства AVR. Регистры Rd и Rr с адресами d и r содержат операнды, к которым обращается команда. Результат операции записывается в регистр Rd.

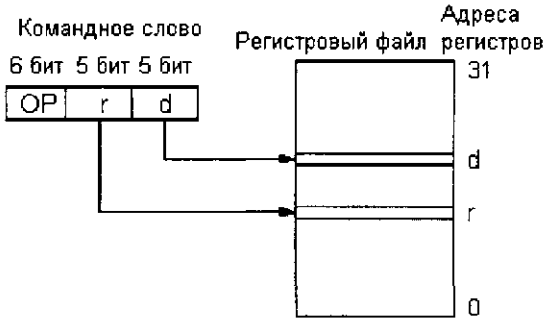


Рис. 3.15. Прямая адресация двух регистров:

Примеры к рис. 3.15:

16-разрядное командное слово

`add Rd, Rr`	0001	11rd	dddd	rrrr
`cp Rd, Rr`	0001	01rd	dddd	rrrr

```
Код    Команда          Комментарий
0ef0   add r15, r16     ; Сумма <r15> + <r16> -
                        ; сохраняется в регистре r15
1548   cp  r20, r8      ; Сравнение <r20> и <r8>
```

Прямая адресация области ввода/вывода

Такой режим адресации (рис. 3.16) используется во всех микроконтроллерах базовой серии семейства AVR.

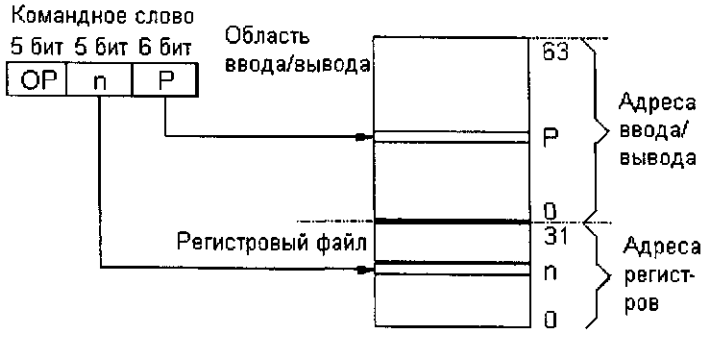


Рис. 3.16. Прямая адресация портов в области ввода/вывода

Буквой "P" обозначен адрес выбранного порта в области ввода/вывода, а буквой "n" — адрес регистра в регистровом файле. Примеры к рис. 3.16:

16-разрядное командное слово

in Rd, P	1011	0PPd	dddd	PPPP
out P, Rr	1011	1PPr	rrrr	PPPP

Код	Команда	Комментарий
b6fe	in r15, 0x3E	; Записать <SPH> в r15
be9d	out 0x3D, r9	; Установить SPL в <r9>

Прямая адресация памяти данных (SRAM)

Такой режим адресации (рис. 3.17) отсутствует в микроконтроллере AT90S1200 базовой серии семейства AVR.

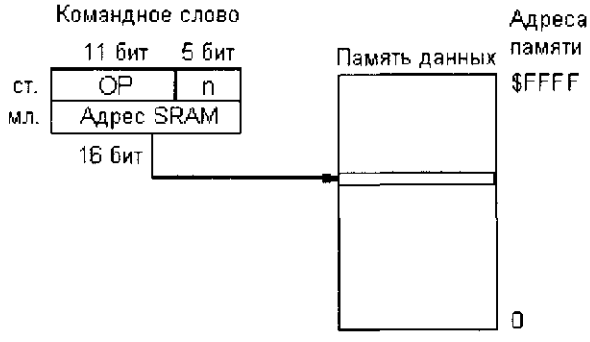


Рис. 3.17. Прямая адресация памяти данных

Команды для прямой адресации памяти данных состоит из двух командных слов, и для их выполнения требуется три тактовых цикла. Старшее командное слово содержит код операции и адрес *n* соответствующего регистра в регистровом файле, — источника и, соответственно, приемника для передачи данных. Младшее командное слово содержит адрес области памяти, к которой необходимо обратиться.

В связи с тем, что адресация может быть выполнена во всей памяти данных (то есть, как внутренней памяти SRAM, так и внешней памяти, которая может использоваться в микроконтроллерах AT90S4414 и AT90S8515), то с помощью этого вида адресации можно также передавать данные от регистра к регистру или от портов ввода/вывода к регистрам, поскольку как область регистров, так и область ввода/вывода отражены в памяти данных SRAM (см. рис. 3.2).

Тем не менее, для этой цели применяют такие специальные команды как `mov Rd,Rr` и, соответственно, `out P,Rr`, которые состоят только из одного слова и обрабатываются за один цикл, вследствие чего прямая адресация памяти SRAM в данном случае невыгодна. Примеры к рис. 3.17:

16-разрядное командное слово

lds Rd,k	1001	000d	dddd	0000	MSW
	kkkk	kkkk	kkkk	kkkk	LSW
sts k,Rr	1001	001d	dddd	0000	MSW
	kkkk	kkkk	kkkk	kkkk	LSW

```
Код          Команда          Комментарий
9100 cccc    lds r16, 0xcccc  ; В r16 загружается
                              ; содержимое ячейки внешней
                              ; памяти с адресом $CCCC
92f0 0060    sts 0x60, r15    ; <r15> сохраняется в 1-й
                              ; ячейке внутренней SRAM
9100 005d    lds r16, 0x5d    ; В r16 загружается
                              ; содержимое SPL
92f0 005e    sts 0x5e, r15    ; <r15> сохраняется в SPH
9100 001f    lds r16, 0x1F    ; В r16 загружается
                              ; содержимое регистра r31
92f0 0000    sts 0x00, r15    ; <r15> сохраняется в r0
```

Как уже было отмечено выше, четыре последние команды допустимы, но их лучше не использовать.

Косвенная адресация памяти данных (SRAM)

Такой режим адресации (рис. 3.18) отсутствует в микроконтроллере AT90S1200 базовой серии семейства AVR.

Если вспомнить такое "узкое место" семейства микроконтроллеров 8051 компании Intel как указатель данных, тогда сразу станет понятно, насколько облегчает работу адресация с тремя возможными указателями x, y и z, используемая в микроконтроллерах AVR.

При косвенной адресации происходит передача данных между регистром, указанным в командном слове с помощью 5-разрядного адреса n, и ячейкой памяти, адресация которой осуществляется с помощью регистра двойной длины x, y или z. При этом x состоит из двух регистров: r26 (младший байт) и r27 (старший байт); y — из r28 (младший байт) и r29 (старший байт); а z — из r30 (младший байт) и r31 (старший байт).

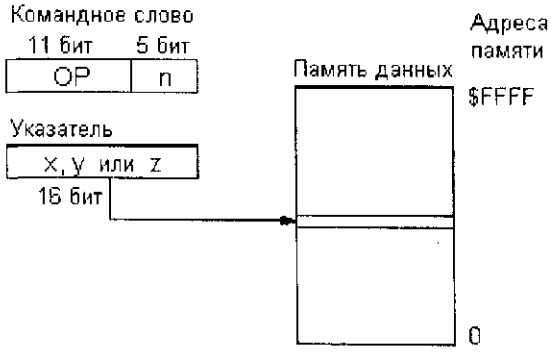


Рис. 3.18. Косвенная адресация памяти данных:

Команды для косвенной адресации памяти данных состоят только из одного командного слова, и для их выполнения требуется два тактовых цикла.

При косвенной адресации, так же как и при прямой, можно обращаться к области регистров и к области ввода/вывода. Примеры к рис. 3.18.

16-разрядное командное слово

st x,Rr	1001	001r	rrrr	1100
ld Rd,y	1000	000d	dddd	1000

```
Код     Команда      Комментарий
8108    ld r16, y    ; В r16 загружается содержимое ячейки,
                     ; адресованной с помощью <y>
92fc    st x, r15    ; <r15> сохраняется в ячейке памяти,
                     ; адресованной с помощью <x>
```

Косвенная адресация регистров в микроконтроллере AT90S1200

Такой режим адресации (рис. 3.19) используется только в микроконтроллере AT90S1200 базовой серии семейства AVR.

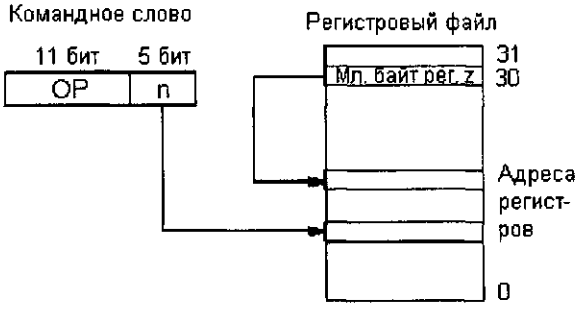


Рис. 3.19. Косвенная адресация регистров в микроконтроллере AT90S1200

При косвенной адресации регистра, используемой в микроконтроллерах модели AT90S1200, данные передаются между регистром, указанным в командном слове с помощью 5-разрядного адреса n, и регистром, адресованным с помощью младшего байта указателя z (адрес регистра 30 = $1E). Содержимое старшего байта указателя z (адрес регистра 31 = $1F) при этом не имеет никакого значения.

Команды для косвенной адресации регистра в микроконтроллерах AT90S1200 состоят из одного командного слова, и для их выполнения требуется два тактовых цикла. Примеры к рис. 3.19:

16-разрядное командное слово

st z,Rr	1000	001r	rrrr	0000
ld Rd,z	1000	000d	dddd	0000

```
Код     Команда      Комментарий
8100    ld r16, z    ; Копировать в r16 содержимое
                     ; регистра, адресованного через <z>
82f0    st z, r15    ; Сохранить <r15> в регистр,
                     ; адресованный через <z>
```

Например, перед выполнением команды `ld r16,z` младший байт (r30) указателя z должен содержать `$18`. Таким образом, указатель z ссылается на регистр r24. Пусть регистр r24 содержит `$55`, тогда после выполнения команды регистр r16 также будет содержать `$55`.

Косвенная адресация памяти данных с последующим приращением адреса

Такой режим адресации (рис. 3.20) отсутствует в микроконтроллере AT90S1200 базовой серии семейства AVR.

При такой адресации данные передаются между регистром, указанным в командном слове с помощью 5-разрядного адреса n, и ячейкой памяти, адресуемой с помощью содержимого регистра двойной длины x, y или z. При этом x состоит из двух регистров: r26 (младший байт) и r27 (старший байт); y — из r28 (младший байт) и r29 (старший байт), а z — из r30 (младший байт) и r31 (старший байт).

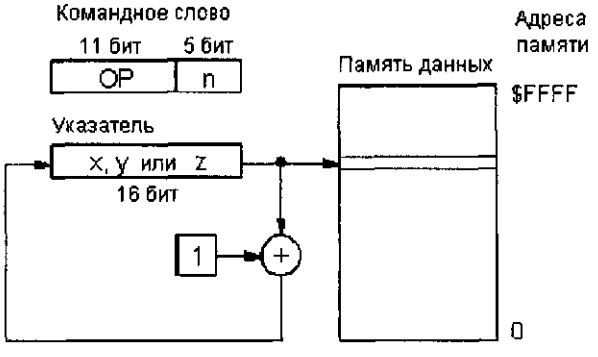


Рис. 3.20. Косвенная адресация памяти данных с последующим приращением адреса

После передачи данных содержимое указателя (то есть, регистра двойной длины x, y или z) автоматически увеличивается на 1 (приращение адреса). Эта команда очень выгодна, когда должны быть обработаны байты, следующие в памяти один за другим.

Такие команды состоят из одного командного слова, и для их выполнения требуются два тактовых цикла. Примеры к рис. 3.20:

16-разрядное командное слово

`st x+,Rr`	1001	001r	rrrr	1101
`ld Rd,z+`	1001	000d	dddd	0001

```
Код     Команда          Комментарий
9101    ld r16, z+       ; Записать байт данных в r16,
                         ; после чего увеличить <z> на 1
92fd    st x+, r15       ; Сохранить <r15>, после чего
                         ; увеличить <x> на 1
```

Косвенная адресация памяти данных с предварительным уменьшением адреса

Такой режим адресации (рис. 3.21) отсутствует в микроконтроллере AT90S1200 базовой серии семейства AVR.

При такой адресации данные передаются между регистром, указанным в командном слове с помощью 5-разрядного адреса n, и ячейкой памяти, адресуемой с помощью содержимого регистра двойной длины x, y или z. При этом x состоит из

двух регистров: r26 (младший байт) и r27 (старший байт); y — из r28 (младший байт) и r29 (старший байт), а z — из r30 (младший байт) и r31 (старший байт).

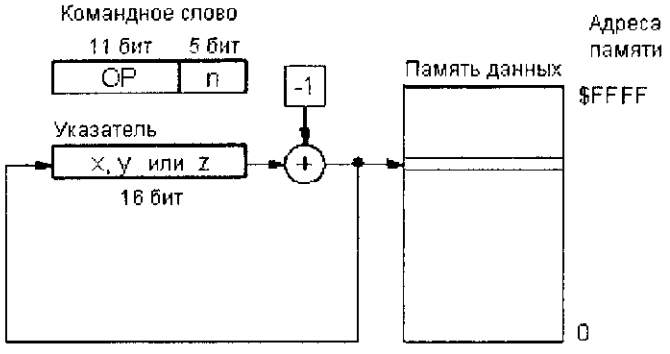


Рис. 3.21. Косвенная адресация памяти данных с предварительным уменьшением адреса

Перед передачей данных содержимое указателя (то есть, регистра двойной длины x, y или z), автоматически уменьшается на 1 (предварительное уменьшение адреса). Эта команда очень выгодна, когда должны быть обработаны байты, следующие в памяти один за другим.

Такие команды состоят из одного командного слова, и для их выполнения требуются два тактовых цикла.

Примеры к рис. 3.21:

16-разрядное командное слово

`st -y,Rr`	1001	001r	rrrr	1010
`ld Rd,-z`	1001	000d	dddd	0010

Код	Команда	Комментарий
9102	`ld r16, -z`	`; Уменьшить <z> на 1, после чего`
		`; скопировать в r16 байт данных`
92fa	`st -y, r15`	`; Декрементировать <y>, после чего`
		`; сохранить в памяти <r15>`

Относительная адресация памяти данных

Такой режим адресации (рис. 3.22) отсутствует в микроконтроллере AT90S1200 базовой серии семейства AVR.

При такой адресации данные передают между регистром, указанным в командном слове с помощью 5-разрядного адреса n, и ячейкой памяти, адресуемой с помощью суммы 6-разрядного адреса q, указанного в командном слове, и содержимого указателя (y или z). При этом указатель — это регистр двойной длины y (r28 — младший байт, r29 — старший байт) или z (r30 — младший байт, r31 — старший байт).

Перед передачей данных содержимое указателя (то есть, регистра двойной длины y или z) складывается с 6-разрядным адресом q (смещение относительно базового адреса). Содержимое указателя после выполнения этой операции остается неизменным.

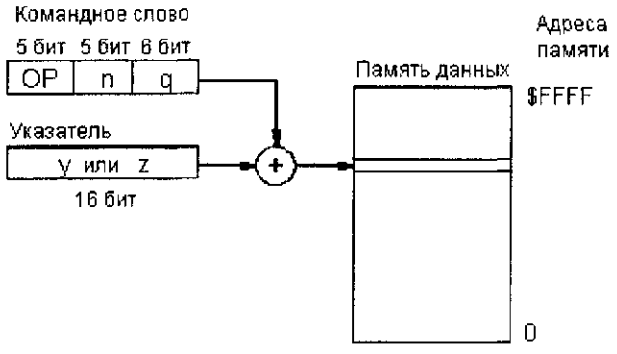


Рис. 3.22. Относительная адресация памяти данных

Такие команды выгодны, когда значения должны быть извлечены из таблицы. Они состоят из одного командного слова, а для их выполнения требуется два тактовых цикла. Примеры к рис. 3.22:

16-разрядное командное слово

`std y+q,Rr`	10q0	qq1r	rrrr	1qqq
`ldd Rd,z+q`	10q0	qq0d	dddd	0qqq

Код	Команда	Комментарий
ad07	`ldd r16, z+0x3F`	`; Копировать содержимое ячейки памяти`
		`; с адресом <z> + $003F в регистр r16`
82ff	`std y+7, r15`	`; Сохранить <r15> в ячейке`
		`; памяти по адресу <y> + $0007`

Адресация констант в памяти программ

Такой режим адресации (рис. 3.23) отсутствует в микроконтроллере AT90S1200 базовой серии семейства AVR.

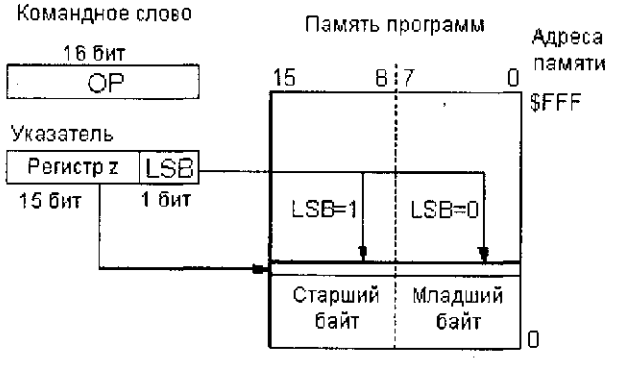


Рис. 3.23. Адресация констант в памяти программ

Микроконтроллеры семейства AVR позволяют использовать в выполняющейся программе предопределенные константы из памяти программ (например, определенные директивой ассемблера `.db`). С помощью команды `lpm` (**L**oad **P**rogram **M**emory — загрузка памяти программ) можно скопировать байт из памяти программ в регистр r0. С этой целью в разряды 1.. 15 регистра z, который опять вы-

ступает в роли указателя, должен быть загружен требуемый адрес памяти команд. В зависимости от построения памяти программ соответствующего микроконтроллера базовой серии семейства AVR, можно адресовать до 4 Кбайт слов (адреса 000...$FFF в микроконтроллере AT90S8515). Бит 0 (LSB) определяет, какой байт должен быть загружен в регистр r0: младший (бит 0 = 0) или старший (бит 0 = 1).

Команда `lpm` состоит из одного командного слова, и для ее выполнения требуется три тактовых цикла. Пример к рис. 3.23:

16-разрядное командное слово

`lpm`	1001	0101	110X	1000

Код	Команда	Комментарий
95c8	lpm	; Копировать в r0 содержимое байта,
		; адресованного с помощью указателя z

Например, перед выполнением команды `lpm` указатель z (r30 — младший байт, r31 — старший байт) содержит `$00F0`. По LSB (бит 0 = 0) выбирается младший байт, адрес $0078 памяти команд определяется с помощью содержимого разрядов 1...15 (смещение $00F0 вправо на один разряд). Слово по адресу $0078 памяти команд должно содержать `$55aa`. В этом примере после выполнения команды `lpm` регистр r0 будет содержать `$aa`.

Прямая адресация памяти программ (команды `jmp` и `call`)

В микроконтроллерах базовой серии семейства AVR эти команды не используются, однако их все равно следует рассмотреть, принимая во внимание разработку новых моделей (рис. 3.24).

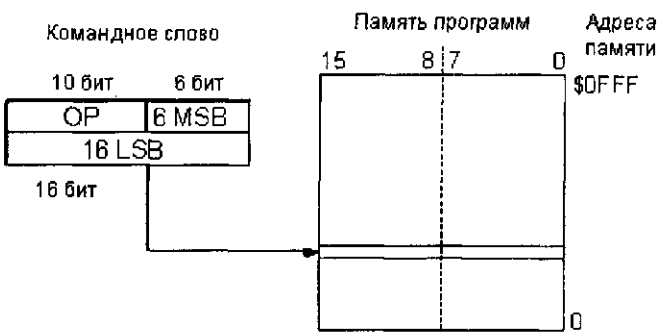


Рис. 3.24. Прямая адресация памяти программ:

Команды `jmp` и `call` выполняют прямой переход (Long Jmp) и, равным образом, вызов подпрограммы (Long Call) по указанному адресу, то есть, в счетчик команд загружается непосредственно указанный адрес, а выполнение программы продолжается с адреса, содержащегося в обоих командных словах. Теоретически, с помощью адреса шириной 22 разряда в памяти программ можно адресовать 4 Мслов. Естественно, на практике доступен только физический объем запоминающего устройства.

Команды `jmp` и `call` состоят из двух командных слов, и для их выполнения необходимы, соответственно, три (`jmp`) и четыре (`call`) тактовых цикла.

Примеры к рис. 3.24:

16-разрядное командное слово

jmp adr k	1001	010k	kkkk	110k	high
	kkkk	kkkk	kkkk	kkkk	low
call adr a	1001	010k	kkkk	111k	high
	kkkk	kkkk	kkkk	kkkk	low

```
Адрес   Код          Метка   Команда
.org 0
000000  940c 0300    label1: jmp label2
.org 0x300
000300  940c 0000    label2: jmp label1
```

Косвенная адресация памяти программ (команды `ijmp` и `icall`)

Такой режим адресации (рис. 3.25) отсутствует в микроконтроллере AT90S1200 базовой серии семейства AVR.

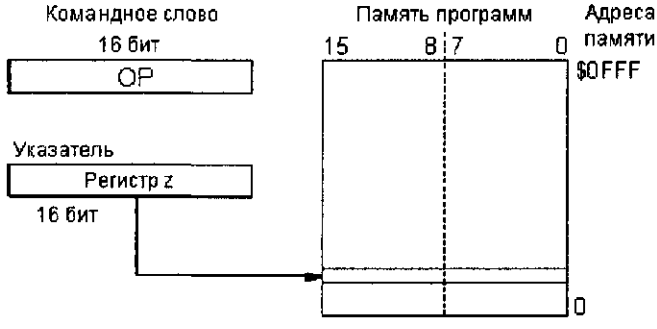


Рис. 3.25. Косвенная адресация памяти программ

Команды `ijmp` и `icall` выполняют косвенный переход и, равным образом, вызов подпрограммы по адресу, находящемуся в указателе z. В счетчик команд загружается содержимое регистра z, и выполнение программы продолжается с этого адреса. Теоретически, с помощью имеющегося в распоряжении 16-разрядного адреса в памяти программ можно адресовать 64 Кслов. На практике же, естественно, доступен только физический объем запоминающего устройства (например, в микроконтроллере AT90S8515 доступно 4 Кслов).

Косвенную адресацию очень удобно применять в случае использования конструкции CASE, разветвляющей программу для выбора нескольких вариантов.

Команды `ijmp` и `icall` состоят из одного командного слова, и для их выполнения необходимы, соответственно, два (`ijmp`) и три (`icall`) тактовых цикла.

Примеры к рис. 3.25:

16-разрядное командное слово

ijmp	1001	0100	XXXX	1001
icall	1001	0101	XXXX	1001

```
Адрес  Код        Метка    Команда
.org 0
000000 e0f3       label1: ldi r31,03    ; старший байт z
000001 e0ea               ldi r30,0x0a  ; младший байт z
000002 9409               ijmp          ; переход к <z>
.org 0x030a
00030a 940c 0000 label2: nop            ; Продолжение
```

Относительная адресация памяти программ (команды `rjmp` и `rcall`)

Этот тип адресации (рис. 3.26) поддерживается всеми микроконтроллерами базовой серии семейства AVR.

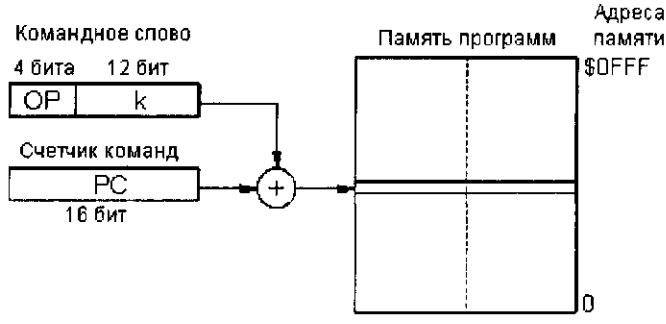


Рис. 3.26. Относительная адресация памяти программ:

При относительной адресации памяти программ константа k шириной 12 разрядов, которая содержится в командном слове, добавляется к текущему содержимому счетчика команд, после чего счетчик команд инкрементируется на 1 (PC = PC + k +1).

Во всех видах памяти команд, размер которых не превышает 4 Кслов или 8 Кбайт (то есть, память всех микроконтроллеров базовой серии семейства AVR), с помощью относительной адресации и применения так называемой круговой адресации (Wrap Around) можно адресовать всю область от $000 до $FFF. Это показано на рис. 3.27.

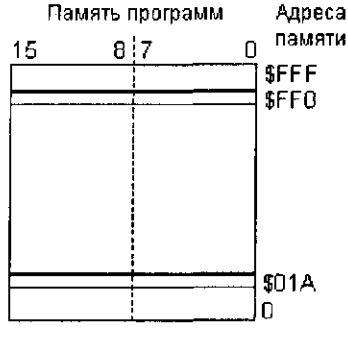


Рис. 3.27. Относительная адресация памяти программ

Если необходимо перейти с текущего адреса $01A на $FF0 (см. в качестве примера рис. 3.27), то к счетчику команд прибавляется 12-разрядная константа k = $FD5, и он содержит $FEF. После автоматического инкрементирования в счетчике команд находится требуемый адрес перехода $FF0.

Если, наоборот, необходимо перейти от текущего адреса $FF0 к адресу $01A, то к содержимому счетчика команд прибавляется 12-разрядная константа k = $029 (круговая адресация). В этом случае счетчик команд содержит $019, так как в микроконтроллере AT90S8515 его ширина составляет только 12 разрядов, а выход за границы, возникающий при суммировании, не учитывается. После автоматического инкрементирования в счетчике команд получится требуемый адрес перехода $01A.

Команды `rjmp` и `rcall` состоят из одного командного слова, и для их выполнения необходимы, соответственно, два (`rjmp`) и три (`rcall`) тактовых цикла.

При круговой адресации микроконтроллеры AVR с объемом памяти 4 Кслов (например, модель AT90S8515) становятся слишком загруженными, поэтому в таком случае она должна быть исключена из исходного кода на ассемблере явно. Для этого в меню **Options** среды разработки ассемблера необходимо установить соответствующий флажок. Если этого не сделать, то ассемблер при расстояниях перехода свыше 2047 ($7FF) командных слов выдаст сообщение об ошибке!

В микроконтроллерах AVR с меньшим объемом памяти команд в круговой адресации нет необходимости вследствие меньших расстояний перехода. В микроконтроллерах AVR с большим объемом памяти команд следует использовать команды вызова `call`.

Примеры к рис. 3.27:

16-разрядное командное слово

`rjmp`	1100	kkkk	kkkk	kkkk
`rcall`	1101	kkkk	kkkk	kkkk

```
Адрес   Код     Метка   Команда
.device AT90S8515
.list
.org $01a
00001a  cfd5    label1: rjmp label 2
.org 0xff0
000ff0  c029    label2: rjmp label 1
```

Время доступа к памяти и время выполнения команд

Тактирование центрального процессора микроконтроллеров AVR выполняется непосредственно через такт системной синхронизации **Ф**, который вырабатывается встроенным в кристалл кварцевым генератором колебаний. Частота кварцевого генератора идентична тактовой частоте и в дальнейшем внутренне не делится. На рис. 1.1 показана параллельная выборка и выполнение команд. Это основополагающая концепция конвейеризации, позволяющая выполнять миллионы команд за один МГц тактовой частоты.

Сброс и обработка прерываний

Сброс во всех микроконтроллерах AVR обеспечивает запуск заданной программы, начиная с адреса 0, и заполнение внутреннего регистра управления заранее определенными значениями в соответствии с табл. 3.1.

В зависимости от модели, микроконтроллеры AVR также могут обрабатывать различное число источников прерываний, то есть, прерывания выполняющейся программы от внешних или внутренних процессов.

Этим источникам прерываний, точно так же как неизменно определенным для сброса адресам входа в память программ, назначены так называемые векторы программ и векторы прерываний. Всем прерываниям поставлены в соответствие отдельные разряды разрешения (Enable), которые устанавливаются в лог. 1, если прерывания разрешены. Однако, если разряд I (бит 7) в регистре состояния сброшен (лог. 0), то запрещены вообще все прерывания, независимо от состояния отдельных разрядов разрешения. Самые младшие адреса в памяти программ зарезервированы для векторов сброса и прерываний. Полный перечень предопределенных векторов (согласно приоритетности) представлен в табл. 3.6.

Таблица 3.6. Адреса сброса и векторы прерываний микроконтроллеров AVR

Прогр. адрес в 90S1200	*Прогр. адрес в 90S2313*	*Прогр. адрес в 90S4414 90S8515*	*Источник прерывания*	*Описание прерывания*
$000	$000	$000	RESET	Включение питания, вывод /RST и сторожевой таймер
$001	$001	$001	INT0	Внешний запрос на прерывание 0
–	$002	$002	INT1	Внешний запрос на прерывание 1
–	$003	$003	TIMER1 CAPT	Захват таймера/счетчика 1
–	$004	$004	TIMER1 COMPA	Совпадение A таймера/счетчика 1
–	–	$005	TIMER1 COMPB	Совпадение B таймера/счетчика 1
–	$005	$006	TIMER1 OVF	Переполнение таймера/счетчика 1
$002	$006	$007	TIMER0 OVF	Переполнение таймера/счетчика 0
–	–	$008	SPI, STC	Передача по интерфейсу SPI завершена
–	$007	$009	UART, RX	UART: прием байта завершен
–	$008	$00A	UART, UDRE	Регистр данных UART пуст
–	$009	$00B	UART, TX	UART: передача данных завершена
$003	$00A	$00C	ANA_COMP	Аналоговый компаратор

Чем меньше адрес вектора прерываний, тем выше приоритетность прерывания. Другими словами, вектор сброса RESET имеет наивысший приоритет, затем следует внешний запрос на прерывание INT 0 и т.д. На рис. 3.28 показано типичное начало программы для микроконтроллера AT90S8515.

```
Адрес Метка   Команда              Комментарий
$000          rjmp INITIAL         ; Подпрограмма обработки сброса
$001          rjmp EXT_INT0        ; Подпрограмма внешнего прерывания 0
$002          rjmp EXT_INT1        ; Подпрограмма внешнего прерывания 1
$003          rjmp TIME1_CAPT      ; Подпрограмма захвата таймера 1
$004          rjmp TIME1_COMPA     ; Подпрограмма сравнения таймера 1 с A
$005          rjmp TIME1_COMPB     ; Подпрограмма сравнения таймера 1 с B
$006          rjmp TIME1_OVFL      ; Подпрограмма переполнения таймера 1
$007          rjmp TIME0_OVFL      ; Подпрограмма переполнения таймера 0
$008          rjmp SPI_HANDLE      ; Передача через SPI завершена
$009          rjmp UART_RxC        ; Устройство UART: байт принят
$00A          rjmp UART_DRE        ; Устройство UART: регистр данных пуст
$00B          rjmp UART_TxC        ; Устройство UART: передача завершена
$00C          rjmp ANA_COMP        ; Аналоговый компаратор

$100  MAIN:   <Команды>            ; Начало основной программы
```

Рис. 3.28. Пример типичного начала пользовательской программы

После сброса, как правило, происходит инициализация используемых в пользовательской программе регистров и таких периферийных функций как стек, UART, SPI, таймер и т.д. Для этого в представленном выше примере выполняется переход к части программы `INITIAL`. После этого может быть начато собственно выполнение программы, начиная с метки `MAIN`.

С помощью команды `rjmp` происходит переход по тому или иному программному адресу прерывания, и начинается выполнение соответствующей подпрограммы обслуживания прерывания. После выполнения этой подпрограммы происходит возврат к прерванному участку программы.

Варианты сброса в микроконтроллерах AVR:

- сброс при включении питания – микроконтроллер сбрасывается, если между выводами V_{CC} и GND появляется рабочее напряжение;
- внешний сброс — микроконтроллер сбрасывается, если на выводе /RST появляется уровень лог. 0;
- сброс от сторожевого таймера — микроконтроллер сбрасывается по истечению времени, заданного сторожевым таймером, если этот таймер был разрешен.

Во время сброса все регистры автоматически инициализируются предопределенным исходным значением в соответствии с табл. 3.1. После поступления сигнала сброса выполнение программы начинается с адреса $000. По адресу $000 должна находиться команда `rjmp` — команда переход к той части программы, в которой расположены команды инициализации, специфические для пользователя (например, инициализация таймеров и UART, и т.д.).

По адресу $000 может быть введен обычный программный код только в том случае, когда программа не использует никаких источников прерываний (то есть, все векторы прерываний остаются неиспользованными).

Блок-схема, представленная на рис. 3.29, демонстрирует схему сброса микроконтроллеров AVR.

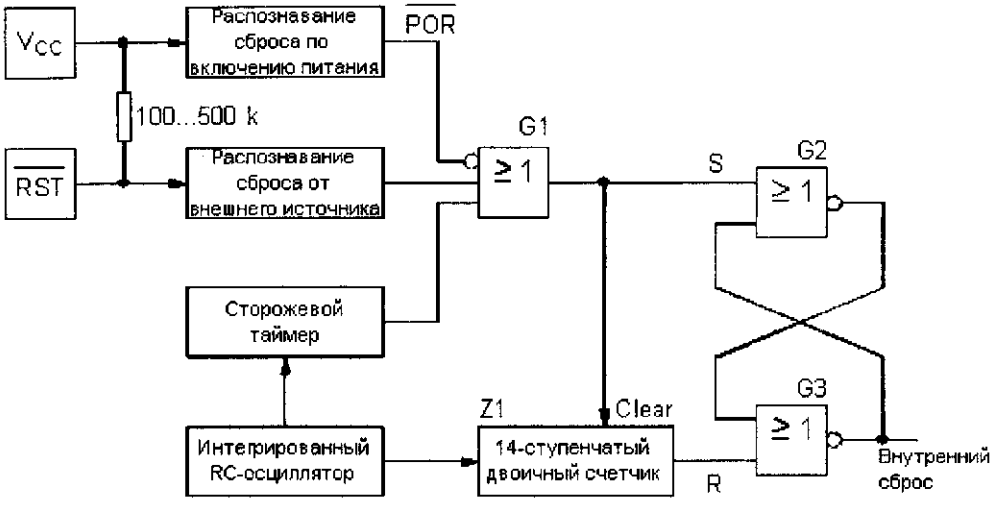


Рис. 3.29. Схема сброса микроконтроллеров AVR

Каждый из трех возможных источников сброса может быть определен с помощью вентиля "ИЛИ" G1, который устанавливает RS-триггер, образованный двумя вентилями "НЕ-ИЛИ" G2 и G3. Выходным сигналом RS-триггера является внутренний сигнал сброса (лог. 1). Одновременно выходной сигнал с вентиля G1 переводит 14-ступенчатый двоичный счетчик Z1, тактированный от внутреннего RC-осциллятора, в исходное нулевое состояние и удерживает его в таком состоянии до тех пор, пока активен соответствующий источник сброса.

Как только выходной сигнал вентиля G1 изменяется на лог. 0, начинает работать двоичный счетчик Z1. Когда счетчик Z1 достигает некоторого заранее заданного состояния, выходной сигнал Z1 сбрасывает RS-триггер, и внутренний сигнал сброса будет снят (лог. 0).

Эта задержка t_{TOUT} внутреннего сигнала сброса необходима по той причине, что внутреннему генератору колебаний после подачи рабочего напряжения на кристалл кварца требуется несколько миллисекунд для того, чтобы колебания стали стабильными.

В прикладных системах, у которых внутренний тактовый сигнал генерируется не кварцем, а, например, керамическим резонатором, при котором генератор колебаний достигает постоянства частоты значительно быстрее, возможен более короткий сигнал сброса. Этому может способствовать то, что во флэш-памяти программируется разряд предохранения FSTRT. Более подробно это описано в главе 11, "Программирование памяти". В табл. 3.7 представлены электрические и динамические параметры схемы сброса для напряжения питания V_{CC} = 5 В.

Таблица 3.7. Электрические и динамические параметры схемы сброса

Обозначение	*Параметр*	*Минимальное*	*Типичное*	*Максимальное*	*Единица измерения*
V_{POT}	Пороговое напряжение сброса при подаче питания	1,8	2	2,2	В
$V_{/RST}$	Пороговое напряжение на выводе /RST		$V_{CC}/2$	0,7 V_{CC}	В

Таблица 3.7. Окончание

Обозначение	Параметр	Минимальное	Типичное	Максимальное	Единица измерения
t_{POR}	Продолжительность сброса при подаче питания	2	3	4	мс
t_{TOUT}	Задержка сброса, если FSTRT *) не запрограммирован	11	16	21	мс
t_{TOUT}	Задержка прерывания, если FSTRT *) запрограммирован	1,0	1,1	1,2	мс

*) FSTRT в микроконтроллере AT90S1200 отсутствует

Частота внутреннего RC-осциллятора зависит от питающего напряжения V_{CC}. Для V_{CC} = 5 В она составляет приблизительно 1,1 МГц. Частоты для других значений питающего напряжения могут быть взяты из рис. 2.10.

Сброс по включению питания

Схема сброса по включению питания (POR — **P**ower-**O**n **R**eset), показанная на рис. 3.29, обеспечивает сброс микроконтроллера только в том случае, когда рабочее напряжение V_{CC} достигнет минимальной величины V_{POT} (**P**ower-**O**n **T**hreshold — порог включения). Для этого нарастающее рабочее напряжение сравнивается с пороговым напряжением включения V_{POT}. Только в том случае, когда рабочее напряжение превышает напряжение включения V_{POT}, начинается отсчет времени активности сигнала сброса t_{POR} (рис. 3.30).

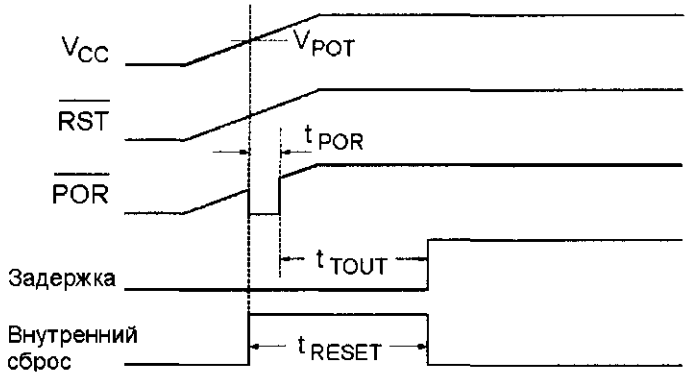


Рис. 3.30. Фаза запуска микроконтроллера AVR: вывод /RST не подключен (внутренний подтягивающий резистор) или связан с V_{CC}; сброс вызывается нарастанием рабочего напряжения V_{CC}

Полная продолжительность t_{RESET} внутреннего сигнала сброса является суммой активного времени сброса t_{POR} и задержки t_{TOUT}, вызванной счетчиком Z1.

Для стабильного включения питания должны быть выполнены следующие условия.

- Если в прикладных случаях применения нет необходимости во внешнем сигнале сброса и не требуется внутрисистемное программирование, а время нарастания питающего напряжения V_{CC} должно быть меньше, чем суммарное время сброса $t_{RESET} = t_{POR} + t_{TOUT}$, то на вывод /RST микроконтроллера можно подать питающее напряжение V_{CC} или оставить его свободным

(без внутреннего подтягивающего резистора) (см. левую часть рис. 3.31). Однако, если для прикладного случая применения требуется внешняя кнопка сброса или необходимо, чтобы микроконтроллер мог программироваться при работе в составе системы, то должна применяться схема, показанная в правой части рис. 3.31.

- Если время нарастания питающего напряжения V_{CC} превышает суммарное время $t_{RESET} = t_{POR} + t_{TOUT}$, то для сброса при включении питания должна быть применена внешняя схема. Эта схема должна поддерживать низкий потенциал на выводе /RST микроконтроллера до тех пор, пока не будет обеспечено то, что напряжение питания V_{CC} за оставшееся время достигнет своего заданного значения t_{TOUT} (см. рис. 3.30). Пригодная для этой цели электрическая схема подключения вывода /RST показана на рис. 3.32.

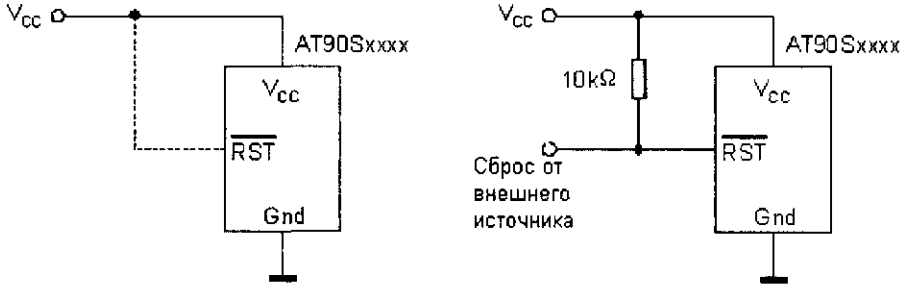


Рис. 3.31. Подсоединение вывода /RST без возможности программирования (слева) и с возможностью программирования при работе в составе системы (справа)

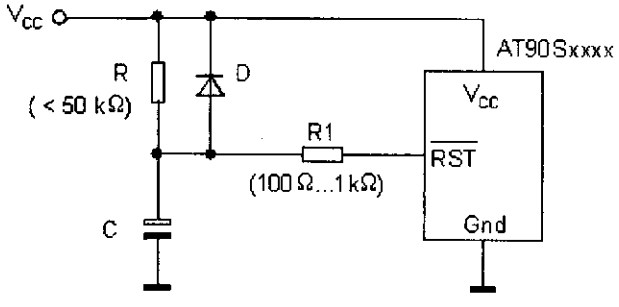


Рис. 3.32. Подключение вывода /RST к схеме с RC-звеном для сброса по включению питания

Удлинение фазы включения, образованное звеном задержки из сопротивления R и емкости C, может быть вычислено по формуле $\tau = RC$, при этом τ — это постоянная времени RC-звена, то есть, время, необходимое для того, чтобы напряжение на конденсаторе поднялось до 63% своего конечного значения.

Диод D предназначен для разрядки конденсатора C при отключении рабочего напряжения. Благодаря этому, предотвращается появление слишком коротких импульсов сброса при повторном включении еще в значительной степени заряженного конденсатора. Сопротивление R1 защищает вывод /RST от слишком больших величин тока на входе при полностью заряженном конденсаторе C.

Тем не менее, с помощью этой схемы не могут быть предотвращены так называемые провалы напряжения (Brown-Out), при которых рабочее напряжение

кратковременно падает ниже минимальной величины $V_{CC\ min}$, поскольку напряжение на выводе /RST не достаточно мало для уверенного обеспечения сброса.

Для того чтобы надежно предотвратить провалы напряжения, можно воспользоваться интегральными схемами сброса, которые предлагают многие фирмы-изготовители. Так, например, компания MAXIM предлагает интегральную схему сброса MAX809 с активным низким уровнем сигнала, в корпусе SOT23 с тремя выводами для различных систем с пороговыми напряжениями 3 В, 3,3 В и 5 В. Если необходима также кнопка внешнего сброса, то может быть применена схема MAX811 в корпусе SOT143 с четырьмя выводами, свойства которой идентичны интегральной схеме MAX809 вплоть до подключения дополнительных кнопок (ключей) (рис. 3.33).

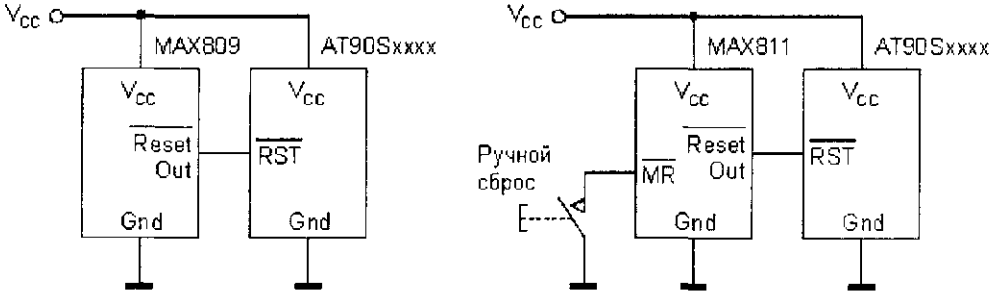


Рис. 3.33. Генерирование сигнала сброса при включении питания с использованием микросхем MAX809 и MAX811 от компании MAXIM

По заявлению компании Atmel, в будущем микроконтроллеры семейства AVR будут располагать блоком для распознавания провалов напряжения. Предлагаются также схемы в комбинации с другими периферийными функциями. Одним из примеров таких схем является устройство S24022 компании Summit Microelectronics, которое представляет собой запоминающее устройство с памятью 2 Кбайт типа EEPROM, подключением к шине I^2C и точной схемой сброса.

Генерирование сигналов сброса при включении питания с помощью специально разработанных для этой цели схем предпочтительнее также и потому, что в случае схемы сброса с простым RC-звеном, в противоположность интегрированным схемам, при отключении рабочего напряжения сброс вообще невозможен. Однако некоторые блоки в схемах сброса внутри микроконтроллеров ряда изготовителей работают при меньших значениях напряжения, чем другие. Пока рабочее напряжение понизится до уровня, достаточного для прекращения работы блоков в схеме, может случиться так, что микроконтроллер будет совершать неконтролируемые операции, выполняя до сигнала сброса нежелательные команды.

Важное предупреждение! Для микроконтроллеров моделей AT90S1200 и AT90S8515 следует обязательно обращать внимание на то, чтобы время увеличения напряжения питания V_{CC} от 0 до 2,2 В в любом случае было меньше 100 мс! В том случае, если напряжение питания во время работы упадет ниже уровня 1 В, в течение 100 мс оно снова должно достичь уровня 2,2 В. Это требование действует также и тогда, когда применяется внешняя схема сброса. Если время увеличения напряжения V_{CC} от 0 до 2,2 В превышает 100 мс, то микроконтроллер не будет работать надлежащим образом и его сброс может быть выполнен только посредством снижения напряжения V_{CC} до нулевого уровня.

Если в некоторых приложениях требуется длительность сброса по включению питания больше длительности, заранее заданной величинами t_{POR} и t_{TOUT}, то этого можно достичь следующим образом. При включении рабочего напряжения с помощью нажатия кнопки или RC-звена на выводе /RST уже будет установлен уровень лог. 0 (см. рис. 3.32). Счетчик Z1, показанный на рис. 3.29, в этом случае начинает работать только тогда, когда внешний сигнал на выводе /RST переходит в состояние лог. 1 (рис. 3.34). Внутренний сигнал сброса в этом случае также активизируется при достижении порогового напряжения включения V_{POT}, однако его длительность можно увеличить до любого значения, увеличив длительность сигнала на выводе /RST.

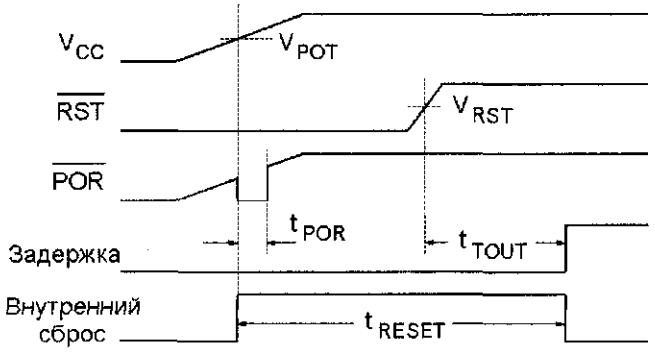


Рис. 3.34. Фаза запуска микроконтроллера AVR: продолжительность внутреннего сброса увеличивается, благодаря внешнему сигналу на выводе /RST

Внешний сброс

Внешний или аппаратный сброс реализуется подачей сигнала низкого уровня на вывод /RST. Сигнал низкого уровня должен сохраняться как минимум два периода такта системной синхронизации. Нарастающий фронт сигнала на выводе сброса /RST после превышения порогового напряжения включения V_{RST} приводит в действие счетчик задержки Z1 (рис. 3.35), и, следовательно, — внутренний сигнал сброса. По истечение времени t_{TOUT} внутренний процесс сброса заканчивается, и микроконтроллер переходит к выполнению программы, начиная с адреса 0.

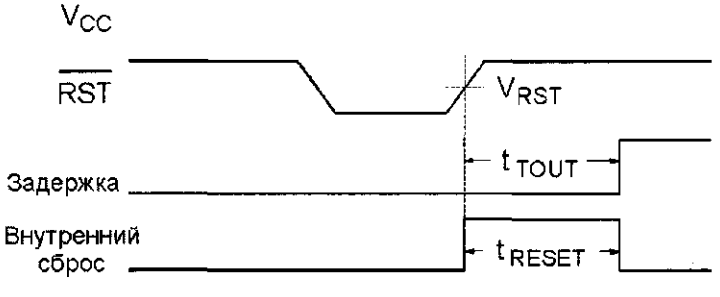


Рис. 3.35. Внешний сброс во время выполнения программы

Сброс от сторожевого таймера

При срабатывании сторожевой таймер генерирует короткий импульс сброса продолжительностью один период такта системной синхронизации. Нарастающий фронт этого импульса инициирует внутренний процесс сброса, а ниспадающий фронт активизирует счетчик задержки (рис. 3.36).

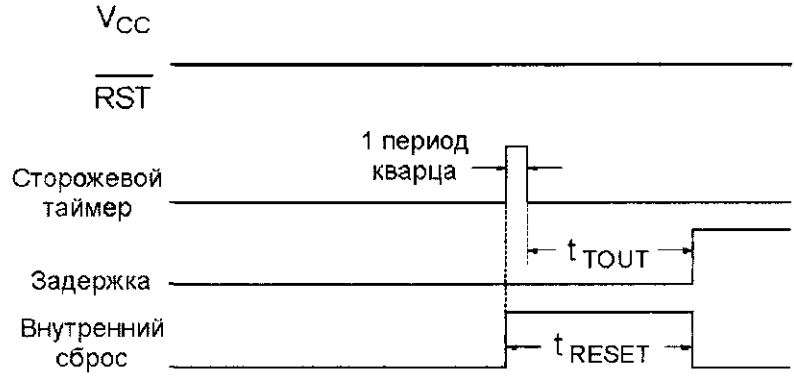


Рис. 3.36. Сброс по сторожевому таймеру во время выполнения программы

После формирования задержки t_{TOUT} внутренний процесс сброса заканчивается, и микроконтроллер переходит к выполнению программы, начиная с адреса 0. Подробнее этот процесс рассматривается в главе 5, "Сторожевой таймер".

Обработка прерываний

В микроконтроллерах базовой серии семейства AVR за обработку прерываний, в первую очередь, отвечают два регистра:

- GIMSK (**G**eneral **I**nterrupt **M**ask Register), который разрешает или запрещает внешние прерывания INT0 и INT1;
- TIMSK (**T**imer/Counter **I**nterrupt **M**ask Register — регистр маски прерываний от таймера/счетчика), который управляют прерываниями во взаимосвязи с таймерами/счетчиками T/C0 и T/C1.

О состоянии соответствующего прерывания сигнализирует его флаг. Флаги для внешних прерываний находятся в общем регистре флагов прерываний GIFR (**G**eneral **I**nterrupt **F**lag **R**egister), а флаги для различных таймеров и счетчиков — в регистре флагов TIFR (**T**imer/Counter **I**nterrupt **F**lag **R**egister). Если возникает прерывание, то в регистре флагов прерываний устанавливается соответствующий флаг.

Прерывания могут (даже если в регистре маски прерываний установлен соответствующий отдельный разряд разрешения прерывания) стать активными только тогда, когда в регистре состояния установлен разряд общего разрешения прерываний I. Если это имеет место, и наступает прерывание, то выполнение программы ответвляется по соответствующему адресу (см. табл. 3.6) и разряд общего разрешения прерываний I в регистре состояния сбрасывается в состояние лог. 0, чем дальнейшие прерывания блокируются. Если требуется возможность прервать

подпрограмму другим прерыванием, то после входа в подпрограмму обработки прерывания программа пользователя должна установить флаг I в лог. 1.

Вместе с входом в подпрограмму обработки прерывания аппаратно сбрасывается также и соответствующий флаг, вызвавший прерывание. Некоторые флаги прерываний могут быть сброшены самим пользователем посредством установки соответствующего флага в лог. 1.

Время ответа на прерывание

Время реагирования, прошедшее от наступления прерывания вплоть до выполнения соответствующей подпрограммы, для всех видов прерываний составляет минимум четыре цикла такта системной синхронизации. В течение этих четырех тактов два байта адреса возврата будут сохранены в стеке, указатель стека будет декрементирован на 2, а также будет выполнен переход по адресу соответствующей подпрограммы. Если прерывание наступает во время выполнения команды, состоящей из нескольких циклов, то в любом случае выполнение этой команды будет закончено а затем будет обработано прерывание.

Возврат из подпрограммы обработки прерывания выполняется (как и при возврате из обычной подпрограммы) также за четыре тактовых цикла. В течение этих четырех тактов системной синхронизации два байта адреса возврата будут перенесены из стека в счетчик команд, и указатель стека будет инкрементирован на 2. Возврат из подпрограммы обработки прерывания реализован, как правило, с помощью команды `reti`. Эта команда не только приводит к возврату к месту прерывания программы, но также устанавливает разряд I в регистре состояния для того, чтобы снова разрешить прерывания.

Если после возврата из подпрограммы обработки прерывания обнаружится прерывание, находящееся в режиме ожидания, то до его обработки будет выполнена как минимум одна команда основной программы.

Важное предупреждение! Содержимое регистра состояния SREG ни при прерываниях, ни при выполнении обычных подпрограмм автоматически не сохраняется. Сохранение регистра SREG, а также его восстановление при выходе из подпрограммы — ответственность пользователя.

В случае прерываний, вызванных событиями, которые могут продолжаться длительное время (например, согласование состояния счетчика T/C1 со значением в выходном регистре сравнения A), флаг прерывания устанавливается при наступлении такого события. Если флаг прерывания был сброшен в результате выполнения подпрограммы обработки прерывания или вручную, то он не будет установлен до наступления нового события.

Внешние прерывания

Внешние прерывания вызываются сигналами на выводах INT0 и INT1. Будет ли прерывание вызвано уровнем лог. 0 или нарастающим/ниспадающим фронтом сигнала, определяется с помощью разрядов ISC11 и ISC10 для INT1 и, соответственно, ISC01 и ISC00 для INT0 в регистре управления MCUCR. Если внешнее или аппаратное прерывание было разрешено по уровню сигнала, то прерывания будут вызываться до тех пор, пока на соответствующем входе INTx находится сигнал низкого уровня.

Если внешние прерывания разрешены разрядом разрешения INTx (x = 0 или 1) в регистре GIMSK, то они вызываются даже в том случае, когда INT0/INT1 сконфигурированы как выход. Это предоставляет пользователю возможность использовать программные прерывания.

Регистр GIMSK

Разрешения на внешние прерывания вырабатываются через регистр GIMSK (General Interrupt **Mask** Register), который находится по адресу $3B ($5B) в области ввода/вывода. После поступления сигнала сброса он инициализируется значением $00.

В микроконтроллерах AT90S8515, AT90S4414 и AT90S2313 используются только разряды 6 и 7 регистра GIMSK. Они доступны как на чтение, так и на запись. Разряды 0–5 зарезервированы компанией Atmel для дальнейшего расширения и доступны только для чтения (всегда содержат лог. 0).

В микроконтроллере AT90S1200 используется только разряд 6, поскольку у этой модели отсутствует INT1. Разряд доступен для чтения и записи. Разряды 0–5 и 7 зарезервированы компанией Atmel и доступны только для чтения (всегда содержат лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$3B ($5B)	INT1 *)	INT0	–	–	–	–	–	–	GIMSK

*) INT1 в модели AT90S1200 отсутствует

Если разряд **INT1** установлен в лог. 1, то внешнее прерывание 1 будет разрешено до тех пор, пока будет установлен разряд общего разрешения прерываний в регистре состояния. Будет ли прерывание вызываться по фронту сигнала или же по уровню лог. 0 на выводе INT1, определяют разряды ISC11 и ISC10 в регистре MCUCR. Адрес вызова внешнего прерывания 1 для соответствующего микроконтроллера семейства AVR может быть взят из табл. 3.6.

Если разряд **INT0** установлен в лог. 1, то будет разрешено внешнее прерывание 0 до тех пор, пока в регистре состояния будет установлен разряд общего разрешения прерываний. Будет ли прерывание вызываться по фронту сигнала или же по уровню лог. 0 на выводе INT0, определяют разряды ISC01 и ISC00 в регистре MCUCR. Адрес внешнего прерывания 0 — $001 для всех микроконтроллеров базовой серии семейства AVR.

Регистр GIFR

Состояние внешнего прерывания определяется по регистру GIFR (General Interrupt **F**lag **R**egister — общий регистр флагов прерываний). В моделях AT90S8515, AT90S4414 и AT90S2313 он находится по адресу $3A ($5A) в области ввода/вывода. После поступления сигнала сброса регистр GIFR инициализируется записью значения $00. В микроконтроллере AT90S1200 регистр GIFR отсутствует.

В моделях AT90S8515, AT90S4414 и AT90S2313 используются только разряды 6 и 7 регистра GIFR, доступные для чтения и записи. Разряды 0–5 зарезервированы компанией Atmel и доступны только для чтения (всегда содержат лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$3A ($5A)	INTF1	INTF0	–	–	–	–	–	–	GIFR *)

*) В микроконтроллере AT90S1200 регистр GIFR отсутствует

Флаг **INTF1** (External **Int**errupt **F**lag 1 — флаг 1 внешних прерываний) устанавливается в лог. 1, если сигнал на выводе INT1 вызывает внешнее прерывание 1.

В случае, если установлен разряд общего разрешения прерываний в регистре состояния и разряд INT1 в регистре GIMSK, то выполнение программы продолжается с соответствующего адреса INT1 микроконтроллера ($002 для AT90S8515, AT90S4414 и AT90S2313; в модели AT90S1200 разряд INT1 отсутствует).

При входе в подпрограмму обработки прерывания разряд INTF1 переводится в исходное состояние аппаратно. Альтернативно, разряд INTF1 может быть сброшен в состояние лог. 0 путем записи лог. 1 в разряд 7 регистра GIFR.

Флаг **INTF0** (External **Int**errupt **F**lag 0 — флаг 0 внешних прерываний) устанавливается в лог. 1, когда сигнал на выводе INT0 вызывает внешнее прерывание 0.

В том случае, если установлен разряд общего разрешения прерываний в регистре состояния и разряд INT0 в регистре GIMSK, выполнение программы продолжается с соответствующего адреса INT0 микроконтроллера ($001 для всех микроконтроллеров базовой серии семейства AVR).

При входе в подпрограмму обработки прерывания разряд INTF0 переводится в исходное состояние аппаратно. В микроконтроллерах AT90S8515, AT90S4414 и AT90S2313 разряд INTF0 может быть сброшен в состояние лог. 0 также посредством записи лог. 1 в разряд 6 регистра GIFR.

В модели AT90S1200 регистр GIFR отсутствует, поэтому прямой доступ пользователя к флагу прерываний для единственного доступного внешнего прерывания INT0 невозможен. Несмотря на это, в случае запуска с управлением по фронту сигнала, флаг INTF0 может быть сброшен следующим образом:

- блокировка INT0 посредством сброса разряда 6 в регистре GIMSK;
- установка запуска с управлением по уровню сигнала с помощью разрядов ISC01 и ISC00 в регистре MCUCR;
- выбор запуска с управлением по фронту сигнала с помощью разрядов ISC01 и ISC00 в регистре MCUCR;
- повторное разрешение INT0 посредством установки разряда 6 в регистре GIMSK.

Регистр TIMSK

Разрешение различных прерываний в связи с таймерами/счетчиками T/C0 и T/C1 осуществляется через регистр TIMSK (**T**imer/Counter **I**nterrupt **M**ask **S**egister — регистр маски прерываний от таймеров/счетчиков), который находится по адресу $39 ($59) в области ввода/вывода. После поступления сигнала сброса регистр TIMSK инициализируется значением $00.

В микроконтроллерах AT90S8515 и AT90S4414 используются только разряды 1, 3 и 5–7 регистра TIMSK, доступные для чтения и записи. Разряды 0, 2 и 4 заре-

зервированы компаний Atmel и доступны только для чтения (всегда содержат лог. 0).

В модели AT90S2313 используются только разряды 1, 3, 6 и 7 регистра TIMSK (для чтения и записи). Разряды 0, 2, 4 и 5 зарезервированы компанией Atmel и доступны только для чтения (всегда содержат лог. 0).

В микроконтроллере AT90S1200 используется только разряд 1 (для чтения и записи). Все остальные разряды зарезервированы и доступны только для чтения (всегда содержат лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$39 ($59)	TOIE1 *)	OCIE1A *)	OCIE1B **)	–	TICIE1 *)	–	TOIE0	–	TIMSK

*) в модели AT90S1200 отсутствует

**) в моделях AT90S1200 и AT90S2313 отсутствует

Когда разряд **TOIE1** (**T**imer/Counter1 **O**verflow **I**nterrupt **E**nable — разрешение прерывания по переполнению таймера/счетчика) и разряд общего разрешения прерываний в регистре состояния SREG установлены в лог. 1, то разрешено прерывание при переполнении T/C1 при состоянии счетчика $0000. В случае переполнения в регистре TIFR (**T**imer/Counter **I**nterrupt **F**lag **R**egister — регистр флагов прерываний от таймера/счетчика) устанавливается флаг TOV1. Если таймер/счетчик T/C1 находится в режиме ШИМ, то при изменении счетчиком направления счета на обратное при состоянии счетчика $0000 устанавливается флаг TOV1. Адрес перехода к подпрограмме обработки прерывания по переполнению T/C1 для различных микроконтроллеров семейства AVR может быть взяты из табл. 3.6.

Если разряд **OCIE1A** (Timer/Counter1 **O**utput **C**ompare**A** Match **I**nterrupt **E**nable) и разряд общего разрешения прерываний в регистре состояния SREG установлены в лог. 1, то разрешено прерывание при совпадении содержимого регистра сравнения A с текущим состоянием T/C1. В случае совпадения, в регистре TIFR устанавливается флаг OCF1A. Адрес перехода к соответствующей подпрограмме одного из микроконтроллеров семейства AVR может быть взят из табл. 3.6.

Если разряд **OCIE1B** (Timer/Counter1 **O**utput **C**ompare**B** Match **I**nterrupt **E**nable) и разряд общего разрешения прерываний в регистре состояния SREG установлен в лог. 1, то разрешается прерывание при совпадении содержимого регистра сравнения B с текущим состоянием T/C1. В случае совпадения, в регистре TIFR устанавливается флаг OCF1B. Адрес перехода к соответствующей подпрограмме для одного из микроконтроллеров AVR может быть взят из табл. 3.6.

Если разряд **TICIE1** (**T**imer/Counter1 **I**nput **C**apture **I**nterrupt **E**nable) и разряд общего разрешения прерываний в регистре состояния SREG установлены в лог. 1, то разрешается прерывание при выполнении условия захвата с помощью сигнала на выходе X мультиплексора Mux1 (см. рис. 4.7). Когда наступает запуск по захвату (Capture-Triggering), то в регистре TIFR устанавливается флаг ICF1. Адрес

перехода к соответствующей подпрограмме для одного из микроконтроллеров семейства AVR может быть взят из табл. 3.6.

Если разряд **TOIE0** (**T**imer/Counter**0** **O**verflow **I**nterrupt **E**nable) и разряд общего разрешения прерываний в регистре состояния SREG установлены в лог. 1, то разрешается прерывание при переходе таймера/счетчика T/C0 от $FF к $00. В таком случае, в регистре TIFR устанавливается разряд TOV0. Адрес перехода к подпрограмме обработки переполнения T/C0 для различных микроконтроллеров семейства AVR может быть взят из табл. 3.6.

Регистр TIFR

Состояние прерываний, имеющих отношение к таймерам/счетчикам T/C0 и T/C1, определяется с помощью регистра флагов TIFR (**T**imer/Counter **I**nterrupt **F**lag **R**egister), который расположен в области ввода/вывода по адресу $38 ($58). После поступления сигнала сброса этот регистр инициализируется значением $00.

В микроконтроллерах AT90S8515 и AT90S4414 применяются только разряды 1, 3 и 5–7 регистра TIFR (для чтения и записи). Разряды 0, 2 и 4 зарезервированы компанией Atmel и доступны только для чтения (всегда содержат лог. 0).

В модели AT90S2313 используются только разряды 1, 3, 6 и 7 регистра TIFR (для чтения и записи). Разряды 0, 2, 4 и 5 зарезервированы и доступны только для чтения (всегда содержат лог. 0).

В микроконтроллерах AT90S1200 используется только разряд 1, доступный для чтения и записи. Все другие разряды зарезервированы и доступны только для чтения (всегда содержат лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$38 ($58)	TOV1 *)	OCF1A *)	OCF1B **)	–	ICF1 *)	–	TOV0	–	TIFR

*) в модели AT90S1200 отсутствует

**) в микроконтроллерах AT90S1200 и AT90S2313 отсутствуют

Флаг **TOV1** (**T**imer/Counter1 **O**verflow Flag — переполнение таймера/счетчика) устанавливается в лог. 1, когда таймер/счетчик T/C1 переходит в состояние $0000. Посредством установки флага TOV1 выполнение программы переходит к соответствующему адресу обработки переполнения T/C1 микроконтроллера ($006 для моделей AT90S8515 и AT90S4414, $005 для модели AT90S2313; в микроконтроллере AT90S1200 таймер/счетчик T/C1 отсутствует) в том случае, если установлен разряд общего разрешения прерываний I и разряд TOIE1 в регистре TIMSK. При входе в подпрограмму обработки прерывания флаг TOV1 сбрасывается аппаратно. Альтернативно, флаг TOV1 может быть сброшен программно посредством записи в разряд 7 регистра TIFR лог. 0. Если таймер/счетчик T/C1 находится в режиме ШИМ, то устанавливается флаг TOV1, когда счетчик при состоянии $0000 меняет свое направление счета на обратное.

Флаг **OCF1A** (**O**utput **C**ompare **F**lag **1A**) устанавливается в лог. 1 при совпадении содержимого регистра сравнения A с текущим состоянием счетчика T/C1. Посредством установки флага OCF1A выполнение программы переходит к соответствующему адресу микроконтроллера ($004 для моделей AT90S8515, AT90S4414

и AT90S2313; в модели AT90S1200 таймер/счетчик T/C1 отсутствует) в том случае, если установлен разряд общего разрешения прерываний в регистре состояния и разряд OCIE1A в регистре TIMSK. При входе в подпрограмму обработки прерывания флаг OCF1A сбрасывается аппаратно. Альтернативно, флаг OCF1A может быть сброшен программно в лог. 0 посредством записи лог. 1 в разряд 6 регистра TIFR.

Флаг **OCF1B** (**O**utput **C**ompare **F**lag **1B**) устанавливается в лог. 1 при совпадении содержимого регистра сравнения B с текущим состоянием таймера/счетчика T/C1. Посредством установки флага OCF1B выполнение программы переходит к соответствующему адресу микроконтроллера ($005 для моделей AT90S8515, AT90S4414 и AT90S2313; в модели AT90S1200 регистр сравнения B отсутствует) в том случае, если установлен разряд общего разрешения прерываний I в регистре состояния и разряд OCIE1B в регистре TIMSK. При входе в подпрограмму обработки прерывания флаг OCF1B сбрасывается аппаратно. Альтернативно, флаг OCF1A может быть сброшен программно в лог. 0 посредством записи лог. 1 в разряд 5 регистра TIFR.

Флаг **ICF1** (**I**nput **C**apture **F**lag — флаг захвата по входу) устанавливается в лог. 1 в случае выполнения условия захвата при поступлении сигнала на вывод ICP. Таким образом, флаг ICF1 указывает на то, что в результате запуска при захвате содержимое таймера/счетчика T/C1 было передано в регистр ICR1 (**I**nput **C**apture **R**egister). Посредством установки флага ICF1 выполнение программы переходит к соответствующему адресу микроконтроллера ($003 для моделей AT90S8515, AT90S4414 и AT90S2313; в микроконтроллере AT90S1200 таймер/счетчик T/C1 отсутствует) в том случае, если установлен разряд общего разрешения прерываний I в регистре состояния и разряд TICIE1 в регистре TIMSK. При входе в подпрограмму обработки прерывания флаг ICF1 сбрасывается аппаратно. Альтернативно, флаг ICF1 может быть сброшен программно в лог. 0 посредством записи лог. 1 в разряд 3 регистра TIFR.

Флаг **TOV0** (**T**imer/**C**ounter**0** **O**verflow Flag) устанавливается в лог. 1, когда таймер/счетчик T/C0 переход из состояния $FF в состояние $00. Посредством установки флага TOV0 выполнение программы переходит к соответствующему адресу микроконтроллера ($007 для моделей AT90S8515 и AT90S4414; $006 для модели AT90S2313; $002 для моделиAT90S1200) в том случае, если установлен разряд общего разрешения прерываний I в регистре состояния и разряд TOIE0 в регистре TIMSK. При входе в подпрограмму обработки прерывания флаг TOV0 сбрасывается аппаратно. Альтернативно, флаг TOV0 может быть сброшен программно в лог. 0 посредством записи лог. 1 в разряд 1 регистра TIFR.

"Спящие" режимы центрального процессора

Микроконтроллеры нередко применяются в приборах с питанием от аккумуляторов и батарей. В таких случаях особенно важно, чтобы потребление тока микроконтроллером было как можно меньшим. Микроконтроллеры семейства AVR изготовлены по технологии КМОП, и потому для их работы необходим ток небольшой силы. Благодаря выбору более низкого уровня питающего напряжения,

потребление тока снижается еще больше. Если и этого недостаточно, то можно выбрать настолько низкую тактовую частоту, насколько позволяет прикладная задача, поскольку из-за токов перезарядки паразитных емкостей ток потребления КМОП-схем почти пропорционален тактовой частоте. В дополнение к перечисленным мерам, центральный процессор микроконтроллеров семейства AVR можно перевести в *режим пониженного энергопотребления* или "спящий режим" (Sleep Mode). В таком режиме потребление тока резко снижается.

В табл. 3.8 показано типичное потребление тока центральным процессором модели AT90S1200 для различных режимов работы при температуре окружающей среды 25^{0}C, отключенном аналоговом компараторе и заблокированном сторожевом таймере.

Таблица 3.8. Типичное потребление тока I_{CC} центральным процессором модели AT90S1200 для различных режимов работы

V_{CC}	Φ	I_{CC} / Активный	I_{CC} / Ждущий	I_{CC} / Пониженное потребление
5 В	12 МГц	9 мА	2,4 мА	0,75 мкА
5 В	2 МГц	4 мА	1 мА	0,75 мкА
3,3 В	6 МГц	2,5 мА	0,7 мА	0,1 мкА
3,3 В	2 МГц	1,8 мА	0,4 мА	0,1 мкА

Для перевода микроконтроллера семейства AVR в один из режимов пониженного потребления энергии необходимо разряд SE (**S**leep **E**nable) регистра управления MCUCR установить в лог. 1, а затем выполнить команду `sleep`. Благодаря связи команды `sleep` с разрядом SE, центральный процессор не может быть непреднамеренно переведен в режим пониженного энергопотребления.

Когда во время режима пониженного энергопотребления происходит прерывание, центральный процессор выходит из "спящего" режима, выполняет подпрограмму обработки прерывания и продолжает выполнение программы с команды, следующей после команды `sleep`. Если во время режима пониженного энергопотребления поступает сигнал сброса, то центральный процессор выходит из "спящего" режима и продолжает выполнение программы с команды, расположенной по адресу $000 области команд.

Содержимое рабочих регистров и регистров ввода/вывода в течение режима пониженного энергопотребления остается неизменным. В том случае, если режим пониженного энергопотребления должен быть отключен в результате прерывания по уровню сигнала, низкий уровень должен сохраняться дольше, чем 16 мс, которые необходимы для перехода осциллятора в рабочий режим. В противном случае, соответствующий флаг прерываний снова должен быть сброшен прежде, чем центральный процессор возобновит работу.

Для микроконтроллеров семейства AVR может быть выбран один из двух "спящих" режимов.

- В **ждущем режиме** (Idle Mode) работа центрального процессора приостанавливается, но таймер/счетчик, сторожевой таймер, система прерываний и тактирования остаются активными. Благодаря этому, центральный процессор может быть возвращен в обычный режим работы с помощью сторожевого таймера, таймера/счетчика или внешнего прерывания. Если нет необходимости в том, чтобы выход из ждущего режима осуществлялся с помощью аналогового компаратора, то компаратор отключается посредством

установки в лог. 1 разряда ACD регистра ACSR. Это еще более снижает потребление тока в ждущем режиме, как это показано в табл. 3.9.

- В **режиме пониженного энергопотребления** (Power Down Mode) системный осциллятор (а значит и весь микроконтроллер) находится в отключенном состоянии. В таком режиме с помощью внутреннего RC-генератора колебаний может включаться лишь сторожевой таймер со своим собственным обеспечением тактовой частотой. Для этого используется разряд WDE регистра WDTCR. Активный сторожевой таймер по истечении времени задержки опять переводит микроконтроллер в нормальное состояние. Если сторожевой таймер также отключен, то в нормальное состояние его может перевести только сигнал сброса на выводе /RESET или внешнее прерывание, активизированное по достижению определенного уровня сигнала.
  В табл. 3.10 показано потребление тока центральным процессором модели AT90S1200 в режиме пониженного энергопотребления при активном и отключенном сторожевом таймере.

Таблица 3.9. Типичное потребление тока I_{CC} процессором модели AT90S1200 в ждущем режиме при активном и отключенном аналоговом компараторе

V_{CC}	Ф	I_{CC} / Активный	I_{CC} / Ждущий Аналоговый компаратор включен	I_{CC} / Ждущий Аналоговый компаратор отключен
5 В	12 МГц	9 мА	2,9 мА	2,4 мА
5 В	2 МГц	4 мА	1,5 мА	1 мА
3,3 В	6 МГц	2,5 мА	0,9 мА	0,7 мА
3,3 В	2 МГц	1,8 мА	0,5 мА	0,4 мА

Таблица 3.10. Типичное потребление тока I_{CC} центральным процессором модели AT90S1200 в режиме пониженного энергопотребления при активном и отключенном сторожевом таймере

V_{CC}	I_{CC} / Пониженное потребление Сторожевой таймер включен	ICC / Пониженное потребление Сторожевой таймер отключен
5 В	80 мкА	0,75 мкА
3,3 В	18 мкА	0,1 мкА

Выбор одного из "спящих" режимов осуществляется с помощью разряда SM регистра MCUCR. Если разряд SM установлен в лог. 1, то микроконтроллер переводится в режим пониженного энергопотребления последующей командой `sleep`, если же разряд SM сброшен в лог. 0, то последующей командой `sleep` микроконтроллер переводится в ждущий режим в том случае, если ранее в регистре MCUCR был установлен разряд SE.

4 Таймеры/счетчики микроконтроллеров базовой серии семейства AVR

Микроконтроллеры базовой серии семейства AVR, вплоть до модели AT90S1200, располагают двумя таймерами/счетчиками: 8-разрядным таймером/счетчиком T/C0 и 16-разрядным таймером/счетчиком T/C1. В микроконтроллере AT90S1200 присутствует только таймер/счетчик T/C0.

Таймеры/счетчики T/C0 и T/C1 выполнены в виде суммирующих счетчиков, когда каждый нарастающий фронт тактового импульса увеличивает содержимое счетчика на единицу. Таймеры/счетчики T/C0 и T/C1 микроконтроллеров AVR базовой серии могут применяться по выбору как задатчики времени (таймеры) или как счетчики. В случае использования в качестве таймера, их тактовая частота является производной величиной от такта системной синхронизации **Ф** внутреннего кварцевого осциллятора. При этом как T/C0, так и T/C1 использует в качестве тактового сигнала (напрямую или через делитель частоты) разделенный такт системной синхронизации **Ф**. Коэффициент деления делителя частоты может настраиваться индивидуально для каждого из двух таймеров с помощью мультиплексора.

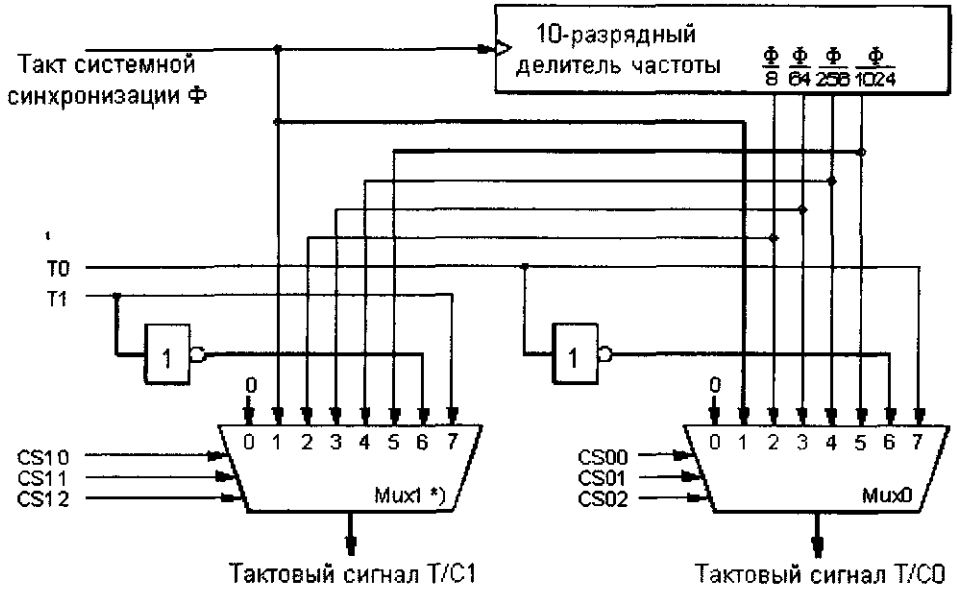
Если T/C0 функционирует в качестве счетчика, то он подсчитывает число импульсов T0, поступающих на контакт ввода/вывода, а T/C1 — число внешних импульсов T1, поступающих на этот же контакт. В этом случае соответствующий вывод должен быть сконфигурирован в инициализационной части программы как вход посредством записи лог. 0 в соответствующий разряд регистра направления передачи данных. В табл. 4.1 показано распределение портов между T0 и T1 для четырех представителей базовой серии семейства AVR.

Таблица 4.1. Занятость портов для импульсов T0 и T1

Внешний тактовый вход	*AT90S1200*	*AT90S2313*	*AT90S4414*	*AT90S8515*
T0	Порт D / Разряд 4	Порт D / Разряд 4	Порт B /Разряд 0	Порт B /Разряд 0
T1	—	Порт D /Разряд 5	Порт B /Разряд 1	Порт B /Разряд 1

Предварительный делитель частоты и схема управления таймером

На рис. 4.1 показана схема управления таймером в микроконтроллерах базовой серии семейства AVR. В режиме работы "Таймер" тактовые частоты для таймеров/счетчиков T/C0 и T/C1 могут быть настроены для каждого отдельно с помощью предварительного делителя частоты и мультиплексоров. Для этой цели в микроконтроллер интегрирован 10-ступенчатый делитель $Q_0...Q_9$, выводы которого Q_2 (**Ф**/8), Q_5 (**Ф**/64), Q_7 (**Ф**/256) и Q_9 (**Ф**/1024), наряду с тактом системной синхронизации **Ф**, могут быть подключены напрямую через соответствующий мультиплексор Mux0 и, равным образом, Mux1 к тактовому входу таймеров/счетчиков T/C0 и, соответственно, T/C1 (в модели AT90S1200 присутствует только Mux0 для таймера/счетчика T/C0).



*) Mux1 в модели AT90S1200 отсутствует

Рис. 4.1. Схема управления для подачи тактовых сигналов на таймер/счетчик микроконтроллеров базовой серии семейства AVR

Далее, если требуется режим работы "Счетчик", в качестве активного фронта с помощью мультиплексора может быть выбран ниспадающий или нарастающий фронт внешнего импульса на входах T0 и T1. При таком режиме работы внешний сигнал T0 (как и T1) синхронизируется с тактом системной синхронизации **Ф** внутреннего кварцевого осциллятора. Для этого внешний сигнал проверяется в течение каждого нарастающего фронта **Ф**.

Для обеспечения надежного распознавания время между двумя изменениями внешнего сигнала на входах T0 и T1 должно быть, как минимум, таким же как период следования тактов системной синхронизации **Ф** (рис. 4.2).

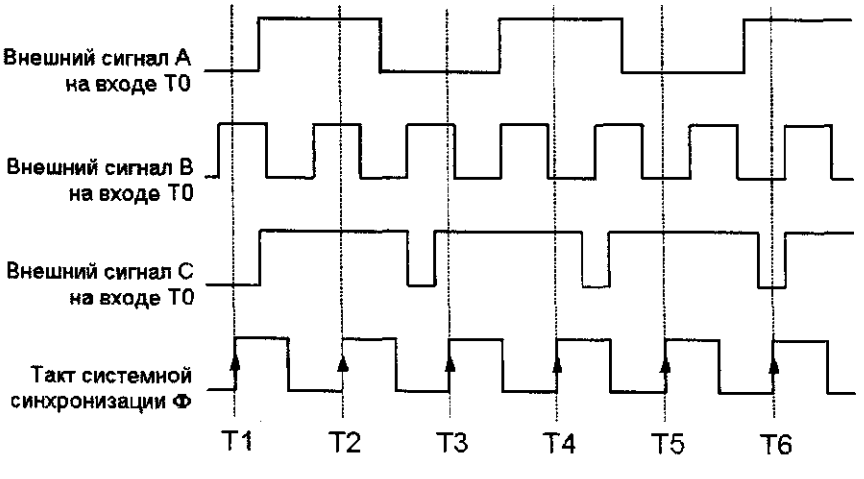


Рис. 4.2. Синхронизация внешнего сигнала на входе T0 с тактом системной синхронизации **Ф**

В случае внешнего сигнала A, показанного на рис. 4.2, длительность как высокого, так и низкого уровня больше, чем период следования тактов системной синхронизации **Ф**. В результате, при каждом опросе четко прослеживается изменение внешнего сигнала. В случае сигнала B, длительность как высокого, так и низкого уровня меньше, чем период следования тактов системной синхронизации **Ф**. В результате, отдельные фронты импульсов между опросами теряются и в счете не участвуют. Так, в точках опроса T1, T2 и T3, следующих одна за другой, всегда распознается лог. 1, а в точках T4, T5 и T6 — всегда лог. 0. Изменения сигналов между точками опроса при счете не учитываются!

В случае сигнала C, условие синхронизации выполняется для высокого уровня, однако продолжительность низкого уровня меньше, чем период следования тактов системной синхронизации **Ф**. В результате, некоторые изменения сигнала также остаются нераспознанными.

Если в таймере/счетчике нет необходимости, то имеется возможность через мультиплексный адрес 0b000 установить на его тактовом входе лог. 0 и таким образом остановить счет. В табл. 4.2 перечислены различные комбинации управляющих сигналов для мультиплексора Mux0 [Mux1] для выбора источника тактирования T/C0 [T/C1].

Таблица 4.2. Выбор входного такта для T/C0 и T/C1 через регистр TCCR0 и TCCR1B

CS02 [CS12]	*CS01 [CS11]*	*CS00 [CS10]*	*Описание [Значения в квадратных скобках относятся к T/C1]*
0	0	0	Задержка T/C0 [T/C1]
0	0	1	Режим "Таймер", такт счетчика = **Ф**
0	1	0	Режим "Таймер", такт счетчика = **Ф** / 8
0	1	1	Режим "Таймер", такт счетчика = **Ф** / 64
1	0	0	Режим "Таймер", такт счетчика = **Ф** / 256
1	0	1	Режим "Таймер", такт счетчика = **Ф** / 1024
1	1	0	Режим "Счетчик", такт счетчика = внешний импульс на выводе T0 [T1], активный ниспадающий фронт
1	1	1	Режим "Счетчик", такт счетчика = внешний импульс на выводе T0 [T1], активный нарастающий фронт

Восьмиразрядный таймер/счетчик T/C0

Восьмиразрядный таймер/счетчик T/C0 присутствует во всех микроконтроллерах базовой серии семейства AVR. Для T/C0 имеют значение только четыре регистра из области ввода/вывода (рис. 4.3).

Кроме, собственно, счетного регистра TCNT0, используются:

- регистр управления TCCR0 для настройки мультиплексора;
- регистр флагов прерываний TIFR с флагом переполнения T/C0 TOV0;
- уведомление о переполнении TCNT0 (переход из состояния $FF в $00);
- регистр TIMSK для разрешения/запрета прерываний T/C0 с помощью флага TOIE0.

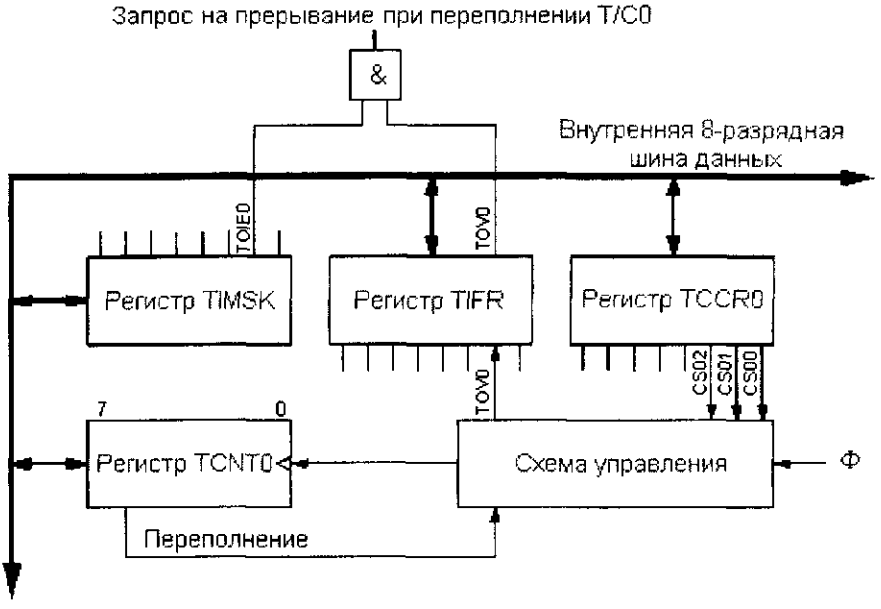


Рис. 4.3. Блок-схема таймера/счетчика T/C0 микроконтроллеров базовой серии семейства AVR

Как только с помощью разрядов CS00, CS01 и CS02 регистра управления TCCR0 (адрес $33 области ввода/вывода, адрес $53 в RAM) для мультиплексора Mux0 будет выбран адрес, отличный от 0b000, таймер/счетчик T/C0 по каждому импульсу, поступающему на тактовый вход, начинает увеличивать на единицу содержимое регистра TCNT0 (адрес $32 области ввода/вывода, адрес $52 в RAM).

Когда состояние счетчика в регистре TCNT0 изменяется с $FF на $00, в регистре TIFR (адресу $38 в области ввода/вывода) устанавливается флаг переполнения TOV0 таймера/счетчика T/C0. Для обработки этого переполнения у программиста есть две возможности.

- Постоянно опрашивать флаг TOV0 в цикле. Как только он установлен, требуемый промежуток времени истек, и выполнение программы может быть продолжено. Конечно, перед этим флаг TOV0 должен быть опять сброшен посредством записи лог. 1 в разряд 1 регистра TIFR.
- Когда будет установлен флаг I для глобального разрешения прерываний (разряд 7 в регистре состояния SREG по адресу $3F области ввода/вывода), а также флаг TOIE0 для разрешения прерываний от T/C0 (разряд 1 в регистре TIMSK по адресу $39 области ввода/вывода), то после установки флага TOV0 из-за перехода регистра TCNT0 из состояния $FF в состояние $00 будет всегда возникать прерывание T/C0. В этом случае флаг TOV0 аппаратно автоматически сбрасывается, если будет выполнен переход по адресу прерывания T/C0. Необходимо учитывать, что время от возникновения условия прерывания TOV0 до выполнения первой команды подпрограммы обработки прерывания (как и при любом другом прерывании), составляет минимум четыре тактовых цикла (сохранение адреса возврата в стеке и относительный переход к подпрограмме обработки прерывания). Если во время посту-

пления запроса на прерывание выполняется какая-либо команда, для которой требуется более одного тактового цикла, то она в любом случае будет завершена. Это время также должно учитываться при определении начального значения регистра TCNT0.

Регистр управления TCCR0 таймера/счетчика T/C0

Регистр управления TCCR0 таймера/счетчика T/C0 расположен в области ввода/вывода по адресу $33 (адрес $53 в RAM). Он инициализируется значением $00 после поступления сигнала сброса, в результате чего счетчик останавливается.

В микроконтроллерах базовой серии семейства AVR используются только разряды 0–2 регистра TCCR0 (доступны для чтения и записи). С их помощью посредством мультиплексора Mux0 осуществляется выбор такта для T/C0. Разряды 3–7 зарезервированы компанией Atmel, поэтому их можно только считывать (всегда содержат лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$33 ($53)	–	–	–	–	–	CS02	CS01	CS00	TCCR0

Счетный регистр TCNT0 таймера/счетчика T/C0

Счетный регистр TCNT0 таймера/счетчика T/C0 находится в области ввода/вывода по адресу $32 (адресу $52 в RAM). Он доступен для чтения и записи, и после поступления сигнала сброса инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$32 ($52)	MSB							LSB	TCNT0

Регистр TCNT0 — это, собственно говоря, и есть счетчик. До тех пор, пока на управляющих входах CS00...CS02 мультиплексора Mux0 не появится адрес 0b000, содержимое счетчика с каждым тактовым импульсом увеличивается на единицу.

Области применения таймера/счетчика T/C0

Таймер/счетчик T/C0 очень хорошо подходит для оценки временных интервалов. Для этого в ходе выполнения программы в регистр TCNT0 записывается исходное значение. Затем может быть запущен таймер/счетчик T/C0 с требуемым тактом, выбранного с помощью мультиплексора Mux0. Программа или ожидает в цикле появления флага переполнения TOV0 в регистре TIFR (режим опроса), или разрешает прерывание таймера/счетчика T/C0. Флаг TOV0 указывает на то, что требуемое время задержки истекло, и выполнение программы может быть продолжено. Это показывают примеры в соответствующих разделах главы 16, имеющие отношение к микроконтроллеру AT90S1200.

Шестнадцатиразрядный таймер/счетчик T/C1

Шестнадцатиразрядный таймер-счетчик T/C1 отсутствует только в модели AT90S1200 базовой серии микроконтроллеров AVR. К нему имеют отношение в

общей сложности восемь регистров в области ввода/вывода (рис. 4.4). В микроконтроллерах AT90S4414 и AT90S8515 используется таймер/счетчик T/C1 в полной комплектации. В микроконтроллере AT90S2313 отказались от применения регистра сравнения B и компонентов, помеченных на рис. 4.4 символом "*".

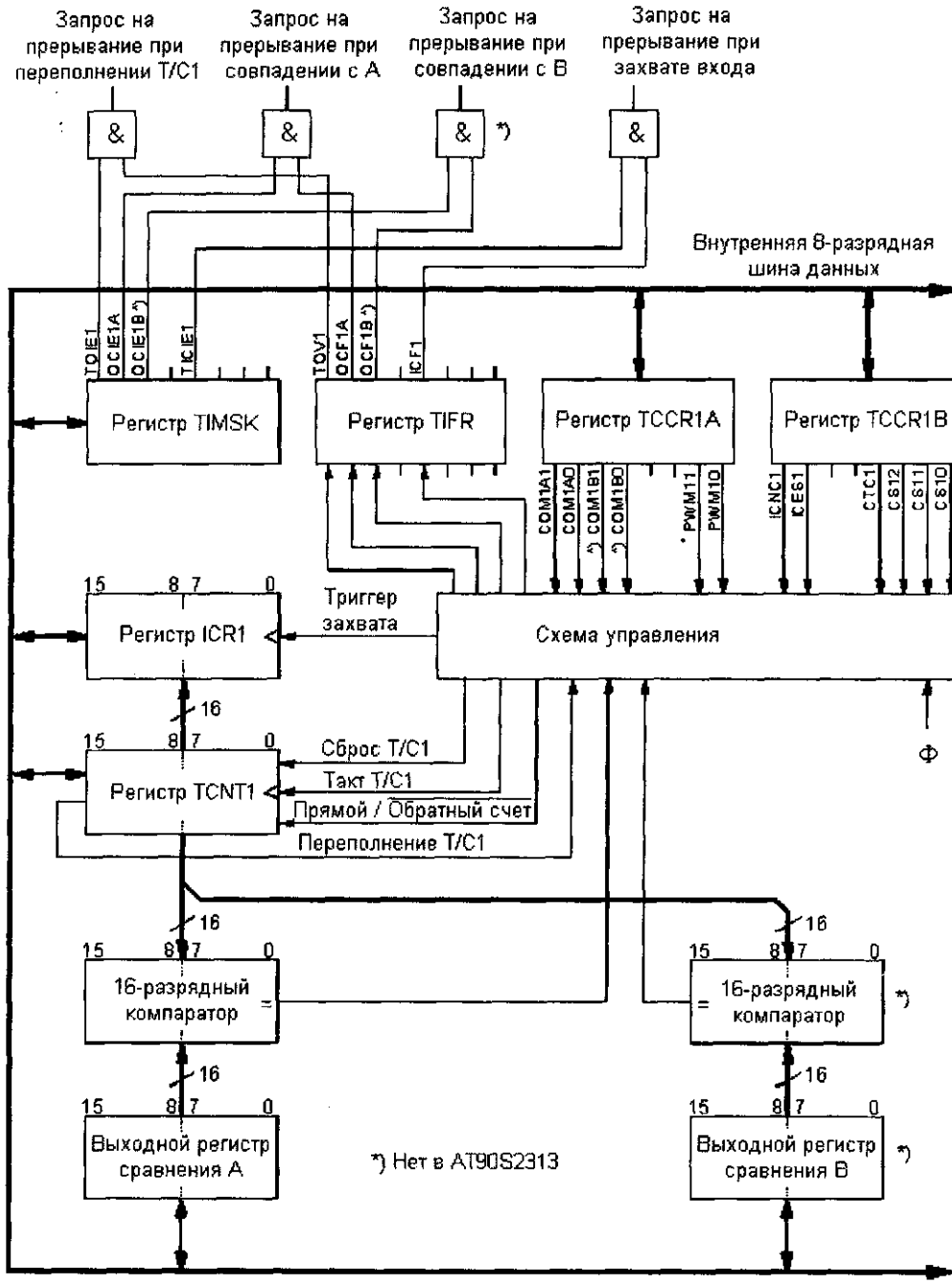


Рис. 4.4. Блок-схема 16-разрядного таймера/счетчика T/C1:

Назначение отдельных регистров на рис. 4.4:

- **TCNT1** — собственно, счетный регистр;
- **TCCR1A** — регистр управления для определения реакции выводов OC1A/OC1B в случае совпадения состояния счетчика в регистре TCNT1 с регистрами сравнения OCR1A/OCR1B, а также для выбора ШИМ;
- **TCCR1B** — регистр управления для настройки мультиплексора Mux1 (см. рис. 4.1), для разрешения подачи сигнала сброса для регистра TCNT1 и для настройки триггера захвата;
- **TIMSK** — содержит разряд TOIE1 для разрешения прерываний по переполнению таймера/счетчика T/C1; разряды OCE1A и OCE1B для разрешения прерываний A и B по сравнению; а также разряд TICIE1, разрешающий прерывание по захвату;
- **TIFR** — содержит флаг TOV1 переполнения таймера/счетчика T/C1, указывающий на переполнение счетчика в регистре TCNT1 при появлении значения $0000; флаги OCF1A и OCF1B, указывающие на совпадение состояния счетчика и регистров сравнения A/B; флаг ICF1, который показывает перенос (захват) состояния счетчика в регистре ICR1;
- **ICR1** — регистр захвата входа (при появлении на выводе ICP фронта входного сигнала, определенного как активный, текущее состояние счетчика будет перенесено в этот регистр);
- **OCR1A, OCR1B** — регистры сравнения; их содержимое постоянно сравнивается с состоянием счетчика. В случае совпадения выполняются действия, определенные регистром TCCR1A.

С помощью таймера/счетчика T/C1 могут быть решены гораздо более сложные задачи, чем в случае описанного ранее таймера/счетчика T/C0. Здесь, кроме регистра захвата, в распоряжение пользователя также предоставляется один (в модели AT90S2313) или два (микроконтроллеры AT90S4414 и AT90S8515) выходных регистра сравнения (A и B).

Содержимое выходного регистра сравнения постоянно аппаратно сравнивается с содержимым счетчика, и при совпадении активизируются заранее определенные действия для соответствующего вывода. Кроме того, существует возможность использовать регистр сравнения A для сброса счетчика при достижении им некоторого требуемого состояния.

С помощью регистра сравнения T/C1 можно вырабатывать два выходных сигнала, модулированных по ширине импульса. При этом может быть запрограммирована разрешающая способность 8, 9 или 10 разрядов.

С помощью функции захвата на входе T/C1 посредством запускающего внешнего сигнала на выводе ICP текущее состояние счетчика T/C1 может быть "заморожено" в регистре ICR1. Альтернативно, ко входу ICP в качестве источника запуска для регистра захвата может быть также подсоединен аналоговый компаратор. Более подробно об этом сказано в главе 9, "Интегрированный аналоговый компаратор".

16-разрядный таймер/счетчик T/C1 в нормальном случае работает как суммирующий счетчик. Единственное исключение — его применение в качестве широт-

но-импульсного модулятора, когда он работает как обратный и суммирующий счетчик.

Как только с помощью разрядов CS10, CS11 и CS12 в регистре управления TCCR1B (адрес $2E области ввода/вывода) для мультиплексора Mux1 будет установлен адрес, отличный от 0b000, таймер/счетчик T/C1 при поступлении на его вход тактовых импульсов начинает увеличивать на единицу содержимое 16-разрядного регистра TCNT1.

Когда состояние счетчика в регистре TCNT1 меняется с $FFFF на $0000, в регистре TIFR (адресу $38 области ввода/вывода) устанавливается флаг TOV1 переполнения T/C1. Для обработки переполнения счетчика у программиста есть две возможности.

- Постоянный опрос флага переполнения TOV1 в цикле. Как только флаг переполнения TOV1 установлен, требуемый промежуток времени истек, и выполнение программы может быть продолжено. Конечно, перед этим флаг TOV1 должен быть снова сброшен посредством записи лог. 1.
- Когда будет установлен флаг I для глобального разрешения прерываний (разряд 7 в регистре состояния SREG по адресу $3F области ввода/вывода), а также флаг TOIE1 для разрешения прерываний T/C1 (разряд 1 в регистре TIMSK по адресу $39 области ввода/вывода), то в результате перехода счетного регистра TCNT1 из состояния $FFFF в состояние $0000 всегда будет происходить прерывание T/C1 из-за установки флага TOV1. В этом случае флаг TOV1 будет аппаратно автоматически возвращен в исходное состояние, если будет выполнен переход по адресу прерывания T/C1. Необходимо принимать во внимание, что период от возникновения условия прерывания TOV1 до выполнения первой команды подпрограммы обработки прерывания (как и для всех прерываний) составляет минимум четыре тактовых цикла (сохранение адреса возврата в стеке и относительный переход к подпрограмме обработки прерывания). Если в момент поступления запроса на прерывание выполняется команда, для которой требуется более одного тактового цикла, то в любом случае она будет выполнена до конца. Это время также должно учитываться при определении начального значения для регистра TCNT1.

Счетный регистр TCNT1

16-разрядный счетный регистр TCNT1 таймера/счетчика T/C1 разделен на два байта. Младший байт расположен в области ввода/вывода по адресу $2C (адрес $4C в RAM), а старший байт — в области ввода/вывода по адресу $2D (адрес $4D в RAM). Оба байта доступны для чтения и записи, и после поступления сигнала сброса инициализируются значением $00.

Разряд	15	14	13	12	11	10	9	8	
$2D ($4D)	MSB								TCNT1H
$2C ($4C)								LSB	TCNT1L
Разряд	7	6	5	4	3	2	1	0	

Счетный регистр TCNT1 — это, собственно, и есть счетчик. До тех пор, пока на мультиплексоре Mux1 через управляющие входы CS10...CS12 не будет выбран адрес 0b000, содержимое счетчика с каждым тактовым импульсом увеличивается на единицу.

Для того чтобы, несмотря на внутреннюю 8-разрядную шину данных, оба байта одновременно могли быть загружены в 16-разрядный счетчик, для доступа к регистру TCNT1 используется внутренний регистр под названием TEMP.

- **Процесс записи в регистр TCNT1**. При инициализации регистра TCNT1 сначала необходимо записать старший байт по адресу $2D области ввода/вывода. Однако этот байт попадает в регистр TCNT1H не сразу же — сначала он внутренне сохраняется в промежуточном регистре TEMP. После этого программа пользователя записывает младший байт по адресу $2C области ввода/вывода. Этот байт внутренне объединяется с байтом, сохраненным в буферном регистре, и оба байта одновременно записываются в 16-разрядный регистр.
- **Процесс чтения регистра TCNT1**. При чтении TCNT1 сначала должен быть считан младший байт по адресу $2C области ввода/вывода. При обращении к этому байту с целью чтения во вспомогательный регистр TEMP автоматически записывается также и текущий старший байт для промежуточного хранения данных о состоянии счетчика. При последующем доступе к регистру TCNT1H с целью чтения возвращается и байт, записанный во вспомогательный регистр TEMP для промежуточного хранения.

На момент издания этой книги был актуален выпуск AVR-Studio 2.0. При тестировании программ для таймера/счетчика T/C1 возникла следующая проблема. В условиях практического применения (как было описано выше) в микроконтроллерах семейства AVR во время одновременной записи (или чтения) значений в 16-разрядный регистр с помощью вспомогательного регистра TEMP непосредственно после выполнения команды `out OCR1AH, r16`, а также перед записью младшего байта AVR-Studio ошибочно отображает в окне **Timer1** это значение после записи старшего байта.

Регистр управления TCCR1A

Регистр управления TCCR1A таймера/счетчика T/C1 находится в области ввода/вывода по адресу $2F (адрес $4F в RAM). После поступления сигнала сброса он инициализируется значением $00.

В моделях AT90S8515 и AT90S4414 используются только разряды 0–1 и 4–7 регистра TCCR1A (доступны для чтения и записи). Разряды 2 и 3 зарезервированы компанией Atmel и доступны только для чтения (всегда содержат лог. 0).

В микроконтроллере AT90S2313 используются только разряды 0, 1, 6 и 7 регистра TCCR1A, поскольку в этой модели нет компаратора B. Эти разряды доступны для чтения и записи. Разряды 2–5 зарезервированы компанией Atmel и доступны только для чтения (всегда содержат лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$2F ($4F)	COM1A1	COM1A0	COM1B1 *)	COM1B0 *)	–	–	PWM11	PWM10	TCCR1A

*) Отсутствует в модели AT90S2313

Разряды **COM1A1** и **COM1A0** для компаратора A и, равным образом, разряды **COM1B1** и **COM1B0** для компаратора B определяют действия на выводе OC1A, подчиненного компаратору A (и, соответственно, — на выводе OC1B, подчиненного компаратору B) при совпадении содержимого регистра сравнения с содержимым счетчика. В случае вывода OC1A речь идет о разряде 5 порта D (AT90S8515 и AT90S4414) и, соответственно, разряда 3 порта B (AT90S2313). Для того чтобы можно было применить этот вывод в качестве выхода для функции сравнения, он должен быть соответствующим образом сконфигурирован посредством записи лог. 1 в регистре направления передачи данных. Возможные настройки для режима сравнения Compare1 показаны в табл. 4.3.

Таблица 4.3. Возможные варианты для работы в режиме сравнения Compare1

COM1A1 [COM1B1]	*COM1A0 [COM1B0]*	*Действия в случае совпадения [Значения в квадратных скобках — для компаратора B]*
0	0	OC1A [OC1B] с таймером/счетчиком T/C1 не связан
0	1	OC1A [OC1B] переключен в другое состояние
1	0	OC1A [OC1B] установлен в лог. 0
1	1	OC1A [OC1B] установлен в лог. 1

В случае активизации режима ШИМ, разряды 4–7 в регистре TCCR1A имеют значения, отличные от указанных в табл. 4.3. Более подробно это освещено в разделе "Широтно-импульсная модуляция".

Разряды **PWM11** и **PWM10** активизируют режим ШИМ и устанавливают разрешающую способность для ШИМ (табл. 4.4).

Таблица 4.4. Возможные варианты для работы в режиме ШИМ

PWM11	*PWM10*	*Описание*
0	0	Режим ШИМ не активен
0	1	T/C1 работает как 8-разрядный широтно-импульсный модулятор
1	0	T/C1 работает как 9-разрядный широтно-импульсный модулятор
1	1	T/C1 работает как 10-разрядный широтно-импульсный модулятор

Регистр управления TCCR1B

Регистр управления TCCR1B таймера/счетчика T/C1 расположен по адресу $2E области ввода/вывода (адрес $4E в RAM). После поступления сигнала сброса он инициализируется значением $00.

В микроконтроллеров базовой серии семейства AVR используются только разряды 0–3 и 6, 7 регистра TCCR1B (доступны для чтения и записи). Разряды 4 и 5 зарезервированы компанией Atmel, и доступны только для чтения (всегда содержат лог.0).

Разряд	7	6	5	4	3	2	1	0	
$2E ($4E)	ICNC1	ICES1	–	–	CTC1	CS12	CS11	CS10	TCCR1B

Как уже было подробно описано в разделе "Предварительный делитель частоты и схема управления таймером", разряды **CS10–CS12** используются для выбора тактовой частоты $f_{T/C1}$ таймера/счетчика T/C1.

Если установлен разряд **CTC1**, то таймер/счетчик T/C1 возвращается в состояние $0000 по импульсу такта системной синхронизации, следующего после совпадения содержимого счетчика и регистра сравнения А. При работе в режиме ШИМ этот разряд на процесс работы никак не влияет.

Когда происходит сброс T/C1 через регистр сравнения OCR1A или непосредственно после записи в регистр TCNT1 значения $0000, то совпадение $0000 с OCR1B не распознается! Однако, когда значение $0000 достигается посредством обычного перехода счетчика из состояния $FFFF в состояние $0000, совпадение будет распознано как корректное. Для любого значения, не равного $0000, совпадение содержимого регистра сравнения OCR1B с состоянием счетчика всегда распознается корректно.

На момент издания этой книги была актуальна версия AVR-Studio 2.0. При тестировании программ для таймера/счетчика T/C1 в ней возникла следующая проблема. В случае сравнения с содержимым регистра OCR1A, сброс таймера/счетчика происходит по тактовому импульсу системной синхронизации, который следует после успешного сравнения (рис. 4.5). При моделировании процесса выполнения программы с помощью AVR-Studio сброс ошибочно отображается в течение того же тактового импульса.

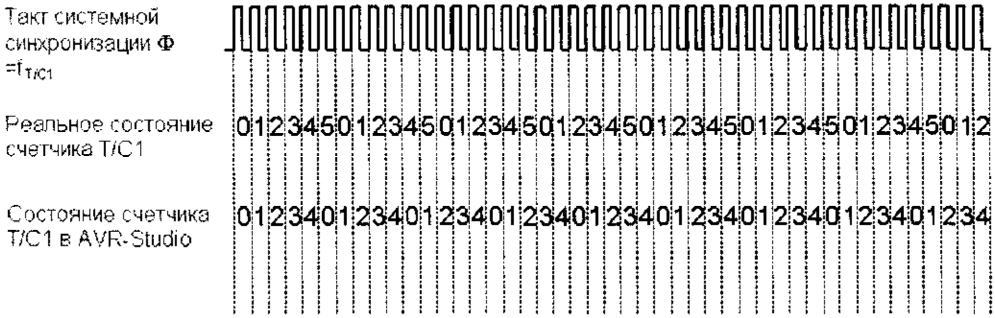


Рис. 4.5. Состояние счетчика T/C1 реальное и в AVR-Studio 2.0 (неправильно!), когда регистр OCR1A получает значение $0005 и не было выполнено предварительное деление входного такта

В реальных микроконтроллерах AT90S8515 для одного периода такта системной синхронизации наступает состояние счетчика, которое соответствует содержимому регистра сравнения. С помощью следующего тактового импульса производится сброс счетчика в $0000. Итак, если период счетчика должен состоять из "*n*" периодов тактовой частоты $f_{T/C1}$, то значение сравнения в регистре OCR1A должно быть *n*–1, когда для T/C1 применяется такт системной синхронизации без предварительного деления (см. рис. 4.5), и "*n*", когда для T/C1 применяется такт с предварительным делением (рис. 4.6).

При тестировании программ для микроконтроллеров AT90S8515 и AT90S4414 в AVR-Studio вывод ICP моделируется через PD4 (разряд 4 порта D), а вывод OC1B — PD6 (разряд 6 порта D). Порты должны быть соответствующим образом установлены в регистре направления передачи данных.

С помощью разряда **ICES1** регистра TCCR1B посредством двойного мультиплексора Mux2 (см. рис. 4.7) определяется, каким образом должна осуществляться передача состояния счетчика в регистр захвата на входе: нарастающим фронтом и, соответственно, следующими друг за другом высокими уровнями входного сигнала (ICES1=1) или ниспадающим фронтом и, соответственно, четырьмя последовательными входными сигналами низкого уровня (ICES1=0).

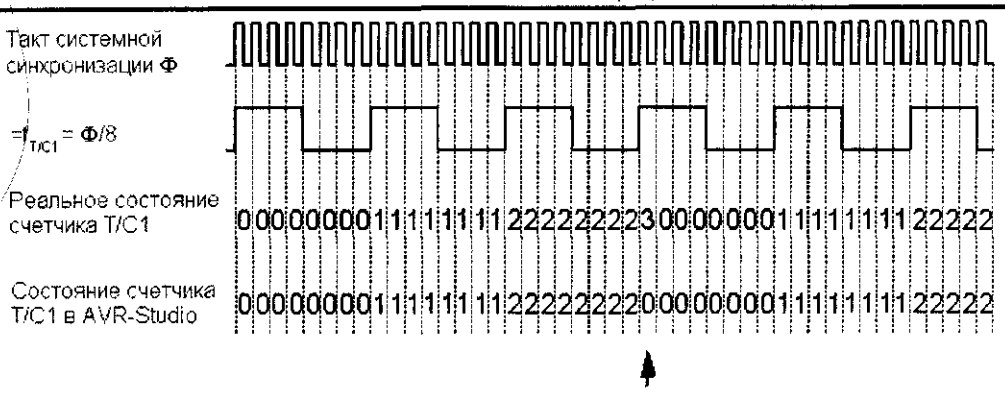


Рис. 4.6. Состояния счетчика T/C1 реальное и в AVR-Studio 2.0 (неправильно!), когда регистр OCR1A получает значение $0003 а входной такт предварительно разделен на 8

Разряд **ICNC1** (Input Capture Noise Canceller) определяет посредством мультиплексора Mux3 (см. рис. 4.7), должно ли быть активизировано подавление помех. Если ICNC1 = лог. 0, то подавление помех отключено, и содержимое счетчика переносится в регистр ICR1 по следующему фронту на выходе X мультиплексора Mux1 (см. рис. 4.7), определенного как активный с помощью разряда ICES1.

Если ICNC1 = лог. 1, то подавление помех для функции захвата на входе активна. Для подавления кратковременных импульсов помех, которые могут привести к ошибочному запуску, входной сигнал зондируется на протяжении четырех периодов такта системной синхронизации. Только после того как будут распознаны четыре последовательных низких или высоких уровней входного сигнала, что определяется разрядом ICES1, при активном подавлении помех будет запущена запись текущего состояния счетчика в регистр ICR1. Схема подавления помех и выбора фронтов для функции захвата на входе схематически показана на рис. 4.7.

С помощью разряда ACIC в регистре управления и состояния ACSR аналогового компаратора посредством мультиплексора Mux1 (см. рис. 4.7) определяется, должен ли входной вывод ICP или аналоговый компаратор работать в качестве источника для функции захвата на входе. Выходной сигнал X мультиплексора Mux1 опрашивается одним из регистров смещения, состоящего из четырех D-триггеров. Выход Y вентиля "НЕ-ИЛИ" переходит в состояние лог. 1, когда на протяжении четырех периодов системного такта на входе распознается лог. 0. Аналогично, выход Z вентиля "И" переходит в состояние лог. 1, когда на протяжении четырех периодов системного такта на входе распознается лог. 1. Принцип подавления нескольких импульсов помех на основании сигнала X на выходе мультиплексора Mux1 показан на рис. 4.8.

Выход Y вентиля "НЕ-ИЛИ" в таком случае переходит в состояние лог. 1 только тогда, когда распознаются четыре последовательных уровня лог. 0 сигнала X. Тем самым выполняется условие захвата, поскольку выполняется равенство ICNC1 = лог. 1 и ICES1 = лог. 0.

Выход Z вентиля "И" переходит в состояние лог. 1 только тогда, когда будут распознаны четыре последовательных уровня лог. 1 сигнала X. Тем самым выполняется условие захвата, поскольку выполняется равенство ICNC1 = лог. 1 и ICES1 = лог. 0.

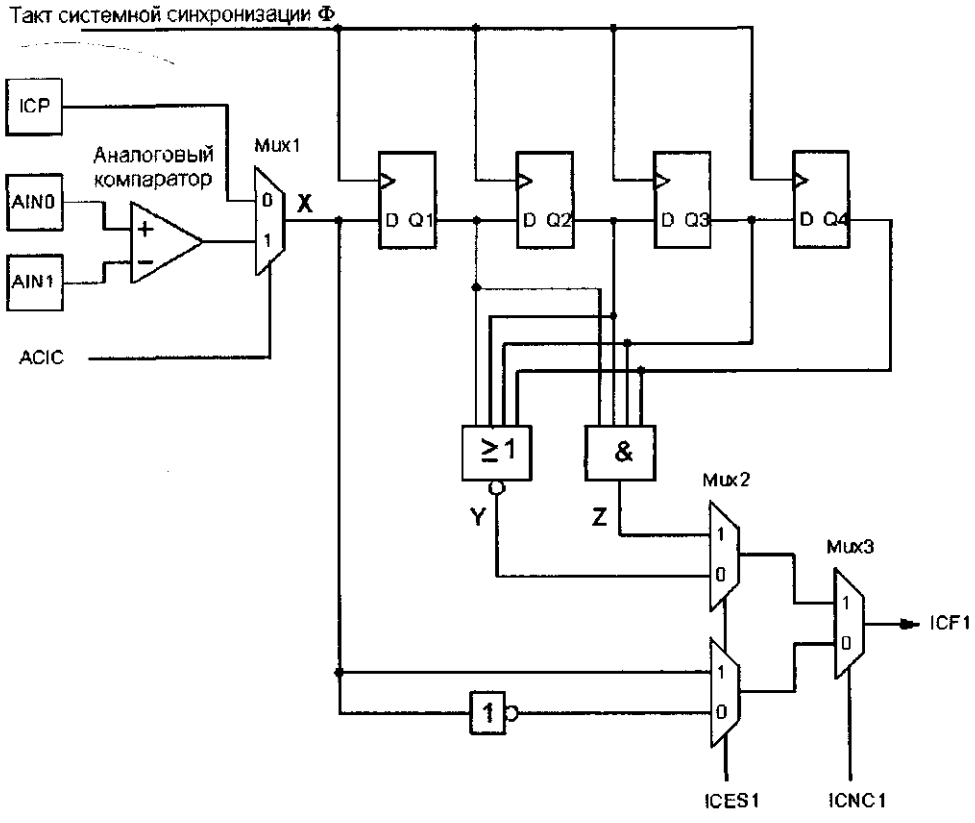


Рис. 4.7. Схема подавления помех и выбора фронта сигнала для функции захвата по входу T/C1

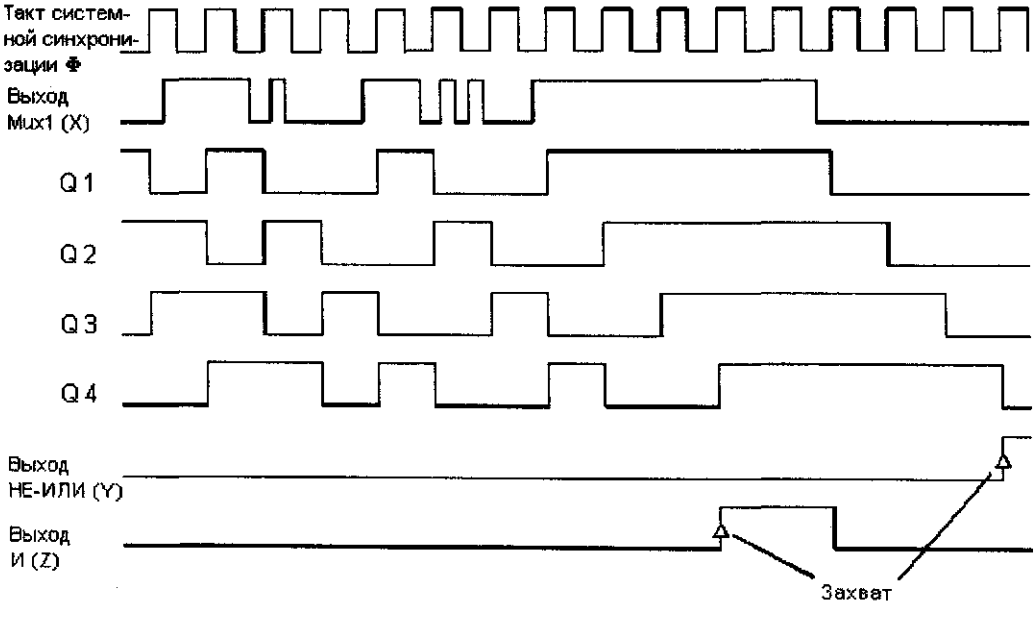


Рис. 4.8. Принцип действия схемы подавления помех таймера/счетчика T/C1

Регистр захвата по входу ICR1

16-разрядный регистр захвата по входу ICR1 разделен на два байта: ICR1L и ICR1H. Младший байт расположен в области ввода/вывода по адресу $24 (или по адресу $44 в RAM). Старший байт расположен в области ввода/вывода по адресу $25 (или по адресу $45 в RAM). Оба байта доступны только для чтения и после поступления сигнала сброса инициализируются значением $00.

Разряд	15	14	13	12	11	10	9	8	
$25 ($45)	MSB								ICR1H
$24 ($44)								LSB	ICR1L
Разряд	7	6	5	4	3	2	1	0	

Как только на выходе X мультиплексора Mux1 (см. рис. 4.3.4) будет обнаружен нарастающий или ниспадающий фронт сигнала (в зависимости от разряда ICES1 регистра TCCR1B), то в регистр ICR1 будет перенесено текущее значение регистра TCNT1. Одновременно с этим, для индикации передачи в регистре TIFR будет установлен флаг захвата на входе ICF1.

Для того чтобы, несмотря на внутреннюю 8-разрядную шину, можно было считать одновременно оба байта 16-разрядного регистра ICR1, для доступа к нему используется внутренний регистр TEMP.

При чтении регистра ICR1 сначала необходимо считать младший байт ICR1L по адресу $24 области ввода/вывода. При обращении к этому байту в промежуточную память автоматически вводится также и старший байт ICR1H из вспомогательного регистра TEMP. При последующем обращении к ICR1H байт, помещенный в буферную память вспомогательного регистра TEMP, возвращается обратно.

Регистры сравнения на выходе OCR1A и OCR1B

Оба 16-разрядных регистра сравнения на выходе OCR1A и OCR1B таймера/счетчика T/C1 разделены на два байта. Младший байт регистра OCR1A расположен в области ввода/вывода по адресу $2A (или по адресу $4A в RAM), а старший байт — в области ввода/вывода по адресу $2B (или по адресу $4B в RAM). Младший байт регистра OCR1B расположен в области ввода/вывода по адресу $28 (или по адресу $48 в RAM), а старший байт — в области ввода/вывода по адресу $29 (или по адресу $49 в RAM). Оба байта доступны только для чтения и после поступления сигнала сброса инициализируются значением $00.

Разряд	15	14	13	12	11	10	9	8	
$2B ($4B)	MSB								OCR1A H
$2A ($4A)								LSB	OCR1A L
Разряд	7	6	5	4	3	2	1	0	

Разряд	15	14	13	12	11	10	9	8	
$29 ($49)	MSB								OCR1BH
$28 ($48)								LSB	OCR1B L
Разряд	7	6	5	4	3	2	1	0	

Для того чтобы, несмотря на внутреннюю 8-разрядную шину, в 16-разрядный регистр OCR1 можно было загрузить одновременно оба байта, используется внутренний регистр TEMP.

Вначале в регистр OCR1AH/ OCR1BH должен быть записан старший байт, но не напрямую, а во вспомогательный регистр TEMP для промежуточного хранения. Вслед за этим пользовательская программа записывает в регистр OCR1AL/ OCR1BL младший байт, который внутренне объединяется с байтом в регистре TEMP, после чего оба байта параллельно записываются в 16-разрядный регистр.

Таймер/счетчик T/C1 как широтно-импульсный модулятор

Когда регистр управления TCCR1A определяет работу таймера/счетчика T/C1 в конфигурации широтно-импульсного модулятора, то в микроконтроллерах AT90S4414 и AT90S8515 регистры OCR1A и OCR1B, а также регистр TCNT1 образуют двухканальный, несинхронизированный, свободный от импульсных помех и синхронный по фазе широтно-импульсный модулятор с программируемой разрешающей способностью 8, 9 или 10 разрядов. Сигнал с ШИМ поступает как на вывод OC1A (вывод 15), так и на вывод OC1B (вывод 29). В микроконтроллере AT90S2313 с выводом OC1A (вывод 15) связан только один канал.

Таймер/счетчик T/C1 работает в этом режиме как суммирующий и вычитающий счетчик, осуществляя циклические переходы от $0000 к максимальному значению TOP, и затем снова возвращаясь к $0000. При запрограммированной разрешающей способности ШИМ в N разрядов значение TOP рассчитывается как

$$TOP = 2^N - 1.$$

Частота $f_{ШИМ}$, с которой повторяются циклы ШИМ, вычисляется по формуле:

$$f_{PWM} = f_{T/C1} / (2^{N+1} - 2),$$

причем частота таймера/счетчика $f_{T/C1}$ выбирается с помощью разрядов CS10–CS12 регистра TCCR1B, и разрешающая способность N — с помощью разрядов PWM10 и PWM11 регистра TCCR1A. Соответствующие взаимосвязи показаны в табл. 4.5.

Таблица 4.5. Значения TOP и частоты ШИМ в зависимости от разрешающей способности ШИМ

PWM11	*PWM10*	*Разрешающая способность*	*Значение TOP*	*Частота ШИМ*
0	1	8 разрядов	$00FF (255)	$f_{T/C1}$ / 510
1	0	9 разрядов	$01FF (511)	$f_{T/C1}$ / 1022
1	1	10 разрядов	$03FF (1023)	$f_{T/C1}$ / 2046

Когда состояние счетчика в регистре TCNT1 совпадает со значением 10 младших разрядов регистра OCR1A/OCR1B, то, в зависимости от состояния разрядов COM1A1/COM1A0 или COM1B1/Com1B0 регистра TCCR1A, вывод OC1A/OC1B последующим тактовым импульсом устанавливается или сбрасывается. Соответствующие взаимосвязи показаны в табл. 4.6.

В случае неинвертирующего широтно-импульсного модулятора, коэффициент заполнения g прямоугольного сигнала на выходном выводе с ШИМ соответствует значению n / (2N – 1), где n — значение в соответствующем регистре OCR, а N — разрешающая способность ШИМ в разрядах (рис. 4.9).

Таблица 4.6. Возможности выбора для режима сравнения Compare1 при работе ШИМ

COM1A1 [COM1B1]	*COM1A0 [COM1B0]*	*Действие в случае совпадения [Значения в квадратных скобках — для компаратора B]*
0	0	Регистр OC1A [OC1B] не связан с таймером/счетчиком T/C1
0	1	Регистр OC1A [OC1B] не связан с таймером/счетчиком T/C1
1	0	Неинвертирующий широтно-импульсный модулятор. В случае соответствия, при суммирующем подсчете на выводе OC1A [OC1B] устанавливается лог. 0, а при подсчете с вычитанием — лог. 1
1	1	Инвертирующий широтно-импульсный модулятор. В случае соответствия, при суммирующем подсчете на выводе OC1A [OC1B] устанавливается лог. 1, а при подсчете с вычитанием — лог. 0

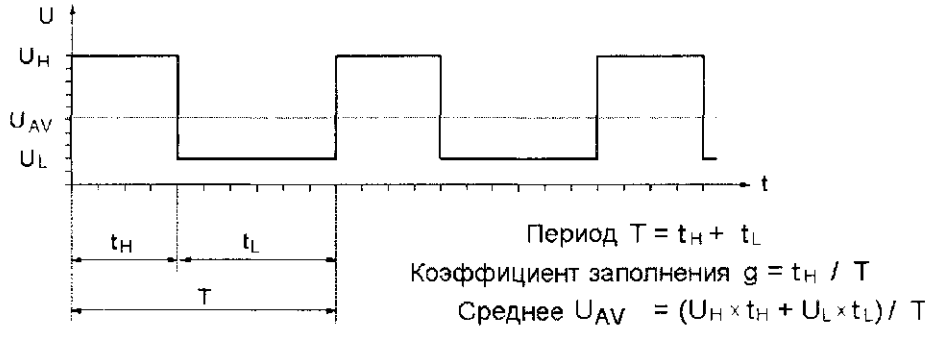


Рис. 4.9. Определение периода T, коэффициента заполнения g и среднего арифметического U_{AV} прямоугольных импульсов напряжения U

Если регистр сравнения OCR1A/OCR1B содержит значение TOP или 0, то на соответствующем выводе, в соответствии с правилами, представленными в табл. 4.7, постоянно поддерживаются уровень лог. 0 или лог. 1.

Таблица 4.7. Вывод ШИМ для особых случаев OCR1A[B] = TOP или OCR1A[B] = 0

COM1A1 [COM1B1]	*COM1A0 [COM1B0]*	*OCR1A [OCR1B]*	*Вывод OC1A [OC1B]*
1	0	0	0
1	0	TOP	1
1	1	0	1
1	1	TOP	0

На рис. 4.10. на примере фиктивной 3-хразрядной ШИМ показано формирование неинвертированного и инвертированного выходного ШИМ-сигнала для выхода OC1B.

На диаграмме A показан примерный вид ступенчатого сигнала, соответствующий состоянию счетчика TCNT1, на диаграмме B — неинвертированный, а на диаграмме C — инвертированный выходной сигнал.

Продолжительность периода T_{PWM} в этом случае вычисляется в соответствии с рассмотренным выше уравнением $T_{PWM} = T_{T/C1} \cdot (2^{N+1} - 2)$. Таким образом, при

N = 3 период ШИМ-сигнала состоит из 14 периодов тактового сигнала $f_{T/C1}$ на входе TCNT1.

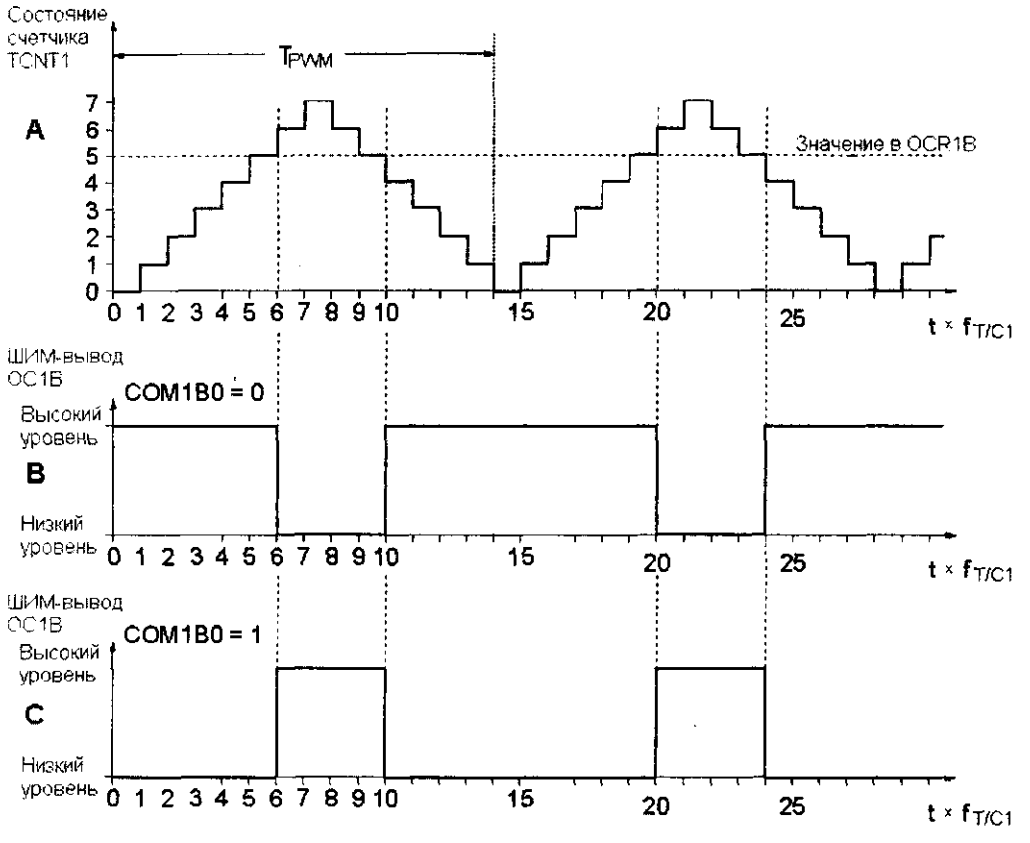


Рис. 4.10. Способ формирования неинвертированных и инвертированных выходных ШИМ-сигналов

В данном примере регистр сравнения OCR1B содержит значение 5. В регистре TCNT1, учитывая тот факт, что его исходное значение равно 0, значение 5 появляется после пяти тактовых импульсов. На следующем тактовом импульсе, после распознания совпадения, в соответствии с табл. 4.6 на выводе OC1B устанавливается уровень лог. 0 (см. рис. 4.10, B).

Регистр TCNT1 инкрементируется далее до тех пор, пока не будет достигнуто значение TOP, которое при трехразрядной ШИМ составляет 7. Как только достигнуто значение TOP, направление счета меняется на обратное, и регистр выполняет вычитание. После девятого тактового импульса, начиная от стартового значения 0, содержимое регистра TCNT1 опять совпадает с содержимым регистра OCR1B. На следующем тактовом импульсе на выходе OC1B, в соответствии с табл. 4.6, устанавливается уровень лог. 1.

Регистр TCNT1 декрементируется далее до тех пор, пока опять не будет достигнуто значение 0. Это происходит после в общей сложности четырнадцати тактовых импульсов, считая от начального значения 0. Таким образом завершается период ШИМ-сигнала, направление счета вновь меняется на обратное и регистр

TCNT1 опять выполняет сложение. Как видно на рис. 4.10 (B), "высокая" составляющая выходного сигнала составляет 6 тактовых периодов, а "низкая" — 4. Таким образом, коэффициент заполнения g = 6/10 или g = 3/5.

Аналогично, диаграмма C на рис. 4.10 показывает соотношения для инвертированного выходного ШИМ-сигнала в соответствии с табл. 4.6. Как уже упоминалось в начале этого раздела, ШИМ-сигнал, сформированный с помощью таймера/счетчика T/C1, синхронный по фазе, что предотвращает появление неопределенных выходных сигналов при изменении содержимого регистра сравнения. Это достигается посредством того, что десять младших разрядов при обращении к OCR1A/OCR1B для записи попадают в регистр не сразу же, а вначале временно сохраняются в буфере, и записываются в регистр только при достижении счетчиком состояния TOP. Соответствующий пример для неинвертированного выходного ШИМ-сигнала показан на рис. 4.11.

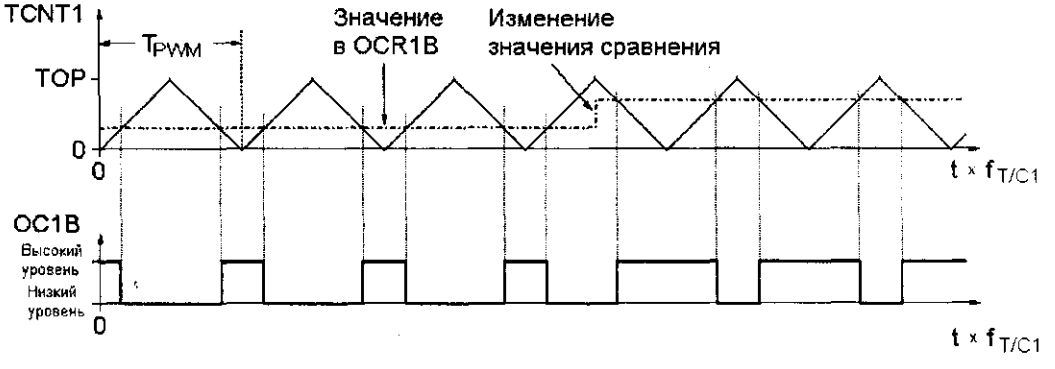


Рис. 4.11. Синхронизированное изменение содержимого регистра сравнения OCR1B

В противоположность этому, на рис. 4.12 показан гипотетический случай асинхронного изменения содержимого регистра сравнения. Высокий импульс, следующий после изменения выходного ШИМ-сигнала, имеет неопределенную длину (показан в виде заштрихованной области).

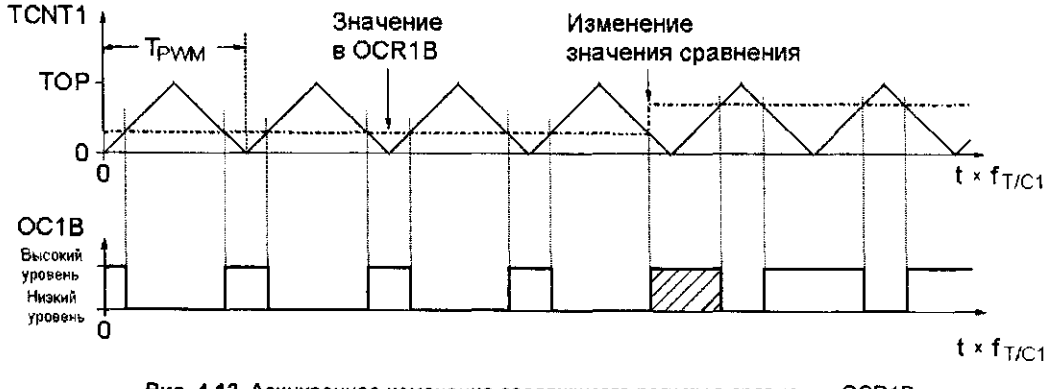


Рис. 4.12. Асинхронное изменение содержимого регистра сравнения OCR1B

В режиме ШИМ устанавливается флаг переполнения TOV1, если счетчик при достижении состояния 0 меняет направление счета на обратное. Это прерывание

по T/C1 при переполнении, как и при нормальной работе в режиме счетчика, вызывается в том случае, если установлен флаг общего разрешения прерываний I в регистре состояния SREG, а также флаг TOIE1 в регистре TIMSK. В соответствии с этим, прерывания при совпадении регистров TCNT1 и OCR1A/OCR1B вызываются тогда, когда в регистре TIMSK установлен флаг общего разрешения прерываний и флаг OCIE1A/OCIE1B.

Области применения таймера/счетчика T/C1

Таймер/счетчик T/C1 прекрасно подходит для фиксации и измерения отрезков времени, а также для выдачи точных частот следования и длительностей импульсов. При работе в режиме ШИМ с помощью T/C1 можно реализовать генератор импульсов прямоугольной формы с настраиваемым соотношением длительности импульса и паузы. При не слишком больших требованиях к скорости таймер/счетчик можно с успехом применять в качестве простого цифро-аналогового преобразователя. Для этой цели в простейшем случае достаточно разместить на выходе широтно-импульсного модулятора пассивный проходной фильтр нижних частот (рис. 4.13).

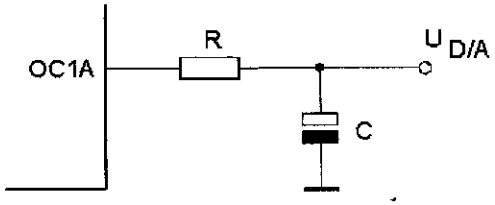


Рис. 4.13. Простой фильтр нижних частот для применения таймера/счетчика в качестве цифро-аналогового преобразователя при работе в режиме ШИМ

Значения величин R и C выбирают таким образом, чтобы их постоянная времени $\tau = R \cdot C$ была заметно больше продолжительности периода частоты ШИМ для того, чтобы в значительной степени подавить основную гармонику и последующие гармонические составляющие, полученные в составе прямоугольного сигнала. Однако выбирать большую постоянную времени также не следует, поскольку это приводит к увеличению периода переходных процессов до установки заданной выходной величины. На практике выбирают $\tau = T_{ШИМ} \cdot (10...1000)$.

Большой точности от цифро-аналогового преобразователя, показанного на рис. 4.13, ожидать, конечно же, не приходится. Это обусловлено величиной допустимого отклонения выходного напряжения на выводе OC1A. В этом случае речь идет о цифровом сигнале, а напряжение U_{OC1A} не достигает ни уровня рабочего напряжения V_{CC}, ни значения 0 В. В связи с тем, что U_{OC1A} зависит от свойств материала, температуры и нагрузки, его значение может находиться в относительно большом диапазоне. Это положение можно исправить только с помощью схемы, показанной на рис. 4.14. В ней выходное напряжение ШИМ не используется в качестве опорного для цифро-аналогового преобразователя, а посредством КМОП-переключателя (например, CD4053) переключается между стабильным опорным напряжением и чистой "землей". Предельная частота 3 дБ фильтра нижних частот, показанного на рис. 4.14, составляет $f_g = 1/2\pi R_2 C$. Она должна находиться в диапазоне $f_{ШИМ} / (10...1000)$.

Усиление схемы фильтра нижних частот составляет $A = -R_{2\,общ.} / R_1$. Оно отрицательное, поскольку основывается на инвертирующей схеме операционного усилителя. По этой причине, для достижения положительного выходного значения напряжения цифро-аналогового преобразователя, КМОП-переключатель подключают между 0 В и отрицательным опорным напряжением. Благодаря использованию операционного усилителя, схема в значительной степени независима от нагрузки. Внутреннее сопротивление КМОП-переключателя включено последовательно с R_1, и участвует в усилении. Обычно его значение находится в пределах 60–100 Ом. В связи с этим, сопротивление R_2 разделено на две части и состоит из последовательного триммера R_{2a} и постоянного сопротивления R_{2b}. С помощью R_{2a} в определенных пределах можно компенсировать сопротивление переключателя и калибровать допустимые отклонения опорного напряжения. Сопротивление R_{2a}, как правило, выбирают в пределах 10–20% от величины $R_{2общ.}$. Сопротивление R_3 предназначено для компенсации тока смещения и должно иметь значение R_2 и R_1 при параллельном включении.

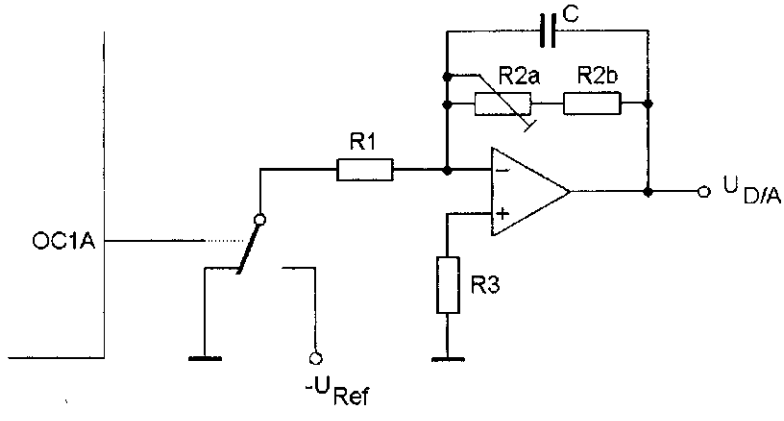


Рис. 4.14. Фильтр нижних частот с операционным усилителем в случае применения таймера/счетчика T/C1 в качестве цифро-аналогового преобразователя при работе в режиме ШИМ

Несколько примеров практического применения таймера/счетчика T/C1 рассмотрено в главе 16.

5 Сторожевой таймер

Для предотвращения перехода микроконтроллера в режим бесконечного цикла, когда на него невозможно повлиять извне (такие ситуации возникают при ошибках в программах пользователя) компания Atmel оснастила все микроконтроллеры базовой серии семейства AVR так называемым сторожевым таймером (рис. 5.1).

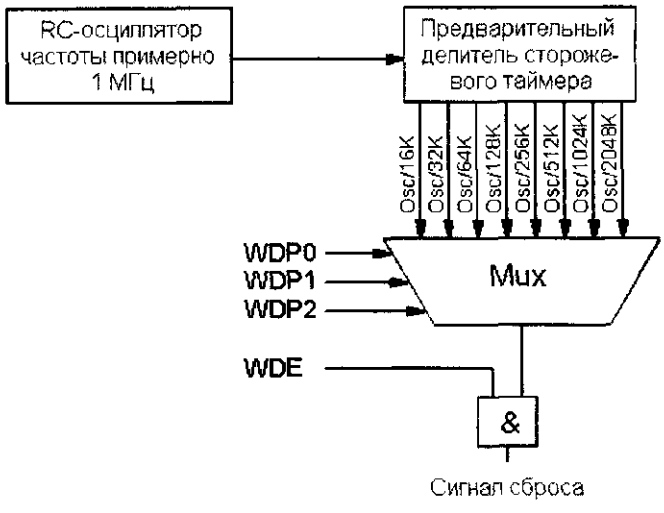


Рис. 5.1. Устройство сторожевого таймера

Если по истечении настраиваемого времени задержки программа пользователя не выполнит команду сброса системы, это сделает сторожевой таймер. После сброса сторожевого таймера отсчет времени задержки возобновляется. Если требуется контролировать ход выполнения программы, то программист должен активизировать сторожевой таймер и через регулярные отрезки времени включать в программу команду сброса, которая обеспечивает своевременный сброс перед началом нового отсчета времени.

Если в какой-либо простой программе контроль с помощью сторожевого таймера не требуется, то его можно отключить. По умолчанию, сторожевой таймер после поступления сигнала сброса по включению питания отключается. Для того чтобы предотвратить непреднамеренное отключение сторожевого таймера, необходимо придерживаться определенной процедуры, описанной ниже в разделе, посвященном регистру управления WDTCR.

Тактирование сторожевого таймера осуществляется с помощью встроенного RC-осциллятора. При комнатной температуре он генерирует колебания с частотой около 1 МГц при рабочем напряжении +5 В (см. рис. 2.10 в разделе "Генерирование такта системной синхронизации с помощью контура RC-осциллятора" главы 2). Сторожевой интервал может принимать одно из 8 значений от 16 до 2048 мс посредством предварительного деления на 1024 такта RC-контура. Когда

отсчет времени сторожевым таймером прекращается, выполнение программы продолжается с адреса $000, как по сигналу сброса при включении питания. Распределение интервалов времени при поступлении сигнала системного сброса в случае переполнения сторожевого таймера показано на рис. 3.36 в разделе "Сброс и обработка прерываний" главы 3.

Регистр управления WDTCR

Регистр управления сторожевого таймера WDTCR расположен в области ввода/вывода по адресу $21 (или по адресу $41 в RAM). После поступления сигнала сброса он инициализируется значением $00.

В микроконтроллерах базовой серии семейства AVR используются только разряды 0–4 регистра WDTCR (доступны для чтения и записи). Разряды 5–7 компанией Atmel зарезервированы и доступны только для чтения (всегда содержат лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$21 ($41)	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR

Разряды **WDP0–WDP2** устанавливают коэффициент предварительного деления входного такта сторожевого таймера. В табл. 5.1 показана типичная продолжительность отсчета времени для рабочего напряжения +5 В.

Таблица 5.1. Настройка времени до наступления сторожевой задержки при рабочем напряжении V_{CC}=+5В

WDP2	*WDP1*	*WDP0*	*Время до наступления сторожевой задержки*
0	0	0	около 16 мс
0	0	1	около 32 мс
0	1	0	около 64 мс
0	1	1	около 128 мс
1	0	0	около 256 мс
1	0	1	около 512 мс
1	1	0	около 1024 мс
1	1	1	около 2048 мс

Если установлен разряд **WDE** (**W**atch**d**og **E**nable), то сторожевой таймер активен. Если разряд WDE сброшен (лог. 0), то сторожевой таймер отключен. Во избежание непреднамеренного отключения сторожевого таймера, разряд WDE может быть сброшен только в том случае, если установлен разряд WDTOE.

Для отключения активного сторожевого таймера необходимо выполнить следующую последовательность действий.

1. С помощью одной команды должны быть установлены разряды WDE и WDTOE. Если сторожевой таймер активен, разряд WDE необходимо установить даже в том случае, если ранее он уже был установлен.
2. На протяжении последующих четырех импульсов такта системной синхронизации в разряд WDE записывается лог. 0, что приводит к отключению сторожевого таймера.

Если сторожевой таймер должен быть отключен, следует установить (лог. 1) разряд **WDTOE** (**W**atch**d**og **T**urn **O**ff **E**nable). После установки этого разряда он в течение четырех периодов такта системной синхронизации остается в состоянии лог. 1, а затем аппаратно сбрасывается в лог. 0 (это не учтено при тестировании в AVR-Studio вплоть до версии 1.50). Программа пользователя имеет возможность отключить сторожевой таймер посредством записи лог. 0 в разряд WDE только во время этих четырех тактов системной синхронизации.

Рассмотрим в качестве примера короткую программу, демонстрирующую отключение активного сторожевого таймера.

```
000100 b501 in r16, WDTCR                       ; Регистр управления в r16
000101 6108 sbr r16, 1<<WDTOE | 1<<WDE         ; Устанавливаем разряды WDE и WDTOE
000102 bd01 out WDTCR, r16                      ; WDTCR обратно, WDE+WDTOE = лог.1
000103 7f07 cbr r16, 1<<WDE                     ; Стираем разряд WDE
000104 bd01 out WDTCR, r16                      ; WDTCR записываем повторно
000105 0000 nop
000106 0000 nop
000107 0000 nop                                 ; WDTOE сбрасывается аппаратно
```

Ожидать истечения времени до сброса разряда WDTOE не обязательно. Три команды `nop` в представленном выше листинге предназначены только для обозначения момента времени сброса.

Для активизации сторожевого таймера в подобной процедуре нет необходимости. Достаточно просто установить разряд WDE в лог. 1.

6 Асинхронная передача данных через приемопередатчик UART

Универсальный асинхронный приемопередатчик UART отсутствует в микроконтроллере AT90S1200.

При синхронной последовательной передаче данных (например, в случае интерфейса SPI или шины I^2C) синхронизируется передача отдельных битов данных с помощью одновременно передаваемого передатчиком тактового сигнала. При этом особых требований к такому тактовому сигналу в отношении синхронизации не предъявляется. Так, например, при использовании нескольких синхронных протоколов передачи такт для приспособления к более медленным устройствам, подключенным к шине, растягивается по времени.

Синхронная последовательная передача данных применяется, главным образом, на уровне печатных плат, в том числе — для обмена данными между различными интегрированными блоками в составе схемы микроконтроллера и различными периферийными схемами (например, для обработки видеосигнала или управления звуком через шину I^2C телевизионного приемника).

В противоположность этому, при асинхронной передаче данных тактовый сигнал не передается. Это выдвигает строгие требования к распределению интервалов времени в каналах передачи и приема. По этой причине временная развертка для систем передачи данных, работающих в асинхронном режиме, в большинстве случаев имеет кварцевую стабилизацию.

Главной областью применения асинхронной передачи данных, как правило, является не обмен данными в составе схемы, а коммуникация между блоками, разделенными пространственно и обладающими признаками собственного интеллекта. В качестве примера можно назвать связь между персональным компьютером и принтером, модемом, программирующим устройством или регистратором данных. Поскольку асинхронная передача данных может осуществляться на большие расстояния вплоть до нескольких сотен метров, ее протоколы не предусматривают уровней TTL, очень подверженных помехам.

Распространенные стандарты асинхронной передачи данных

Очень устойчив к помехам **токовый интерфейс** с силой тока 20 мА, который может применяться для асинхронной передачи данных со скоростью 9600 бод. Этот интерфейс не нормирован, однако, благодаря простоте применения, нашел широкое применение на протяжении многих лет.

Соединение двух устройств здесь осуществляется через замкнутый приемопередающий контур. На активной стороне в контуре протекает постоянный ток силой 20 мА. Для передачи сигналов низкого уровня подвод тока прерывается, а при передаче сигнала высокого уровня контур остается замкнутым.

В случае токовых интерфейсов можно легко реализовать разделение потенциалов между передающей и принимающей стороной через оптоэлектронное устройство, как это показано на рис. 6.1, поэтому они пригодны для передачи данных на расстояние до 1000 м и по-прежнему находят широкое применение в промышленности.

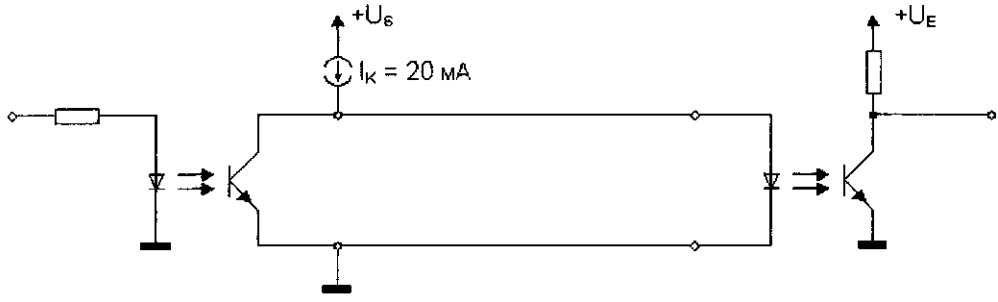


Рис. 6.1. Пример передающего контура для токового интерфейса

Уже устаревшие, но все еще широко распространенные интерфейсы **RS232C** или **V.24** передают сигналы низкого и высокого уровня как значения напряжения. При этом на стороне приемника высокому уровню TTL сигнала (+2...+5 В) выделен диапазон напряжений –3...–15V ("Отметка"), а низкому уровню TTL сигнала (0...+0,8 В) — диапазон напряжений +3...+15 В ("Пробел"). На стороне передатчика нижняя граница для компенсации потерь напряжения в линии поднята до +5 В и, соответственно, –5 В. На рис. 6.2 показан уровень сигнала, допускаемый согласно нормам.

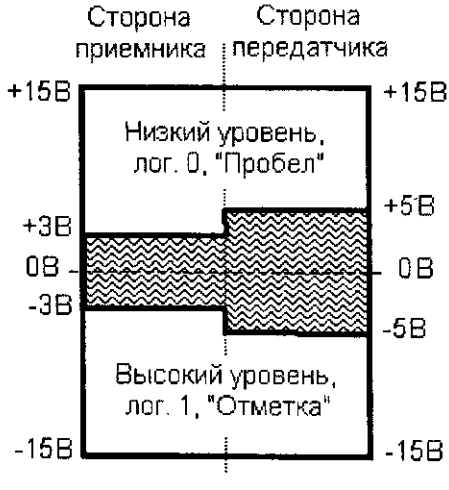


Рис. 6.2. Уровень напряжения в соответствии со стандартом RS232C

Для преобразования уровня от стандарта TTL в стандарт V.24 можно приобрести целый ряд интегральных схем. Пожалуй, наиболее распространенным драйверным модулем для этой цели является микросхема MAX232 компании Maxim (или одна из ее многочисленных производных), которая благодаря встроенным

преобразователям напряжения может обходиться единственным видом рабочего напряжения +5 В.

Для формирования соединения с квитированием стандарт также предусматривает некоторые управляющие сигналы, которые, впрочем, не применяются в принудительном порядке. Однако они также не должны оставаться неподключенными в схеме, поскольку это может привести к ошибочным интерпретациям в протоколе.

В простейшем случае для коммуникации между двумя устройствами достаточно иметь простое соединение с помощью трех проводов: скрещенные передающий и принимающий провод TxD и RxD, а также провод заземления. Не задействованные управляющие выводы могут быть заняты согласно рис. 6.3.

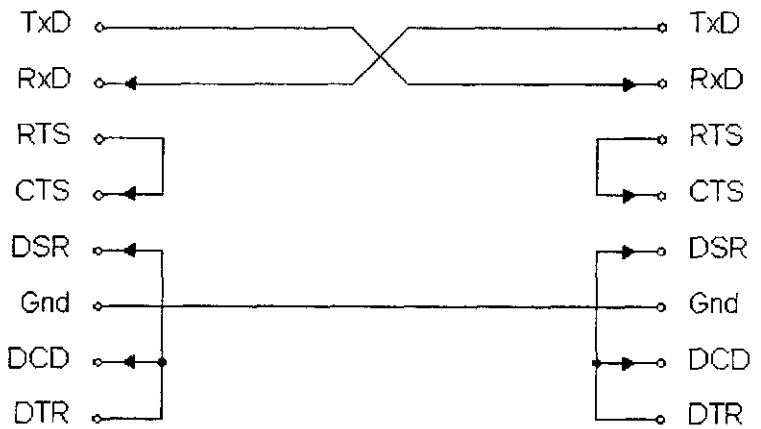


Рис. 6.3. Простое трехпроводное соединение между двумя приборами

На практике, на интерфейсе RS232C невозможно достичь скорости передачи данных выше 19200 бод, а из-за гальванического соединения и возникающих нарушений в работе, обусловленных токами переходных процессов в проводе заземления, передачу данных на расстояния свыше 15–20 метров осуществить невозможно, хотя стандарт и определяет допустимый предел в 30,5 м.

Растущие требования к дальности и скорости передачи данных привели к разработке новых стандартов асинхронной передачи данных. С помощью интерфейсов **RS423A** или **V.11** стали возможны скорости передачи данных вплоть до 100000 бод и дальности до 1200 метров. В случае интерфейса RS423A, передатчик и приемник соединены коаксиальным кабелем сопротивлением 50 Ом с волновым сопротивлением Z (рис. 6.4).

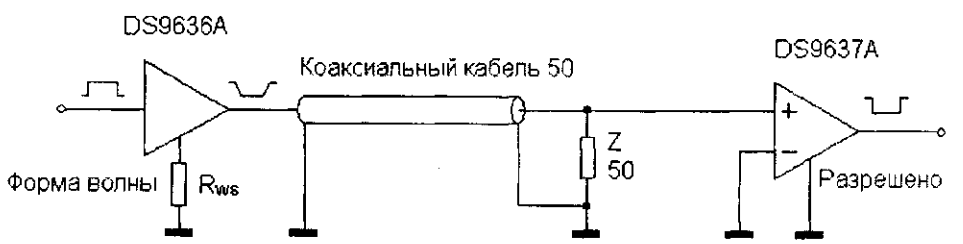


Рис. 6.4. Передача через коаксиальный кабель в соответствии с требованиями стандарта RS423A

Драйверы для преобразования необходимого уровня сигнала из стандарта TTL в стандарт V.11 можно приобрести у различных фирм-изготовителей. В качестве возможных примеров можно было назвать передатчик DS9636A с интерфейсом RS423A и приемник DS9637A с интерфейсами RS422A / RS423A от компании National Semiconductor. С помощью сопротивления R_{WS} время нарастания и ниспадания выходного напряжения драйвера DS9636A можно настроить в пределах от 1 до 100 мс.

Интерфейс RS423A, так же как и интерфейс RS232C, работает асимметрично, то есть, провод заземления служит в качестве обратного провода. По этой причине в среде с высоким уровнем помех применять этот интерфейс не целесообразно, однако, если нет необходимости передачи данных на очень большие расстояния, то схема, показанная на рис. 6.4, может дать ощутимые преимущества по сравнению с интерфейсом RS232C.

Еще больше преимуществ дает интерфейс, соответствующий стандартам **RS422A** или **V.10**. Здесь передача данных происходит симметрично, то есть, драйвер передатчика имеет выходной мостовой каскад, через который одновременно передается сигнал и инвертированный сигнал. Информация здесь также представлена не в виде абсолютного значения напряжения между выходом и "землей", а как разница между обоими выходными напряжениями.

Благодаря применению витой пары, достигается высокая помехоустойчивость, поскольку рассеянные импульсы помех взаимно устраняют друг друга (рис. 6.5).

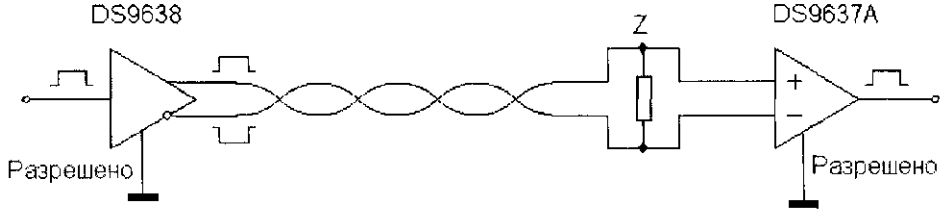


Рис. 6.5. Передача сигналов с помощью витой пары через интерфейс RS422A

Поясним это на примере. Если на выходе драйвера напряжение +3 В, то напряжение на его инвертирующем выходе на втором проводе, соответственно, будет –3 В. В результате разница двух выходных напряжений составляет 6 В. Когда оба напряжения из-за импульсов помех повышаются на +1 В, то первый провод имеет потенциал +4 В по отношению к "земле", а второй — –2 В, вследствие чего разница напряжений по прежнему составляет 6 В.

Благодаря такому методу, возможны скорости передачи данных вплоть до 10 Мбод при дальности до 1200 м.

Драйверы для преобразования уровня сигнала из стандарта TTL в стандарт RS422A можно приобрести у различных компаний-изготовителей. В качестве примера можно назвать передатчик DS9638 с интерфейсом RS422A от компании National Semiconductor. Как и в случае стандарта RS423A, может быть применен передатчик DS9637A.

Возможные скорости передачи данных в зависимости от требуемых расстояний передачи для всех рассмотренных интерфейсов показаны на рис. 6.6. Сравнение параметров различных интерфейсов показано в табл. 6.1.

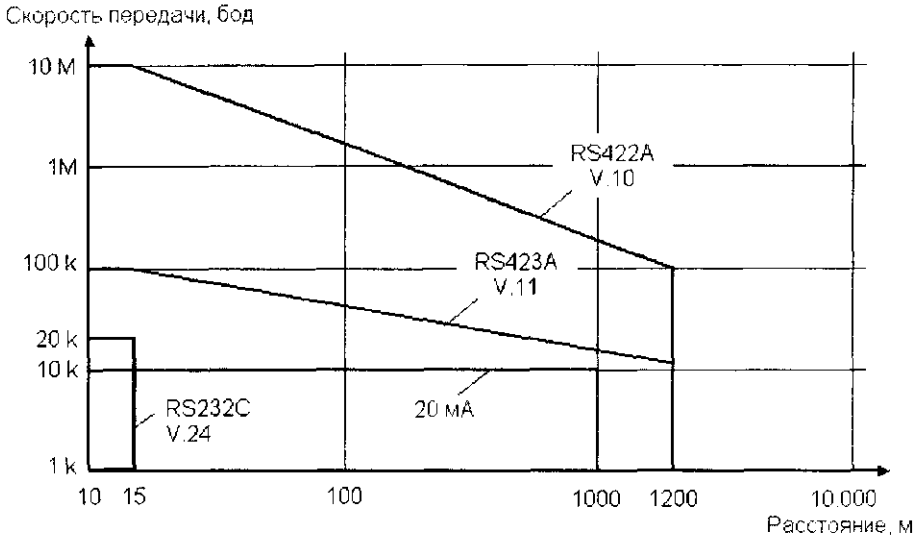


Рис. 6.6. Допустимые значения скорости передачи данных в зависимости от расстояния для асинхронных интерфейсов различных стандартов

Таблица 6.1. Сравнение основных параметров интерфейсов рассмотренных стандартов

Параметр	*20 мА*	*RS232C*	*RS423A*	*RS422A*
Вид передачи	Симметричная	Асимметричная	Асимметричная	Симметричная
Вид провода	Витой	Витой	Коаксиальный	Витой
Максимальная скорость передачи данных, бод	10 К	20 К	100 К	10 М
Максимальная длина провода	1000 м	15 м	1200 м	1200 м
Максимальный выходной сигнал, без нагрузки	20 мА	±25 В	±6 В	Разница ± 6 В
Выходной сигнал, с нагрузкой	20 мА	±5 В	±3,6 В	Разница ± 2 В
Чувствительность входа	10 мА	±3 В	±0,2 В	Разница ± 0,2 В

Формат передачи по асинхронному интерфейсу

В соответствии с определением стандарта V.24, в линии передачи данных в состоянии ожидания установлена лог. 1 ("Отметка"). Передача может быть начата в любой момент времени. Для того чтобы передать приемнику сообщение о начале передачи, посылается стартовый бит с уровнем лог. 0 ("Пробел"). После этого следуют разряды данных (в большинстве случаев 7 или 8, их число оговаривается

в протоколе передачи данных), при этом сначала передается младший значащий бит.

Для повышения надежности передачи данных может быть добавлен бит четности. Бит четности дополняет разряды данных таким образом, что достигается прямая четность (сумма всех битов, включая бит четности, четная) или непрямая четность (сумма всех битов, включая бит четности, нечетная). Требуемый вид четности (отсутствие, прямая или непрямая) также должен быть оговорен в протоколе передачи данных.

Передача заканчивается стоп-битом с уровнем лог. 1 ("Отметка"), который в некоторых микроконтроллерах может быть запрограммирован с продолжительностью 1; 1,5 или 2 длины бита. После отправки стоп-бита линия данных опять переходит в исходное состояние ожидания и готова к следующей передаче. Временная диаграмма передачи байта $E5 ($11100101_b$) с непрямой четностью и стоп-битом через интерфейс типа V.24 показана на рис. 6.7.

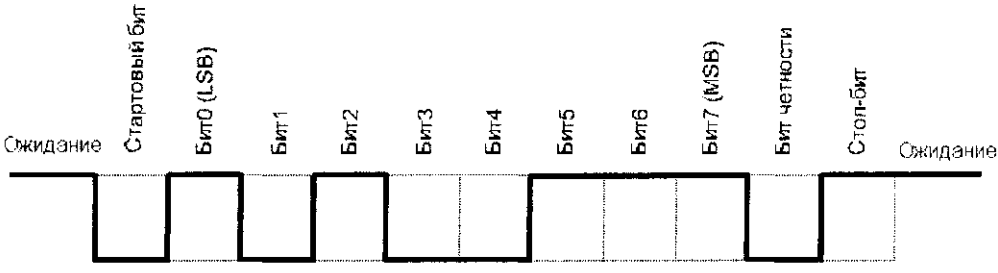


Рис. 6.7. Временная диаграмма передачи байта $E5 ($11100101_b$) с непрямой четностью и стоп-битом через интерфейс типа V.24

В связи с тем, что такт для синхронизации передачи не пересылается, начало разряда может быть распознано только по измерению времени, прошедшего с момента появления ниспадающего фронта стартового бита. Именно поэтому необходимо оговаривать скорости передачи данных, которые являются обязательными для передающей и принимающей стороны. Эти скорости определены в стандарте по интерфейсу RS232C, и во избежание ошибок при распознавании отдельных разрядов, их необходимо в точности придерживаться.

На рис.6.8 на основании представленного выше примера показаны последствия слишком большого различия между тактами передачи и приема, а также случая, когда передача осуществляется по нарастающему фронту такта, а опрос на стороне приемника — по ниспадающему фронту такта, который может попадать в середину импульса данных.

На рис. 6.8 показан случай, когда разряд 4 и разряд 7, а также бит четности распознаны неправильно. Поскольку в этом примере оговорена нечетность (непрямая четность), то ошибка может быть замечена на основании бита четности, так как байт, распознанный как $75, и установленный бит четности дают в результате прямую четность, но в случае возникновения двух ошибок бит четности об этом не просигнализировал бы.

Различие в тактах на передающей и принимающей стороне в рассмотренном примере составляет 20%, что недопустимо много. Такое значение было выбрано с той целью, чтобы четко показать эффект его влияния, однако во избежание оши-

бок и для обеспечения надежной передачи данных следует обязательно избегать отклонений, превышающих 3%. Точность передачи гарантирована, если ее фактическая скорость не отклоняется от указанных в стандарте значений более, чем на 2%.

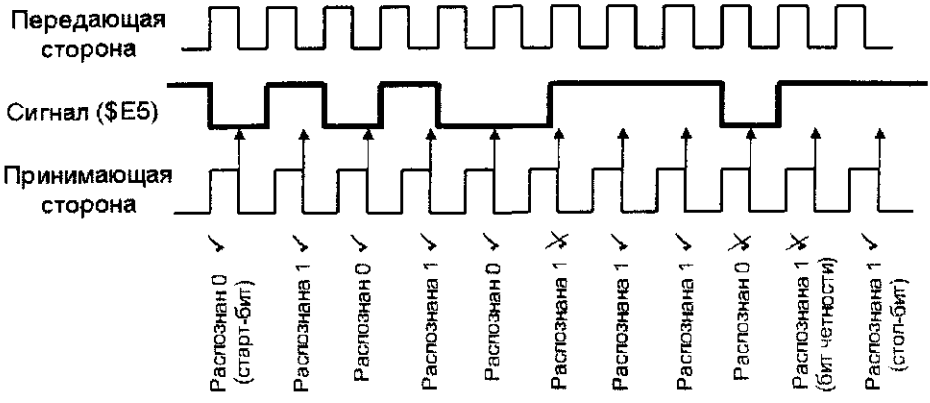


Рис. 6.8. Влияние различных тактов передачи на передающей и принимающей стороне

Физическое устройство приемопередатчика UART

Конечно же, асинхронную передачу данных можно реализовать и с помощью программного обеспечения (программная реализация UART для микроконтроллера AT90S1200 рассматривается в соответствующем разделе главы 16), однако это будет занимать довольно много ресурсов центрального процессора, и потому компания Atmel оснастила всех представителей базовой серии семейства AVR (за исключением модели AT90S1200) аппаратным приемопередатчиком UART. Он может работать в дуплексном режиме и, благодаря этому, в состоянии одновременно передавать и принимать данные, тем самым чувствительно разгружая центральный процессор.

Для работы UART выделены в общей сложности четыре регистра в области ввода/вывода. Наряду с регистром управления UCR (**U**ART **C**ontrol **R**egister), предназначенного для управления функциями приемопередатчика и для разрешения/запрета прерываний UART, используется также регистр состояния USR (**U**ART **S**tatus **R**egister); регистр данных UDR (**U**ART **D**ata **R**egister), физически состоящий из двух регистров (обращение к этим регистрам осуществляется по одному и тому же адресу, причем один из них используется для передачи, а другой — для приема данных), а также регистр UBRR (**U**ART **B**aud **R**ate **R**egister) для настройки требуемой скорости передачи данных с помощью встроенного контроллера, позволяющего устанавливать наиболее распространенные скорости передачи по стандарту RS232C.

Универсальный асинхронный приемопередатчик UART микроконтроллеров базовой серии семейства AVR может автоматически распознавать два вида ошибок передачи данных и вызывать три различных прерывания.

Передающий элемент UART (трансмиттер)

Схема передающего элемента приемопередатчика UART показана на рис. 6.9. Передача данных начинается с записи подлежащего передаче байта в регистр ввода/вывода UDR. Когда после передачи стоп-бита происходит запись следующего байта в регистр UDR, новый байт немедленно загружается в сдвиговой регистр, после чего начинается передача.

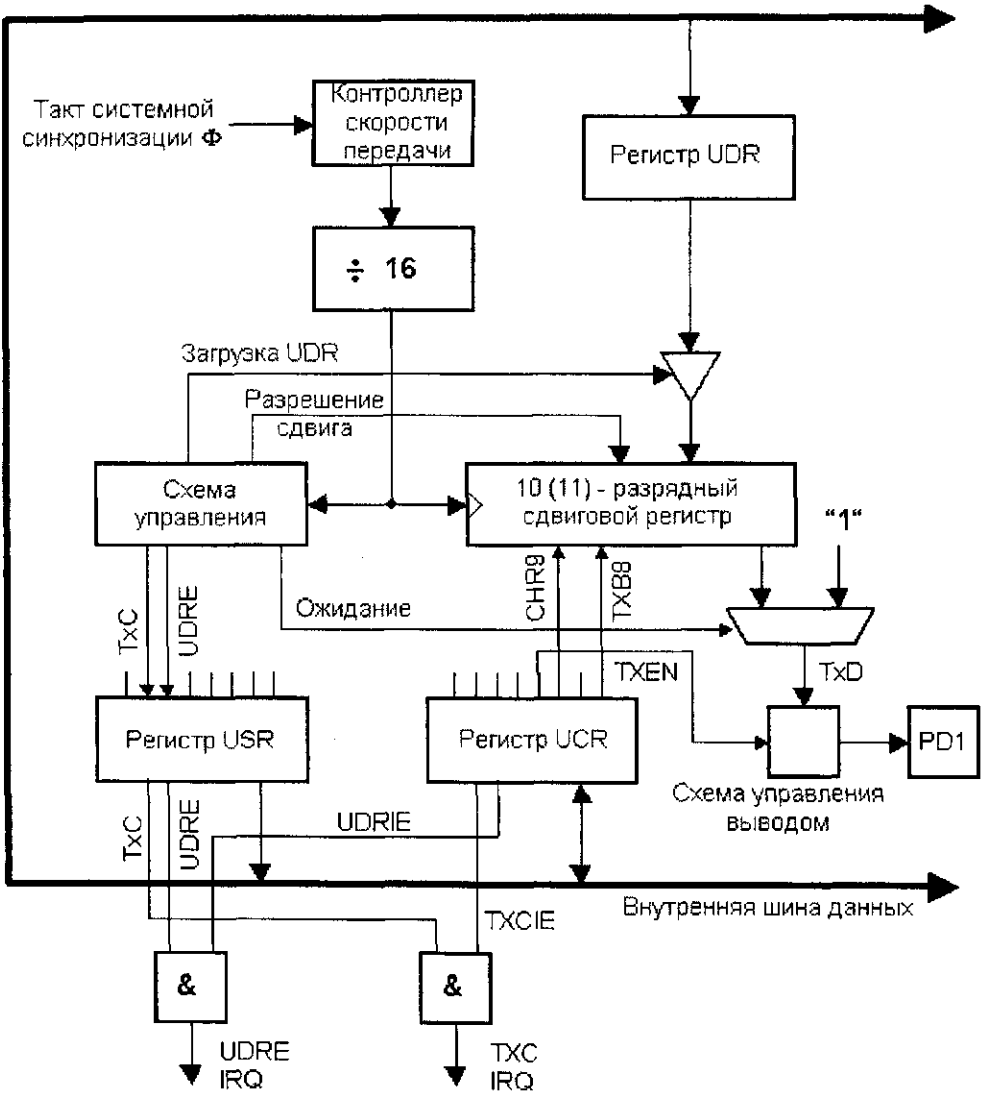


Рис. 6.9. Блок-схема передающего элемента (трансмиттера) приемопередатчика UART

Если в регистр UDR в процессе передачи записывается новый байт, то он может быть загружен в сдвиговой регистр, а передача его может начаться с помощью стоп-бита только после окончания текущей передачи данных.

Как только байт из регистра UDR переносится в сдвиговой регистр передачи, в регистре состояния USR устанавливается флаг UDRE (**U**ART **D**ata **R**egister **E**mpty — регистр данных приемопередатчика пуст), что указывает на готовность приемопередатчика к приему нового байта данных в регистр UDR.

Общее число переданных битов может составлять 10 или 11, в зависимости от того, какая длина слова данных была установлена с помощью разряда CHR9 регистра UCR: 8 разрядов (CHR9 = лог. 0) или 9 разрядов (CHR9 = лог. 1). Девятый разряд может быть применен для записи дополнительных информационных данных, как бит четности или как еще один стоп-бит.

При приеме подлежащего передаче байта в разряды 1–8 сдвигового регистра бит 0 (старт-бит) этого регистра автоматическое стирается, и в том случае, если CHR9 = лог. 1 и установлен стоп-бит (разряд 9 при CHR9 = лог. 0 или разряд 10 при CHR9 = лог. 1), разряд TXB8 из регистра управления UCR копируется в разряд 9 сдвигового регистра.

Вместе со следующим тактовым импульсом подлежащий передаче байт из регистра UDR переносится в сдвиговой регистр, бит 0 сдвигового регистра в качестве стартового бита переносится на вывод TxD, после чего следуют 8/9 разрядов данных (вначале младший разряд) и стоп-бит. Если во время текущей передачи данных в регистр UDR загружается новый байт, то этот байт сразу же после передачи стоп-бита передается в сдвиговой регистр, и начинается новая передача. Одновременно с передачей этого байта в сдвиговой регистр в регистре состояния USR устанавливается флаг UDRE, указывающий на то, что передающий элемент приемопередатчика UART опять готов к приему нового символа в регистр UDR.

Если после передачи стоп-бита в регистре UDR не окажется нового слова данных, ожидающего передачи, то флаг UDRE, установленный при передаче предыдущего байта из регистра UDR в сдвиговой регистр, будет содержать лог. 1 до тех пор, пока в регистр UDR не будет опять записан байт. После этого флаг UDRE сбрасывается. После пересылки стоп-бита в регистре состояния USR устанавливается флаг **TXC** (UART **T**ransmit **C**omplete — передача через UART завершена), указывающий на то, что слово данных было передано, и данные, ожидающие передачи, отсутствуют.

С помощью разряда **TXEN** регистра управления UCR работа трансмиттера может быть заблокирована (TXEN = лог. 0) или разблокирована (TXEN = лог. 1). Если трансмиттер заблокирован, то вывод PD1 может быть использован в качестве общего входа/выхода. Если он разблокирован, то выход сдвигового регистра будет соединен с выводом PD1, невзирая на настройку DDD1 в регистре направления передачи данных DDRD.

Принимающий элемент приемопередатчика UART (ресивер)

Блок-схема принимающего элемента асинхронного приемопередатчика UART показана на рис. 6.10.

Схема управления приемом опрашивает входной сигнал 16 раз во время каждого периода передачи данных. Если в ждущем режиме на линии приема распознается уровень лог. 0, это расценивается как ниспадающий фронт стартового им-

пульса, и начинается последовательность действий для распознавания действительного стартового бита.

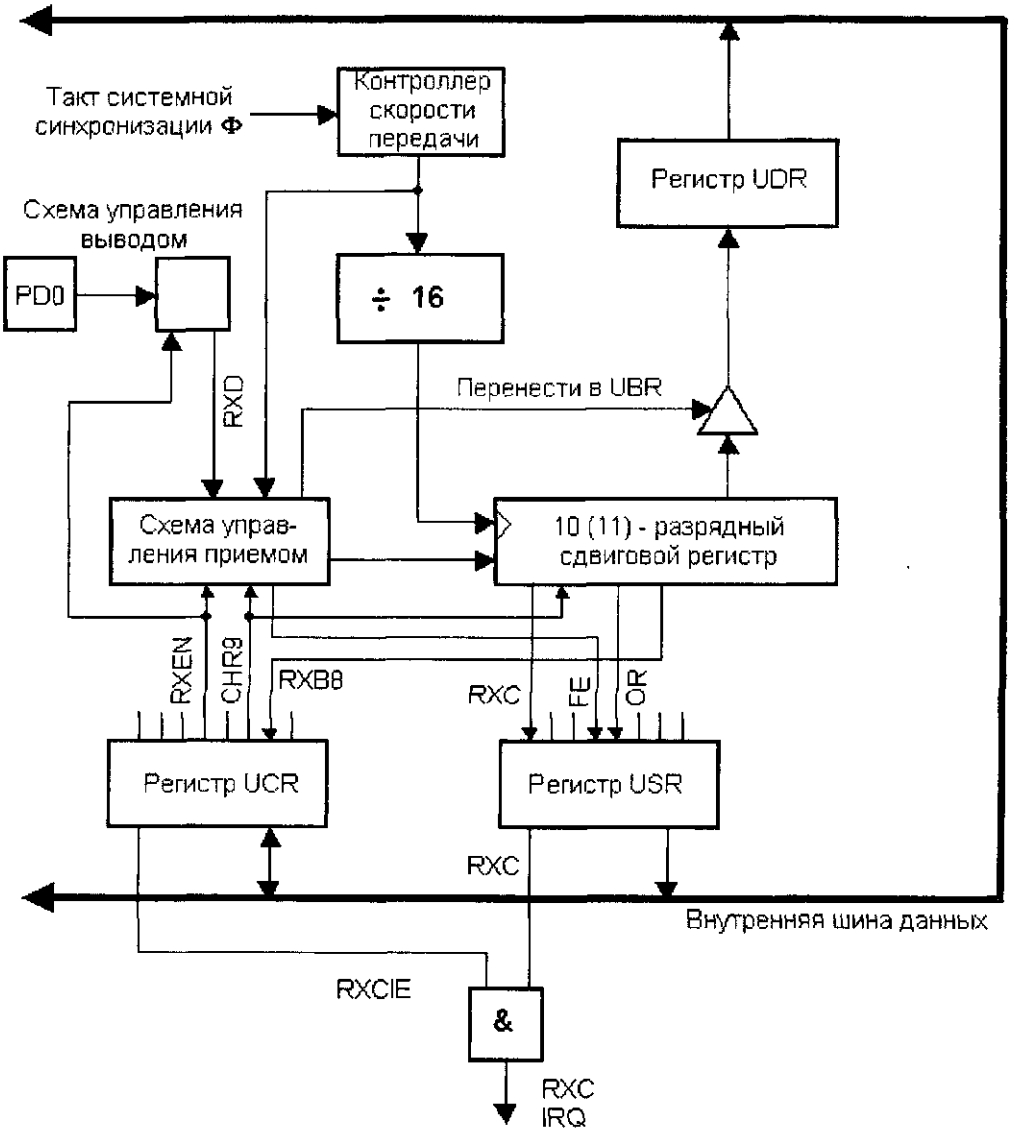


Рис. 6.10. Блок-схема принимающего элемента приемопередатчика UART

К 8-му (а также к 9-му и к 10-му) моменту времени опроса (считая от момента, когда был распознан ниспадающий фронт) линия приема должна содержать низкий уровень сигнала. Если при этих трех сканированиях два или три раза будет распознан высокий уровень, то стартовый бит будет отклонен как импульс помехи, и схема управления приемом опять начинает поиск ниспадающего фронта стартового бита.

Если будет обнаружен действительный стартовый бит, то поступившие последовательно один за другим разряды данных будут записаны в сдвиговой регистр. Как и в случае стартового бита, здесь действующим является уровень сигнала на линии приема вплоть до 8-го, а также 9-го и 10-го момента опроса, считая от начала поступающего разряда. Логический уровень, который распознается, как минимум, при двух опросах из трех, будет воспринят как значение бита. На рис. 6.11 показан процесс опроса для приема символа, состоящего из 8 бит (без бита четности).

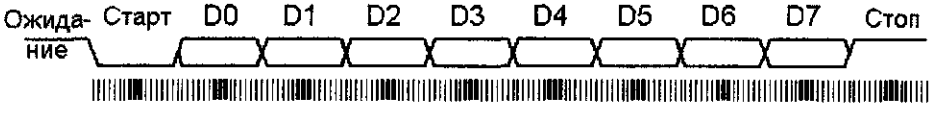


Рис. 6.11. Процесс опроса для приема символа, состоящего из 8 бит

Для стоп-бита как минимум два из трех опросов должны дать в результате лог. 1. Если этого не происходит, то в регистре состояния USR устанавливается флаг ошибки кадрирования **FE** (**F**raming **E**rror), указывающий на то, что стоп-бит поступившего символа был распознан как некорректный. Программа пользователя перед чтением регистра UDR должна постоянно проверять флаг FE для того, чтобы распознать потенциально недействительный символ в регистре приема.

По окончании цикла приема данных в регистре состояния USR всегда устанавливается флаг RXC, и выполняется загрузка прочитанного символа в регистр приема UDR, независимо от того, был распознан корректный или недействительный стоп-бит.

Если с помощью разряда CHR9 в регистре управления UCR было указано, что длина слова данных составляет 9 бит (CHR9 = лог. 1), то этот девятый бит переносится в разряд RXB8 регистра UCR.

Если в сдвиговой регистр поступает новое слово данных еще до того, как будет считан уже принятый и находящийся в регистре UDR байт, то новый символ не может быть перенесен в регистр UDR и теряется. В этом случае в регистре USR для индикации переполнения устанавливается флаг OR. Этот флаг буферизированный. Это означает, что после считывания действительно верного символа он обновляется в соответствии с содержимым регистра UDR. Именно поэтому программа пользователя после чтения регистра UDR должна постоянно проверять флаг OR, чтобы распознать потерю одного поступившего символа.

С помощью разряда **RXEN** регистра управления UCR принимающий элемент приемопередатчика UART может быть заблокирована (RXEN = лог. 0) или разблокирована (RXEN = лог. 1). Если принимающий элемент приемопередатчика заблокирован, то вывод PD0 может применяться как общий вход/выход. Если же он разблокирован, то вход принимающего элемента приемопередатчика будет соединен с выводом PD0, невзирая на настройку DDD0 в регистре направления передачи данных DDRD.

Регистры UART

Регистр ввода/вывода данных UDR

Этот регистр приемопередатчика UART физически состоит из двух регистров, обращение к которым происходит по одному и тому же адресу, причем один из них используется для передачи, а другой — для приема данных. При чтении из UDR происходит обращение ко входному регистру, а при записи — к передающему регистру.

Регистр UDR располагается в области ввода/вывода по адресу $0C (или по адресу $2C в RAM). Он доступен для чтения и записи и после поступления сигнала сброса инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$0C ($2C)	MSB							LSB	UDR

Регистр состояния USR

Регистр состояния USR приемопередатчика UART находится в области ввода/вывода по адресу $0B (или по адресу $2B в RAM). После поступления сигнала сброса все разряды вплоть до разряда 5 (UDRE) инициализируются с помощью лог. 0. Разряд UDRE устанавливается в лог. 1 для того, чтобы указать, что приемопередатчик готов к передаче нового байта данных.

Разряд	7	6	5	4	3	2	1	0	
$0B ($2B)	RXC	TXC	UDRE	FE	OR	–	–	–	USR

Регистр состояния USR информирует программу пользователя о состоянии приемопередатчика UART.

В микроконтроллерах базовой серии семейства AVR используются только разряды 3–7 регистра USR. За исключением флага TXC, они доступны только для чтения. Разряды 0–2 зарезервированы компанией Atmel и доступны только для чтения (всегда содержат лог. 0).

Флаг **RXC** (UART **R**eceive **C**omplete — прием завершен) устанавливается в лог. 1 без учета возможных ошибок кадрирования, которые могли возникнуть во время передачи данных, если принятое слово данных было перенесено из сдвигового регистра в регистр UDR.

В том случае, если в регистре состояния установлен разряд I общего разрешения прерываний, а в регистре управления UCR установлен разряд RXCIE, то выполнение программы разветвляется с помощью флага RXC микроконтроллера, установленного по адресу UART Receive Complete ($009 в AT90S8515 и AT90S4414, $007 в AT90S2313; в модели AT90S1200 приемопередатчик UART отсутствует). Флаг RXC устанавливается в исходное состояние посредством чтения регистра UDR.

Если пользовательская программа будет считывать свои данные через прерывание UART Receive Complete, то в течение выполнения подпрограммы обслуживания прерывания обязательно должен быть считан регистр UDR. В противном

случае флаг RXC остается установленным, а подпрограмма обработки прерывания будет вызвана повторно сразу же после выхода.

Флаг **TXC** (UART Transmit Complete — передача приемопередатчиком завершена) будет установлен в лог. 1, если символ в сдвиговом регистре был передан полностью (то есть, включая стоп-бит), и из регистра UDR не ожидается новый байт данных. Флаг очень полезен в полудуплексном режиме работы, когда непосредственно после передачи необходимо переключиться в режим приема.

В том случае, если в регистре состояния установлен разряд I общего разрешения прерываний, а в регистре управления UCR установлен разряд TXCIE, то выполнение программы разветвляется с помощью флага TXC, установленного по адресу UART Transmit Complete ($00B в AT90S8515 и AT90S4414, $009 в AT90S2313; в модели AT90S1200 приемопередатчик UART отсутствует).

При входе в подпрограмму обработки прерывания флаг завершения передачи TXC сбрасывается аппаратно. Альтернативно, флаг TXC также может быть сброшен записью лог. 1 в разряд 6 регистра USR.

Флаг **UDRE** (UART Data Register Empty — регистр данных приемопередатчика пуст) устанавливается в лог. 1, если содержимое регистра UDR было перенесено в сдвиговой регистр. С его помощью пользователь получает уведомление о том, что приемопередатчик готов к передаче нового символа.

В том случае, если в регистре состояния установлен разряд I общего разрешения прерываний, а в регистре управления UCR установлен разряд UDRIE, выполнение программы разветвляется с помощью флага UDRE, установленного по адресу UART Data Register Empty ($00A в AT90S8515 и AT90S4414, $008 в AT90S2313; в модели AT90S1200 приемопередатчик UART отсутствует). Эта подпрограмма обработки прерывания будет выполняться до тех пор, пока будет установлен флаг UDRE. Флаг UDRE сбрасывается при записи байта данных в регистр UDR.

Если пользовательская программа будет передавать свои данные через прерывание UART Data Register Empty, то во время выполнения подпрограммы обслуживания прерывания в регистр UDR обязательно должен быть записан символ. В противном случае флаг UDRE остается установленным, и подпрограмма обработки прерывания после своего завершения будет запущена опять.

Во время сброса при включении питания флаг UDRE устанавливается в лог. 1, чтобы показать, что приемопередатчик готов к передаче нового байта данных.

Флаг **FE** (Framing Error) устанавливается в лог. 1 при обнаружении ошибки кадрирования. Это происходит, если при трех сканированиях стоп-бита был более одного раза обнаружен лог. 0, и тем самым стоп-бит был распознан как сигнал низкого уровня. Флаг FE сбрасывается, когда стоп-биту соответствует сигнал высокого уровня.

Пользовательская программа должна постоянно проверять флаг FE перед чтением регистра UDR, чтобы можно было распознать потенциально некорректный символ в регистре приема.

Флаг **OR** (Over Run — перегрузка) устанавливается в лог. 1, если один из символов, переданных в регистр UDR из сдвигового регистра, не был прочитан перед следующим поступившим символом. Этот флаг буферизирован (то есть, он

обновляется после считывания действительного символа из регистра UDR), поэтому пользовательская программа должна всегда проверять флаг OR после чтения регистра UDR, чтобы распознать потерю одного поступившего символа. Флаг перегрузки сбрасывается при переносе считанного символа в регистр UDR.

Регистр управления UCR

Регистр управления UCR находится в области ввода/вывода по адресу $0A (по адресу $2A в RAM). Разряды 2–7 доступны для чтения и записи, разряд 1 — только для чтения, а разряд 0 — только для записи. После поступления сигнала сброса регистр управления инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$0A ($2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UCR

Если разряд **RXCIE** (**RX** Complete **I**nterrupt **E**nable — разрешение прерывания по завершению приема) и разряд общего разрешения прерываний I в регистре состояния SREG установлены в лог. 1, то разрешается прерывание по завершению приема UART Receive Complete. Этот процесс начинается после установки флага завершения приема RXC в регистре USR.

Если разряд **TXCIE** (**TX** Complete **I**nterrupt **E**nable — разрешение прерываний по завершению передачи) и разряд общего разрешения прерываний I в регистре состояния SREG установлены в лог. 1, то разрешается прерывание по завершению передачи UART Transmit Complete. Этот процесс начинается после установки в регистре USR флага завершения передачи TXC.

Если разряд **UDRIE** (**UART Data Register Empty Interrupt Enable** — разрешение прерывания по опустошению регистра данных приемопередатчика) и разряд общего разрешения прерываний I в регистре состояния SREG установлены в лог. 1, то разрешается прерывание по опустошению регистра данных приемопередатчика UART. Этот процесс начинается после установки в регистре USR флага UDRE.

Если разряд **RXEN** (**Receiver Enable** — разрешение приема) установлен в лог. 1, то происходит разблокирование приемника, и вывод PD0 становится входом UART. Если разряд RXEN содержит низкий уровень, то принимающий элемент приемопередатчика UART блокируется, и вывод PD0 может использоваться в качестве обычного входа/выхода.

Если разряд RXEN содержит лог. 0, то флаги OR и FE не могут быть установлены. Если эти флаги все же установлены, то они с помощью RXEN не сбрасываются.

Если разряд **TXEN** (**Transmitter Enable** — разрешение передачи) установлен в лог. 1, то происходит разблокирование передатчика, а вывод PD1 становится выходом приемопередатчика UART. Если разряд TXEN содержит низкий уровень, то передающий элемент UART блокируется, и вывод PD1 может использоваться в качестве обычного входа/выхода.

Если разряд TXEN во время процесса передачи устанавливается в лог. 0, то передатчик не блокируется до тех пор, пока текущий символ в сдвиговом регист-

ре, а также символ, возможно, ожидающий на передачу в регистре UDR не будут полностью переданы.

Если разряд **CHR9** (**9**-Bit **Ch**aracters — символ длиной 9 бит) установлен в лог. 1, то слова данных, подлежащие передаче/считыванию имеют длину 11 бит (9 разрядов данных плюс стартовый и стоп-бит). Девятый бит из разряда TXB8 при передаче попадает в UCR, а при приеме — в разряд RXB8 регистра UCR. Девятый бит может быть использован для размещения дополнительных информационных данных, например, в качестве бита четности или второго стоп-бита.

Если разряд CHR9 установлен в лог. 0, то слова данных, подлежащие передаче/ считыванию имеют длину 10 бит (8 разрядов данных плюс стартовый и стоп-бит).

Разряд **RXB8** — это девятый бит данных считанного символа, если разряд CHR9 в регистре управления UCR содержит лог. 1.

Разряд **TXB8** — это девятый бит символа, подлежащего считыванию, когда разряд CHR9 в регистре управления UCR установлен в лог. 1.

Регистр скорости передачи данных UBRR

Регистр скорости передачи данных UBRR приемопередатчика UART находится в области ввода/вывода по адресу $09 (по адресу $29 в RAM). Он доступен для чтения и записи и после поступления сигнала сброса инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$09 ($29)	MSB							LSB	UBRR

Встроенный контроллер скорости передачи данных представляет собой делитель частоты, определяющий скорость передачи данных непосредственно на основании такта системной синхронизации **Ф**.

Скорость передачи может быть вычислена по следующему уравнению:

$$f_{Baud} = \Phi/(16\ (UBRR+1))$$

где f_{Baud} — скорость передачи в бодах, **Ф** — такт системной синхронизации; UBRR — содержимое 8-разрядного регистра UBRR (0...255)

Значения регистра UBRR для наиболее распространенных скоростей передачи данных представлены в табл. 6.2. Все значения, для которых погрешность получается меньше 2%, в табл. 6.2 выделены полужирным шрифтом.

Таблица 6.2. Значения регистра UBRR для наиболее распространенных скоростей передачи данных и частоты работы кварцевого генератора колебаний

Скорость передачи данных, бод	*1,8432 МГц*	*Погрешность (%)*	*3,6864 МГц*	*Погрешность (%)*	*4 МГц*	*Погрешность (%)*
1200	UBRR = **95**	0,0	UBRR = **191**	0,0	UBRR = **207**	0,2
2400	UBRR = **47**	0,0	UBRR = **95**	0,0	UBRR = **103**	0,2
4800	UBRR = **23**	0,0	UBRR = **47**	0,0	UBRR = **51**	0,2

Таблица 6.2. Окончание

Скорость передачи данных, бод	*1,8432 МГц*	*Погрешность (%)*	*3,6864 МГц*	*Погрешность (%)*	*4 МГц*	*Погрешность (%)*
9600	UBRR = **11**	0,0	UBRR = **23**	0,0	UBRR = **25**	0,2
14400	UBRR = 7	0,0	UBRR = **15**	0,0	UBRR = 16	2,1
19200	UBRR = **5**	0,0	UBRR = **11**	0,0	UBRR = **12**	0,2
2400	UBRR = **207**	0,2	UBRR = *287*	–	UBRR = *312*	–
4800	UBRR = **103**	0,2	UBRR = **143**	0,0	UBRR = **155**	0,2
9600	UBRR = **51**	0,2	UBRR = **71**	0,0	UBRR = 77	0,2
14400	UBRR = **34**	0,8	UBRR = **47**	0,0	UBRR = **51**	0,2
19200	UBRR = **25**	0,2	UBRR = **35**	0,0	UBRR = **38**	0,2

Значения, выделенные курсивом, превышают 255, и потому не могут быть установлены в регистре UBRR, имеющем длину всего 8 разрядов. Если потребуется соответствующая скорость передачи данных, то необходимо переходить на более низкие частоты колебаний кварцевого генератора.

7 Синхронная передача данных через последовательный интерфейс (SPI)

Последовательный интерфейс периферийных устройств SPI (Serial Peripheral Interface) используется только в микроконтроллерах AT90S4414 и AT90S8515 базовой серии семейства AVR.

В семействе микроконтроллеров AVR существует две принципиальные возможности последовательного обмена данными:

- обмен данными с помощью приемопередатчика UART (описан в предыдущей главе), который поддерживает режим асинхронной передачи;
- применение синхронного последовательного интерфейса периферийных устройств SPI.

Как можно предположить из его названия, интерфейс SPI служит, в первую очередь, для коммуникации микроконтроллера с периферийными блоками. В качестве таких блоков могут выступать простые сдвиговые регистры или буквенно-цифровые модули индикации, а также сложные микропроцессорные системы или системы регистрации данных. Многие фирмы-изготовители предлагают большой выбор блоков с интерфейсом SPI.

Интерфейс такого типа также применяется при программировании в последовательном режиме центральных процессоров микроконтроллеров базовой серии семейства AVR. Подробно этот вопрос рассматривается в разделе "Последовательный режим программирования" главы 11.

Через интерфейс типа SPI можно очень быстро и просто обмениваться данными между ведущим микроконтроллером (Master) и одним или несколькими ведомыми блоками (Slave). Строение интерфейса SPI схематически показано на рис. 7.1.

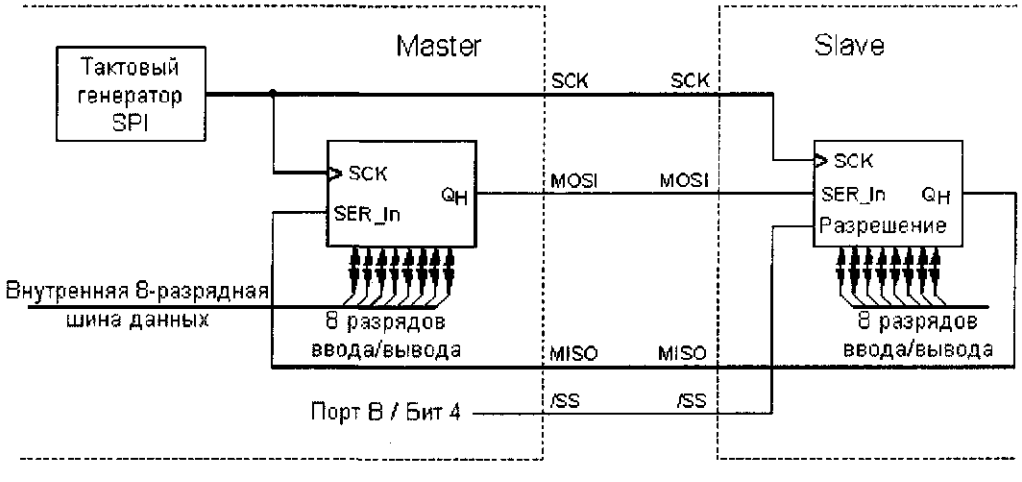


Рис. 7.1. Схематическое представление последовательного интерфейса SPI

При передаче данных через интерфейс SPI обмен данными всегда происходит между двумя устройствами: Master и Slave. Каждый микроконтроллер семейства AVR, оснащенный интерфейсом SPI, может быть сконфигурирован на режимы работы как Master, так и Slave посредством установки/сброса разряда MSTR в регистре управления интерфейса SPI. При этом Master берет на себя активную часть обмена данными, вызывая передачу и управляя процессом. Ведомое устройство (Slave) не может само быть активным. Оно принимает и передает данные только тогда, когда происходит его активизация со стороны ведущего устройства по линии /SS (**S**lave **S**elect — выбор ведомого). Это можно сравнить с сигналом /CS (**C**hip **S**elect — выбор кристалла) при использовании блоков памяти RAM или EPROM. Ведущее устройство также генерирует такт для передачи по выходной линии SCK. Для ведомого блока вывод SCK является входом, через который он получает от Master-устройства такт.

Если Slave активизируется ведущим устройство по линии /SS, то начинается обмен данными: Master записывает подлежащий передаче байт в свой регистр данных SPDR последовательного интерфейса SPI. С помощью каждого выработанного тактового импульса Master перемещает один бит данных на выход MOSI (**M**aster **O**ut **S**lave **I**n — выход ведущего, вход ведомого), а Slave одновременно в обратном направлении передает один бит на вход MISO (**M**aster **I**n **S**lave **O**ut — вход ведущего, выход ведомого) ведущего блока. По этой причине интерфейс SPI можно сравнивать с 16-ступенчатым регистром циклического сдвига, разделенным на два 8-разрядных сдвиговых регистра, первый из которых находится в ведущем устройстве, а второй — в ведомом. В результате, в течение цикла SPI, состоящего из восьми тактовых импульсов, Master и Slave обмениваются байтами данных.

По окончании передачи данных в регистре состояния интерфейса SPI (как в конфигурации Master, так и в конфигурации Slave) устанавливается флаг прерывания от интерфейса SPIF (**SPI** **I**nterrupt **F**lag). Этот флаг указывает на окончание передачи и вызывает запрос на прерывание после того как в регистре управления SPCR (**SPI** **C**ontrol **R**egister) будет установлен разряд SPIE (**SPI** **I**nterrupt **E**nable — разрешение на прерывание от интерфейса SPI), а в регистре состояния SREG — разряд I общего разрешения прерываний. В режиме "Master" текущая передача данных может быть преждевременно завершена выдачей в линию /SS сигнала лог. 1. Счетчик разрядов и внутренняя логика Slave в результате сбрасываются, режим "Slave" становится неактивным и переходит в высокоомное состояние (с тремя состояниями).

Как показано на рис. 7.2, к интерфейсу SPI в качестве ведомых могут быть одновременно подключены несколько периферийных устройств, однако активным будет только то из них, на вход /SS которого через порт B ведущего устройства будет подан уровень лог. 0. Выходы MISO незадействованных ведомых блоков находятся в высокоомном состоянии и не влияют на процесс передачи данных.

Блокам G1 и Gn на рис. 7.2 соответствуют полноценные ведомые интерфейсы SPI, у которых данные передаются в обоих направлениях. Блок G2, с точки зрения ведущего блока, является только блоком выдачи (в качестве примера такого блока можно привести цифроаналоговый преобразователь с интерфейсом SPI). Блок G3

по исполнению аналогичен блоку G2, но только как блок приема (например, цифроаналоговый преобразователь).

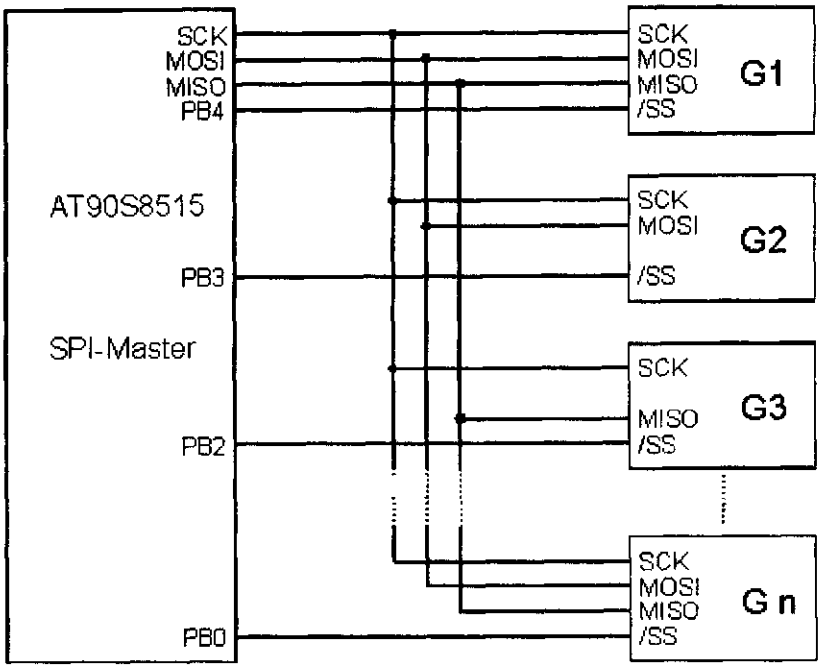


Рис. 7.2. Подключение нескольких устройства к интерфейсу SPI

Входы и выходы интерфейса SPI

Интерфейс SPI состоит из четырех линий: такта сдвига SCK, линий передачи данных MOSI и MISO, а также входа разблокирования /SS, активизируемого сигналом низкого уровня. Они выполнены в виде альтернативных функций порта B (табл. 7.1).

Таблица 7.1. Функции разрядов порта B для линий интерфейса SPI

Линия	*Вывод*	*Функция*
/SS	PortB / Разряд 4	Разрешение (Slave Select)
MOSI	PortB / Разряд 5	Линия данных (Master Out Slave In)
MISO	PortB / Разряд 6	Линия данных (Master In Slave Out)
SCK	PortB / Разряд 7	Такт сдвига (Shift ClocK)

Когда интерфейс SPI отключен (разряд SPE в регистре интерфейса SPCR содержит лог.0), то эти четыре линии функционируют как самые обычные линии ввода/вывода, состояние которых определяется по состоянию регистра направления передачи данных DDRB порта B. Однако, если интерфейс SPI активен (разряд SPE в регистре управления SPCR содержит лог. 1), то в общем случае действует следующее правило.

Входы интерфейса SPI после установки разряда SPE в регистре управления SPCR автоматически конфигурируются как входы, а соответствующий разряд в регистре направления передачи данных будет переписан.
Выходы интерфейса SPI должны быть определены пользовательской программой посредством установки соответствующего разряда в регистре направления передачи данных.

Направление передачи данных через выводы активного интерфейса SPI показано в табл. 7.2.

Таблица 7.2. Направление передачи данных через выводы активного интерфейса SPI

Вывод	*Направление передачи данных при конфигурации "Master"*	*Направление передачи данных при конфигурации "Slave"*
MOSI	Определяется пользователем	Вход
MISO	Вход	Определяется пользователем
SCK	Определяется пользователем	Вход
/SS	Определяется пользователем	Вход

Линия **SCK** является выходом, когда интерфейс SPI работает в режиме "Master", и входом, когда интерфейс SPI сконфигурирован для работы в режиме "Slave" (разряд MSTR в регистре управления SPCR). При работе в режиме "Slave" интерфейс игнорирует свой вход SCK до тех пор, пока на его линия /SS не появится сигнал низкого уровня.

Линия SCK делится внутренним тактом системной синхронизации **Ф** центрального процессора. Коэффициент деления может быть выбран с помощью разрядов SPR1 и SPR0 регистра управления SPCR. Когда Master начинает передачу данных, то генерируются восемь тактовых импульсов, которые синхронизируют передачу. Чтобы гарантировать надежную постановку бита данных, он сдвигается вперед с помощью одного фронта тактового сигнала SCK, а с помощью другого фронта принимается (рис. 7.3). Выбор фронта сигнала зависит от протокола передачи данных.

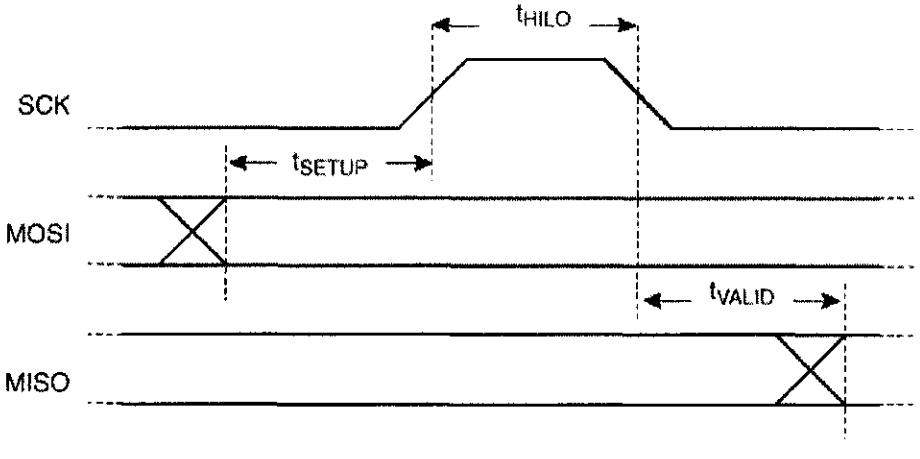


Рис. 7.3. Временная диаграмма передачи по интерфейсу SPI

На рис. 7.3 биты данных принимаются по нарастающему фронту тактового сигнала SCK, а дальнейший сдвиг осуществляется посредством ниспадающего фронта. Master передает бит данных на свой выход MOSI, и только по прошествии

времени t_{SETUP} следует нарастающий фронт тактового сигнала SCK, чтобы гарантировать, что бит после приема стабильно установлен.

После того как появился ниспадающий фронт тактового сигнала SCK, Master ожидает период времени t_{VALID}, чтобы убедиться в наличии бита данных от Slave, который можно было бы принять с помощью следующего фронта тактового сигнала SCK. Продолжительность времени t_{HILO} импульса высокого уровня тактового сигнала SCK, а также периодов времени t_{SETUP} и t_{VALID} зависит от требований, выдвигаемых изготовителем к параметрам периферийных устройств, подключаемых через интерфейс SPI. Для того чтобы можно было подключать также и более медленные периферийные устройства, частота тактового сигнала SCK может быть поделена вплоть до **Ф**/128.

Выводы **MISO** и **MOSI** служат для передачи и приема последовательных битов данных. На стороне Master MOSI — это выход данных, а MISO — вход. На стороне Slave эти два вывода меняются ролями. Как показано на рис. 7.2, к интерфейсу типа SPI может быть подключено несколько ведомых блоков. В этом случае все линии SCK соединены между собой, так же как и все линии MOSI и MISO. Несмотря на то, что по интерфейсу SPI, в принципе, могут быть соединены несколько ведущих устройств, активным может быть только одно из них. Этот активный Master выбирает по линии /SS требуемый блок Slave и пересылает через его выходы SCK и MOSI тактовые импульсы и биты данных на входы SCK и MOSI ведомых блоков. В течение того же цикла Slave может пересылать биты данных через свой выход MISO на вход MISO ведущего блока. Благодаря автоматическому переключению направления передачи данных на этих выводах (см. табл. 7.2), предотвращается возникновение конфликта при смене ведущего устройства в системах с несколькими такими устройствами.

Назначение линии /**SS** может меняться, в зависимости от того, определен ли интерфейс SPI как Master или как Slave. Ведомый блок деблокируется ведущим для передачи данных с помощью лог. 0 на выводе /SS, а его вывод MISO превратится в выход, как только пользовательская программа установит в лог. 1 разряд DDB6 в регистре направления передачи данных порта B. Когда на выводе /SS находится высокий уровень, Slave игнорирует свои входы SCK и MOSI, а затем переводит выход MISO в высокоомное состояние.

При работе в режиме "Master" вывод /SS может быть сконфигурирован как обычный выход или как вход для распознавания конфликтов в интерфейсе SPI. Этот выбор определяется разрядом DDB4 в регистре направления передачи данных порта B. Если этот разряд содержит лог. 1, то линия функционирует как общий выход, который не оказывает влияния на интерфейс SPI. Если разряд DDB4 в регистре направления передачи данных установлен в лог. 0 (вход), то линия /SS должна удерживаться в состоянии лог. 1, если система SPI должна корректно работать как Master.

Посредством сигнала низкого уровня на этом выводе Master получает сигнал о том, что другой ведущий блок выбрал его в качестве ведомого и будет передавать ему данные. Это заставляет выбранный ведущий блок немедленно освободить шину передачи данных по интерфейсу SPI и посредством автоматического переключения перейти в конфигурацию ведомого блока (см. табл. 7.2).

Во избежание серьезных повреждений выходного драйвера, аппаратно выполняются следующие действия.

1. Разряд MSTR в регистре управления SPCR устанавливается в лог. 0, и тем самым система интерфейса SPI конфигурируется на работу в режиме ведомого блока. Как следствие, выводы MOSI и SCK превращаются во входы.
2. Флаг SPIF в регистре состояния SPSR устанавливается в лог.0, и вызывается прерывание, как только в регистре управления SPCR будет установлен разряд SPIE, а в регистре состояния SREG — разряд общего разрешения прерываний I.

По этой причине подпрограмма обработки прерывания от интерфейса SPI при его переходе в режим Master всегда будет перепроверять, установлен ли еще разряд MSTR, если существует возможность внешнего перевода линии /SS в состояние лог. 0.

Когда разряд MSTR в результате требования перейти в режим Slave внешне установлен в лог. 0, пользовательская программа должна сама позаботиться о том, чтобы он опять был установлен.

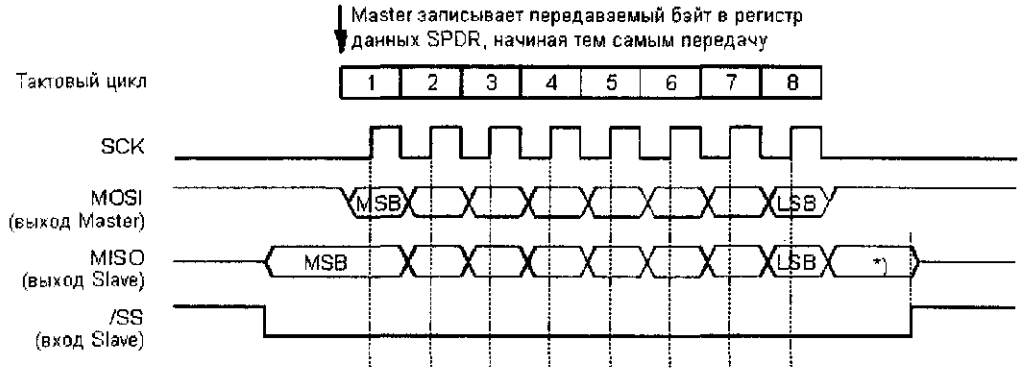
Протокол передачи

Для передачи данных через интерфейс SPI используются два принципиально различных формата передачи. Соответствующий формат выбирается с помощью разряда CPHA (**C**lock **Pha**se — фаза синхронизации) регистра управления SPCR.

Для каждого из двух форматов передачи (CPHA = лог. 0 и CPHA = лог. 1) активная фаза синхронизации может быть выбрана индивидуально посредством установки/сброса разряда CPOL в регистре данных SPDR.

На представленных ниже рисунках показаны четыре возможные конфигурации CPHA и CPOL. Во всех четырех случаях предполагается, что линия /SS ведущего блока или сконфигурирована как выход, или находится в состоянии лог. 1, а регистр управления SPCR инициализируется следующим образом: SPE = лог. 1 (интерфейс SPI разблокирован); DORD = лог. 0 (сначала выводится старший разряд байта данных — MSB). Передача через интерфейс SPI всегда инициализируется и управляется ведущим блоком (Master).

Чтобы начать передачу данных так, как это показано на рис. 7.4, Master переводит линию /SS, подсоединенную к одному из его портов, в состояние лог. 0. Для соответствующего ведомого блока передача начинается по ниспадающему фронту этого сигнала. Его выход MISO переходит из высокоомного в активное состояние, и старший разряд байта, находящегося в его регистре данных SPDR, появляется на выходе MISO. Собственно передачу данных Master начинает записью подлежащего передаче байта данных в свой регистр данных SPDR. Вслед за этим на выходе MOSI ведущего блока появляется разряд MSB. На протяжении первой половины первого тактового импульса тактовая линия еще остается в состоянии покоя для того, чтобы обеспечить стабильную установку на соответствующем входе бита данных от Master и Slave.

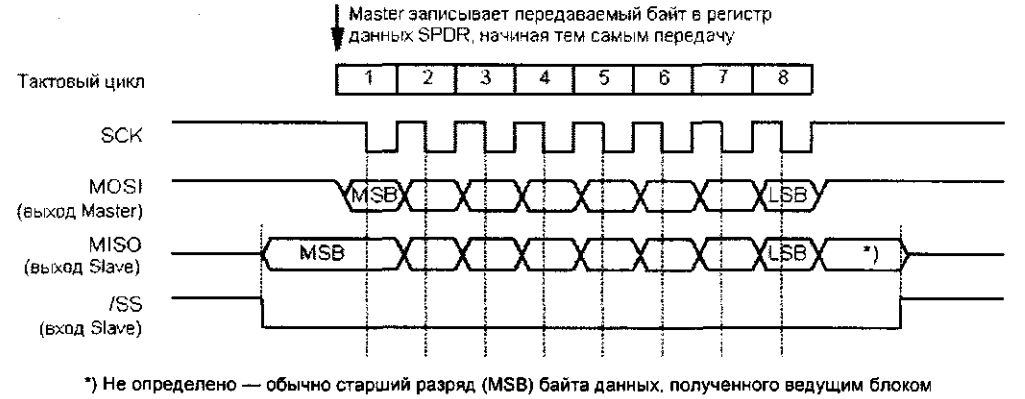


*) Не определено — обычно старший разряд (MSB) байта данных, полученного ведущим блоком

Рис. 7.4. Формат передачи через интерфейс SPI, если CPHA = лог. 0, CPOL = лог. 0

По нарастающему фронту первого и каждого последующего тактового импульса принимаются биты, расположенные на входах Master и Slave, а по ниспадающему фронту следующий бит сдвигается дальше. После восьмого тактового импульса передача данных завершена, флаги SPIF в регистрах состояния ведущего и ведомого блоков установлены, а содержимое сдвиговых регистров будут перенесено в соответствующие приемные буферы. Выход MOSI ведущего блока возвращается в состояние покоя (лог. 1), а на выходе MISO ведомого блока, как правило, находится старший разряд (MSB) байта, только что принятого ведущим блоком. Одновременно со сбросом линии /SS в исходное состояние (лог. 1) Master завершает передачу, Slave становится неактивным, а его выход MISO переходит в высокоомное состояние.

Для второго случая, когда CPHA = лог. 0 и CPOL = лог. 1 (рис. 7.5), условия аналогичны первому случаю с той разницей, что состояние покоя тактовой линии здесь устанавливается при лог. 1, биты данных перенимаются первым и каждым последующим тактовым импульсом, а сдвиг осуществляется по нарастающему фронту сигнала.



*) Не определено — обычно старший разряд (MSB) байта данных, полученного ведущим блоком

Рис. 7.5. Формат передачи через интерфейс SPI, если для CPHA = лог. 0, CPOL = лог. 1

Для третьего случая CPHA = лог. 1 и CPOL = лог. 0 (рис. 7.6).

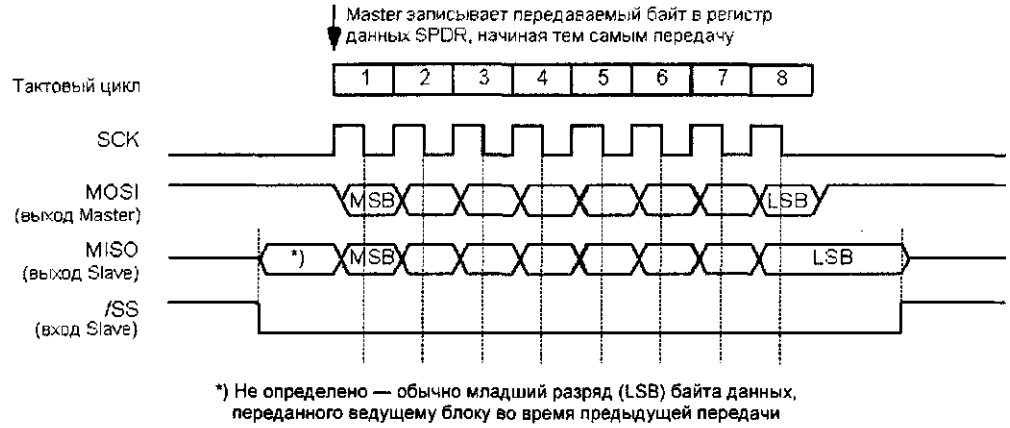


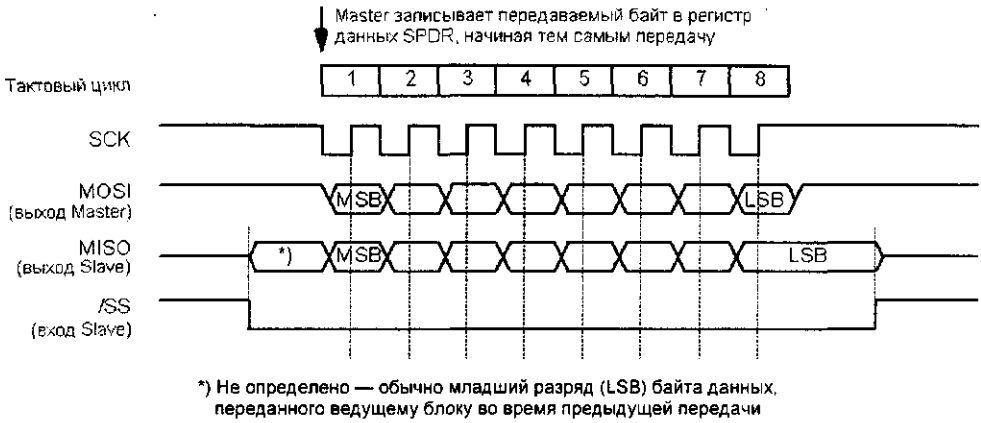
Рис. 7.6. Формат передачи интерфейса SPI, если CPHA = лог. 1, CPOL = лог. 0

Для того чтобы при этом режиме начать передачу данных, Master, как и в первом случае, переводит линию /SS, подсоединенную к одному из его портов, в состояние лог. 0. Блок Slave разблокирован, и его выход MISO переходит из высокоомного в активное состояние. Логический уровень на MISO для этого случая не определен, но, как правило, на MISO находится младший разряд байта, переданного во время предыдущей передачи от Slave к Master.

Собственно передачу данных Master в этом режиме точно так же начинает посредством записи байта данных, подлежащего передаче, в регистр SPDR. Для ведомого блока передача начинается по нарастающему фронту тактового сигнала. Старшие разряды подлежащих передаче байтов в ведущем и ведомом блоках с помощью нарастающего фронта первого тактового импульса устанавливаются на выходе MOSI ведущего блока выходе MISO ведомого блока. По ниспадающему фронту первого и каждого последующего тактового импульса они переносятся в на входы Master и Slave, а по нарастающему фронту следующий разряд сдвигается.

После восьмого тактового импульса передача данных завершается, устанавливаются флаги SPIF в регистрах состояния интерфейсов Master и Slave, а содержимое их сдвиговых регистров переносится в соответствующие буферы приема. Выход MOSI ведущего блока возвращается в состояние покоя (лог. 1), на выходе MISO ведомого блока остается младший разряд байта, только что переданного ведущему блоку. Одновременно с возвратом в исходное состояние линии /SS (лог. 1) Master завершает передачу в целом, Slave становится неактивным, а его выход MISO переходит в высокоомное состояние.

Для четвертого случая, когда CPHA = лог. 1, а CPOL = лог. 1 (рис. 7.7), условия аналогичны третьему случаю с той разницей, что состоянием покоя тактовой линии здесь является лог. 1, а биты данных сдвигаются по ниспадающему фронту первого и каждого последующего тактового импульса, а принимаются по нарастающему фронту.



*) Не определено — обычно младший разряд (LSB) байта данных, переданного ведущему блоку во время предыдущей передачи

Рис. 7.7. Формат передачи интерфейса SPI, если CPHA = лог. 1, CPOL = лог. 1

Системные конфликты SPI

В микроконтроллерах семейства AVR система SPI буферизирована одноразово в направлении передачи, и двукратно — в направлении приема. Это означает, что подлежащий передаче байт может быть записан в регистр данных SPI только тогда, когда будет завершена текущая передача. В противоположность этому, только что принятый байт немедленно переносится в параллельный буфер приема, вследствие чего сдвиговый регистр сразу же после передачи может осуществлять прием второго последовательного байта. Пока первый байт будет считываться из приемного буфера, второй байт не записывается пока не будет принят полностью, иначе принятый первым байт будет затерт следующим и потеряется.

Если во время передачи данных через интерфейс SPI предпринимается попытка записи в регистр данных, то для индикации сбоя в регистре состояния SPSR устанавливается флаг WCOL (**W**rite **Col**lision — конфликт записи). Поскольку система SPI в направлении передачи не использует двойной буферизации, то запись в регистр данных следует поставить на один уровень с прямой записью в сдвиговый регистр SPI. Это привело бы к разрушению только что переданного байта данных. По этой причине текущая передача данных будет доведена до конца, и новый байт не будет записан в сдвиговый регистр интерфейса SPI. Неудачная попытка записи индицируется с помощью флага WCOL.

Схема интерфейса SPI распознает конфликты записи как на стороне Master, так и на стороне Slave. В обычном случае, конечно же, от этой ошибки пострадают только ведомые блоки, поскольку ведущий блок знает, когда начал передачу, а ведомый блок не может оказать никакого влияния на передачу, начатую ведущим блоком. Если требуется, чтобы Slave избежал этой ошибки, рекомендуется проверить уровень сигнала на входном выводе /SS перед записью в регистр данных интерфейса SPI, и осуществлять обращение на запись только при высоком уровне сигнала.

Регистр управления SPCR

Регистр управления SPCR интерфейса SPI находится в области ввода/вывода по адресу $0D (по адресу $2D в RAM). После поступления сигнала сброса он инициализируется значением $00 и доступен для чтения и записи.

Разряд	7	6	5	4	3	2	1	0	
$0D ($2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR

По окончании передачи данных через интерфейс SPI аппаратная часть устанавливает в регистре состояния SPCR разряд SPIF (флаг прерываний от интерфейса SPI). Этот флаг указывает на завершение передачи, и приводит к запросу на прерывание как только в регистре управления SPCR будет установлен разряд **SPIE** (SPI Interrupt Enable — разрешение прерывание от SPI), а в регистре состояния SREG будет установлен разряд общего разрешения прерываний I.

Разряд **SPE** (**SP**I **E**nable — интерфейс SPI доступен) включает систему интерфейса (SPE = лог. 1) или выключает ее (SPE = лог. 0). После поступления сигнала сброса этот разряд возвращается в исходное состояние, и тем самым система SPI отключается.

Если разряд **DORD** (**D**ata **Ord**er) содержит лог. 0, то сначала будет передан старший разряд байта данных. При DORD = лог. 1 первым передается младший разряд.

Когда разряд **MSTR** (**Mast**er/Slave Select — выбор режима Master/Slave) содержит лог. 0, то система SPI определяется как ведомая (Slave), а при MSTR = лог. 1 она будет определена как ведущая (Master). Когда линия /SS в режиме Master сконфигурирована как вход, то разряд MSTR при низком уровне сигнала на выводе /SS сбрасывается в исходное состояние, и тем самым интерфейс SPI определяется как Slave. В этом случае в регистре состояния устанавливается флаг SPIF. Для того чтобы опять сконфигурировать интерфейс SPI как Master, пользователь должен установить разряд MSTR.

Когда разряд **CPOL** (**C**lock **Pol**arity — полярность тактового импульса) содержит лог. 0, то выход SCK в неактивном состоянии содержит сигнал низкого уровня. Если CPOL = лог. 1, то SCK в неактивном состоянии содержит сигнал высокого уровня. Подробности см. выше в разделе "Протокол передачи".

С помощью разряда **CPHA** (**C**lock **Pha**se Select — выбор фазы синхронизации) устанавливается один из двух основных режимов передачи данных. Более подробно эти вопросы освещены выше в разделе "Протоколы передачи".

Разряды **SPR1** и **SPR0** (**SP**I Clock **R**ate Select — выбор частоты тактовых импульсов) при работе интерфейса SPI в режиме Master служат для выбора тактовой частоты для линии SCK. Если система SPI сконфигурирована как Slave, то эти разряды не имеют никакого значения. Взаимосвязь между разрядами SPR1, SPR0 и частотой импульсов в линии SCK показана в табл. 7.3.

Таблица 7.3. Частота импульсов в линии SCK в зависимости от разрядов SPR1, SPR0 и такта системной синхронизации **Ф**

SPR1	*SPR0*	*Частота импульсов в линии SCK*
0	0	**Ф** / 4
0	1	**Ф** / 16
1	0	**Ф** / 64
1	1	**Ф** / 128

Регистр состояния SPSR

Регистр состояния SPSR интерфейса SPI расположен в области ввода/вывода по адресу $0E (по адресу $2E в RAM). После поступления сигнала сброса он инициализируется значением $00 и доступен только для чтения. В микроконтроллерах базовой серии семейства AVR, содержащих интерфейс SPI, используются только разряды 6 и 7. Остальные разряды компанией Atmel зарезервированы и всегда содержат лог. 0.

Разряд	7	6	5	4	3	2	1	0	
$0E ($2E)	SPIF	WCOL	–	–	–	–	–	–	SPSR

После окончания процесса передачи данных через интерфейс SPI аппаратная часть устанавливает разряд **SPIF** (**SP**I **I**nterrupt **F**lag — флаг прерываний от интерфейса SPI). Этот флаг указывает на завершение передачи и вызывает запрос на прерывание, как только в регистре управления SPCR будет установлен разряд SPIE, а в регистре состояния SREG — разряд общего разрешения прерываний I.

Когда линия /SS в режиме Master сконфигурирована как вход, то при низком уровне сигнала на выводе линии /SS также будет установлен флаг SPIF. Флаг SPIF сбрасывается автоматически аппаратной частью при выполнении подпрограммы обработки прерывания от интерфейса SPI. Альтернативно, сброс может быть выполнен вручную посредством считывания регистра состояния SPSR и последующего обращения к регистру данных интерфейса SPI.

Флаг **WCOL** (**W**rite **Col**lision — конфликт записи) в регистре состояния SPSR устанавливается в том случае, когда во время передачи данных через интерфейс SPI предпринимается попытка записи в регистр данных SPI, что приводит к разрушению только что переданного байта данных. По этой причине текущая передача данных доводится до завершения, а новый байт не записывается в сдвиговый регистр интерфейса SPI. Флаг WCOL должен быть сброшен пользователем вручную посредством считывания регистра состояния и последующего обращения к регистру данных интерфейса SPI.

Регистр данных SPDR

Регистр данных SPDR интерфейса SPI расположен в области ввода/вывода по адресу $0F (по адресу $2F в RAM). После поступления сигнала сброса он инициализируется значением $00 и доступен для чтения и записи.

Разряд	7	6	5	4	3	2	1	0	
$0F ($2F)	MSB							LSB	SPDR

Запись байта в регистр данных начинает передачу через интерфейс SPI. При считывании регистра данных возвращается тот байт, который находится в буфере приема интерфейса SPI.

Практические возможности применения периферийного последовательного интерфейса SPI наглядно показаны в некоторых примерах, рассмотренных в главе 16.

8 Последовательная передача данных по шине I²C

Интегральные микросхемы становятся все сложнее, а это неизбежно приводит к значительному увеличению числа соединительных контактов, определяющего габариты блоков. Из-за большого количества соединительных линий требуется столько же проводов и мест пайки, а при переносе на другую печатную плату — большое число разъемов. Это увеличивает стоимость системы, повышает вероятность возникновения сбоев в ее работе и ограничивает количество блоков, которые могут быть расположены на печатной плате. Еще одним последствием также является относительно высокая цена корпусов интегральных схем.

Однако в той высокой скорости, с которой может осуществляться передача данных через современную параллельную шину (32 разряда адреса, 32 разряда данных), зачастую нет необходимости. Перечислим некоторые примеры небольшой скорости передачи данных.

- Работа на клавиатуре. Даже при быстром вводе максимальная скорость составляет 5–10 знаков в секунду.
- Управление дисплеем. Вследствие инерционности человеческого глаза, нет смысла изменять показания чаще 2–3 раз в секунду.
- Обслуживание механического устройства. Механические системы очень инерционны (исполнительные механизмы или, например, системы регулирования числа оборотов).

Во всех этих случаях работа с устройствами связана с небольшими скоростями передачи данных. Кроме того, данные должны передаваться последовательно, побитно. Незначительная скорость передачи данных в данном случае не играет роли (для последовательного интерфейса SPI, описанного в предыдущей главе, скорость передачи составляет более одного миллиона бит в секунду, а для шины I²C — до 100000 бит в секунду). По этой причине компания Philips разработала шину I²C, которая также называется Inter-IC — то есть, шина для передачи данных между интегральными схемами (рис. 8.1).

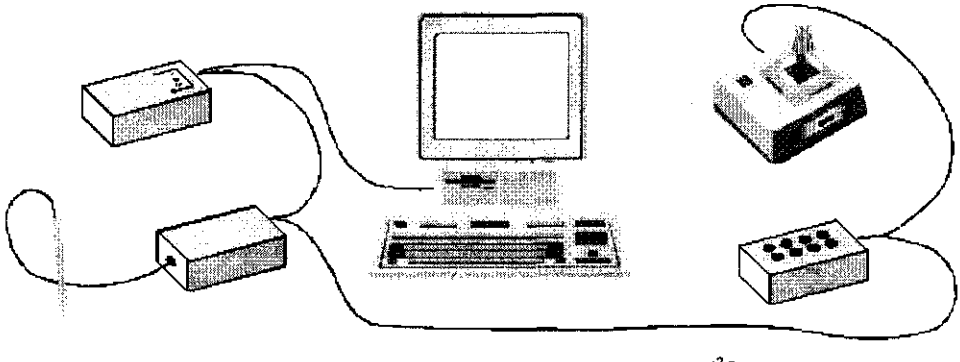


Рис. 8.1. Управление компонентами через шину I²C

Преимущества шины I^2C:

- интегральные схемы могут быть подсоединены к шине или отсоединены от нее без оказания какого-либо влияния на другие подключенные к шине интегральные схемы;
- нет необходимости в таких обычных схемах типа "шина–интерфейс" как драйвер и приемник линии, поскольку интерфейс шины I^2C уже интегрирован в кристалл;
- одна и та же схема шины типа I^2C зачастую может быть применена в самых разнообразных сферах;
- время разработки новых схем значительно сокращается, поскольку проектировщик быстро осваивается с функцией шины I^2C, а схемы с шиной этого типа просты в применении.

Технические характеристики шины I^2C:

- последовательная работа только с двумя проводами (см. рис. 8.1 и рис. 8.2), благодаря чему требуется меньше мест соединения и минимизируются затраты на проводку;
- зона действия — до 3 м;
- возможность работать в режиме с одним ведущим блоком (Single-Master) или с несколькими ведущими блоками (Multi-Master).

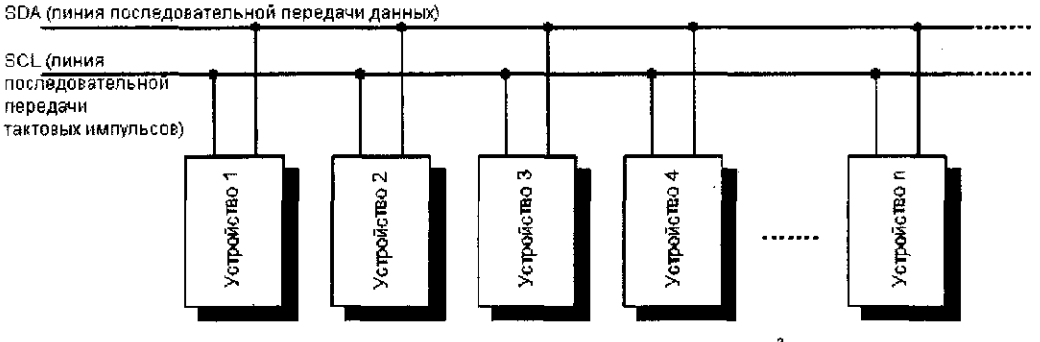


Рис. 8.2. Подсоединение отдельных модулей к шине I^2C

Принцип действия шины I^2C

Шина I^2C состоит всего из двух проводов:

- SCL (**S**erial **C**lock **L**ine) — линия последовательной передачи синхроимпульсов;
- SDA (**S**erial **D**ata **L**ine) — линия последовательной передачи данных.

Линия SCL

До тех пор, пока через шину не передаются никакие данные ("шина свободна"), линия SCL содержит высокий уровень сигнала.

Тактовая шина SCL выполнена как структура с открытым стоком/коллектором (рис. 8.3), благодаря чему существует возможность ее подключения как монтажное "И" (то есть, тактовая шина содержит низкий уровень сигнала, когда хотя бы один блок генерирует сигнал низкого уровня, и высокий уровень, когда все блоки генерируют сигнал высокого уровня). Тактовая шина соединена с питающим напряжением через подтягивающий резистор.

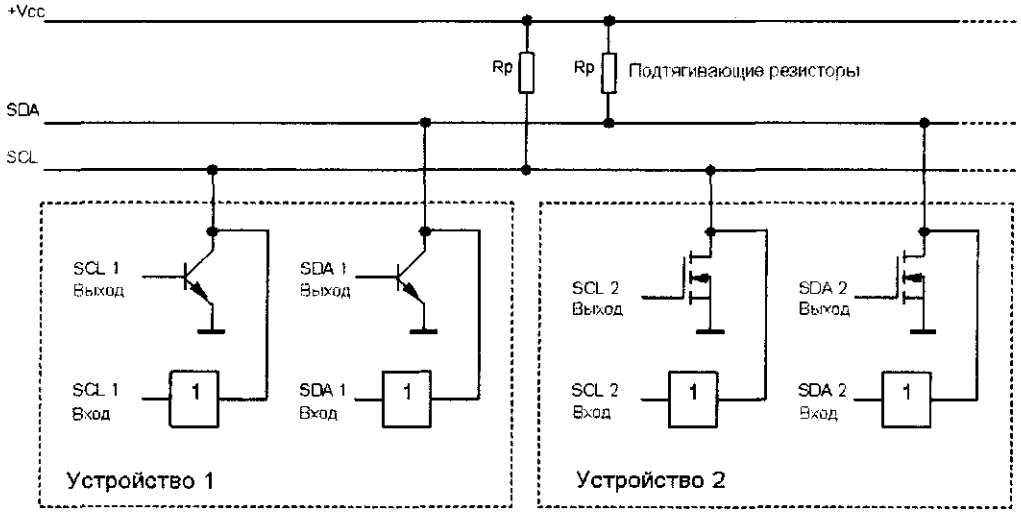


Рис. 8.3. Структура интегральной схемы I^2C и занятость шин I^2C

При питающем напряжении 5 В определены следующие уровни:

- сигнал низкого уровня — самое больше 1,5 В;
- сигнал высокого уровня — самое меньшее 3 В.

Блоки типа I^2C в случае сигнала низкого уровня имеют максимальное выходное напряжение 0,4 В и при этом в состоянии принимать ток силой до 3 мА.

Линия SDA

До тех пор, пока через шину не передаются данные, линия последовательной передачи данных SDA имеет высокий потенциал. Данные по линии SDA передаются последовательно. Они действительны в фазе высокого уровня такта и могут менять свое состояние только в фазе низкого уровня. Каждый блок, подсоединенный к шине, во время передачи данных может быть или приемником (ресивером), или передатчиком (трансмиттером).

Линия SDA, так же как и тактовая шина SCL, выполнена в виде структуры с открытым стоком/коллектором (см. рис. 8.3), благодаря чему ее можно подключать как монтажное "И" (то есть, линия передачи данных содержит сигнал низкого уровня, когда по хотя бы один блок генерирует сигнал низкого уровня, и сигнал высокого уровня, когда все блоки генерируют сигналы высокого уровня). Линия соединена с питающим напряжением через подтягивающий резистор. Уровень напряжения определяется так же, как для тактовой шины SCL.

Режимы работы блоков, подсоединенных с помощью шины I^2C

Блок, подсоединенный к шине I^2C, может действовать как приемник или передатчик. В режиме приемника блок принимает сигналы, переданные на линию SDA и данные по линии SCL, синхронизированные с помощью тактового сигнала. В режиме передатчика блок через линию SDA передает на линию SCL свои данные, синхронизованные с помощью тактового сигнала. Является ли блок передатчиком или приемником, зависит только от его особых свойств. Например, запоминающее устройство ROM может быть только передатчиком, в то время как светодиодный драйвер, наоборот, всегда является приемником. В то же время, запоминающее устройство RAM или микропроцессор могут быть как передатчиками, так и приемниками.

Режим работы блока может быть изменен прямо во время передачи данных. Например, возможен случай, когда микропроцессор сначала передает данные (передатчик) запоминающему устройству RAM (приемник). При обращении на выборку данных этот же процессор выступает в качестве приемника, считывая данные из RAM (передатчик).

Каждый блок, подсоединенный к шине, во время передачи данных может быть ведущим (Master) или ведомым (Slave). Ведущее устройство инициирует передачу данных. В частности, Master занимает шину тем, что генерирует стартовый сигнал на линии SCL и начинает обмен с ведомым устройством. До тех пор, пока на шине работает ведущее устройство, не может быть активным никакой другой Master-блок.

Тактовый сигнал на линии SCL может быть сформирован только ведущим устройством.

Ведомым является устройство, по адресу которого обращается ведущее устройство с требованием передачи данных. Теоретически Master может одновременно снабжать одними и теми же данными несколько ведомых устройств.

В роли ведущего устройства в большинстве случаев выступает микропроцессор, оборудованный или специальной аппаратной частью с шиной типа I^2C, или контроллером шины, работающим под управлением специального программного обеспечения.

В зависимости от того, каким образом должна происходить передача данных через шину I^2C, различают следующие случаи:

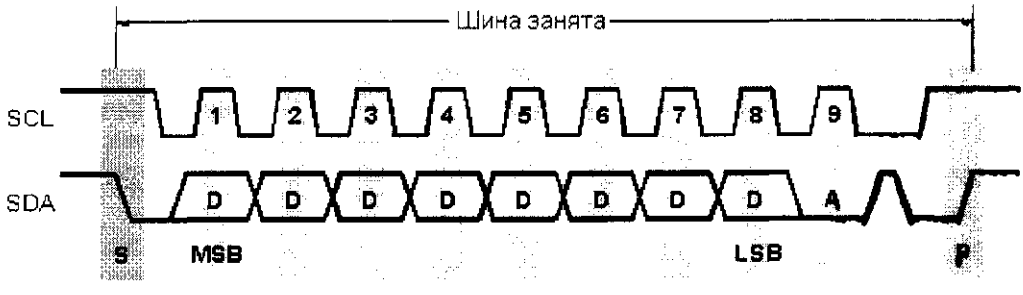
- **Master-передатчик** — ведущее устройство, которое передает данные ведомому устройству (приемнику);
- **Master-приемник** — ведущее устройство, которое принимает данные от ведомого устройства (передатчика);
- **Slave-передатчик** — ведомое устройство, которое передает данные ведущему устройству (приемнику);
- **Slave-приемник** — ведомое устройство, которое принимает данные от ведущего устройства (передатчика).

Электрические свойства

Скорость передачи данных определяется посредством тактового сигнала в линии SCL. В качестве верхней границы для такта установлена частота 100 КГц, а нижней границы не существует. Каждый блок, подключенный к шине, должен быть в состоянии придерживаться тактовой частоты. Несмотря на то, что соотношение тактов установлено как 1:1, это — необязательное требование. Более важным определяющим фактором для безошибочной передачи данных является крутизна фронта сигнала: время спадания как для данных, так и для такта не должно превышать 300 нс. Именно здесь заключается ограничение для шины: емкостная нагрузка линий не должна превышать 400 пФ! В связи с тем, что емкостная нагрузка зависит как от числа блоков, подсоединенных к шине, так и от длины проводов, это значение должно учитываться при проектировании аппаратного обеспечения. Согласно спецификациям, существуют также режимы работы шины I^2C с низкой и высокой скоростью, но здесь будут рассмотрен только стандартный режим работы.

Протокол шины

Для того чтобы несколько блоков могли обмениваться данными, необходим некоторый протокол, который описывает процесс передачи данных по шине и в любой момент времени не допускает ошибочной интерпретации состояния шины. В случае шины с несколькими ведущими устройствами (что относится и к I^2C) необходимо также установить, когда и какое ведущее устройство имеет право занимать шину. Эти условия регулируются протоколом шины (рис. 8.4).



S: Условие начала
P: Условие завершения
D: Бит данных (не может быть изменен, когда SCL = выс. ур.)
A: Подтверждение (от приемника о получении байта)
MSB: Старший значащий бит
LSB: Младший значащий бит

Рис. 8.4. Протокол передачи байта данных через шину I^2C

Занятость шины

Работа шины и, следовательно, ее занятость, определяется условием начала передачи.

Условие начала передачи

Линии SCL и SDA находятся в состоянии высокого уровня (то есть, в состоянии покоя). Ведущее устройство, которое хочет начать передачу данных, изменяет состояние линии SDA к низкому уровню (рис. 8.5).

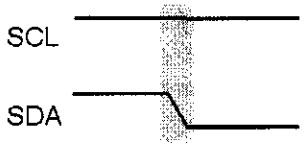


Рис. 8.5. Условие начала передачи данных

После наступления условия начала передачи шина будет занята ведущим устройством, создавшим это условие, вследствие чего все другие ведущие устройства будут заблокированы.

Условие начала является однозначным состоянием на шине, потому что смена уровня сигнала на линии SDA, как правило, допускается только тогда, когда тактовая линия SCL находится в состоянии низкого уровня.

Все устройства, подключенные к шине, должны распознать условие начала передачи и переключаются на прием (ведомое устройство/приемник). Условие начала передачи также учитывается и другим ведущим устройством, которое со своей стороны имело намерение занять шину. В результате оно немедленно отзывает свое требование. Устройство, создавшее условие начала передачи, в данный момент времени является для шины ведущим (Master), а все остальные устройства — потенциальными ведомыми (Slave). Ведущее устройство теперь отвечает за тактовый сигнал и становится передатчиком.

Процесс передачи данных

После создания условия начала ведущее устройство начинает передачу данных. Оно переводит тактовую шину в состояние низкого уровня и теперь может занять линию передачи данных затребованным информационным разрядом (высокий или низкий уровень в линии SDA). Затем тактовая шина опять переводится в состояние высокого уровня (см. рис. 8.4).

Изменение в линии SDA может происходить только в фазе низкого уровня сигнала в линии SCL. Во время фазы высокого уровня в линии SCL линия SDA должна быть стабильной.

Передача данных, как правило, выполняется побайтно, при этом первым передается старший значащий бит (MSB). После передачи полного байта данных, состоящей из восьми тактовых циклов, следует бит подтверждения от приемника.

Бит подтверждения

Бит подтверждения (acknowledge) — это реакция приемника на принятый байт данных. Он является для передатчика признаком того, что приемник физически присутствует и "прослушивает" линию. Одновременно с этим бит подтверждения можно рассматривать как сигнал синхронизации.

Бит подтверждения, как правило, генерируется приемником. Если ведущее устройство принимает от ведомого устройства несколько байтов данных, то оно квитирует каждый отдельный байт битом подтверждения, за исключением последнего. Такое отрицательное квитирование сообщает ведомому устройству, что передача данных завершена, и далее последует условие завершения или новое условие начала передачи.

Для передачи бита подтверждения ведущее устройство генерирует на линии SCL дополнительный тактовый импульс (рис. 8.6). Приемник выдает сигнал положительного квитирования, переводя линию SDA в состояние низкого уровня, или отрицательного квитирования, переводя линию SDA в состояние высокого уровня.

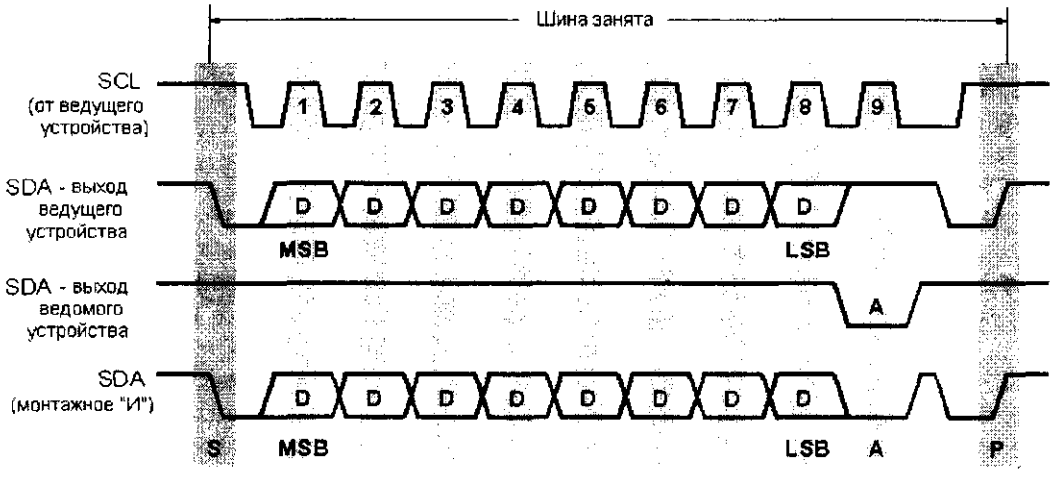


Рис. 8.6. Выработка бита квитирования приемником, работающим в режиме ведомого устройства

Таким образом, для передачи одного байта данных, как правило, требуется девять тактовых циклов, вследствие чего при тактовой частоте 100 КГц возможна скорость передачи данных максимум 11111 байт/с.

После передачи одного байта данных и приема бита подтверждения передача данных может быть сразу же продолжена. Если приемник реагирует на передачу байта данных отрицательным квитированием, то ведущее устройство должно завершить передачу данных, опять освободив шину.

Освобождение шины передачи данных

Шина после окончания передачи данных, которая может состоять из любого количества байтов, опять освобождается ведущим устройством. Освобождение шины осуществляется с помощью условия завершения.

Условие завершения передачи

Изменение в линии SDA уровня сигнала с низкого на высокий в то время, когда по тактовой шине SCL передается сигнал высокого уровня, оценивается как условие завершения передачи (рис. 8.7).

Рис. 8.7. Условие завершения передачи

После создания условия завершения передачи шина освобождается, то есть, опять становится доступной для всех блоков и устройств. Условие завершения передачи также представляет собой однозначное состояние на шине. Все блоки и устройства распознают его и подготавливаются к появлению нового условия начала передачи. В том случае, если ведущее устройство из-за промежуточной занятости шины отзывает свое требование занять шину, то оно может предпринять новую попытку создать условие начала передачи и тем самым получить шину для своих нужд.

Полная передача данных через шину I^2C, в принципе, состоит из условия начала передачи, одного или нескольких байтов данных (за которыми, соответственно, следует бит квитирования), и условия завершения передачи.

Адресация ведомых устройств

Выбор ведомого устройства, с которым хотело бы обмениваться данными ведущее устройство, осуществляется посредством первого байта, который всегда определяется как адрес ведомого устройства.

Адрес ведомого устройства

Первый байт последовательности данных представляет собой адрес ведомого устройства. Он однозначно сопоставлен определенному устройству, подключенному к шине, и имеет длину 7 бит (биты от 1 до 7). Теоретически, таким образом можно адресовать до 128 ведомых устройств, однако по определению некоторые адреса ведомых устройств имеют особое значение. В действительности система состоит не более, чем из 20 ведомых блоков. Адрес ведомого устройства состоит из двух частей: постоянной и переменной (рис. 8.8).

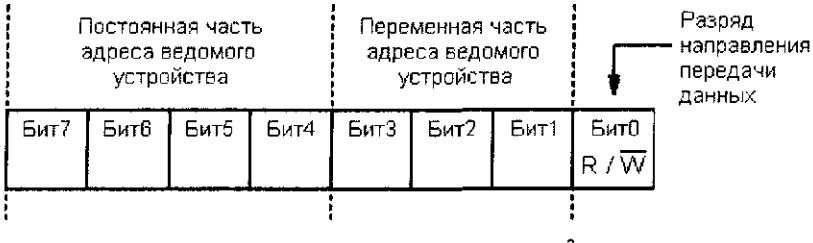


Рис. 8.8. Формат адреса шины I^2C

Постоянная часть адреса ведомого устройства

Постоянная часть адреса описывает требования к определенным группам устройств и определяется изготовителем кристалла. Его длина определена в результате практического опыта и в большинстве случаев составляет 4 бита (см. рис. 8.8). Он будет тем короче, чем больше однотипных устройств в схеме. Постоянная часть адреса жестко "прошита" в интегральной схеме и не может быть изменена пользователем.

Переменная часть адреса ведомого устройства

Переменная часть адреса ведомого устройства (см. рис. 8.8) служит для выбора определенного устройства из группы однотипных кристаллов, среди которых все имеют постоянную часть адреса ведомого устройства. Благодаря этому, к шине могут быть подсоединены несколько однотипных интегральных схем. Переменная часть в большинстве случаев определяется пользователем с помощью внешних схем (через дополнительные выводы).

Разряд направления передачи данных

С помощью первых семи разрядов (7–1) первого байта данных после создания условия начала передачи (то есть, адреса ведомого устройства) однозначно идентифицируется требуемый ведомый блок. Восьмой передающийся разряд (разряд 0 — см рис. 8.8) задает направление передачи данных. Он определяет, должны ли быть приняты или переданы данные.

Если разряд направления передачи данных содержит "1" (чтение), то ведущее устройство находится в режиме приемника, а ведомое — в режиме передатчика. Если разряд направления передачи данных содержит "0" (запись), то ведущее устройство будет работать как передатчик, а ведомое — как приемник.

Рабочий режим, выбранный с помощью адреса ведомого устройства, действителен для последовательности байтов данных любой длины. В связи с тем, что только одно ведущее устройство определяет, сколько байтов данных будет передано, ведомые устройства в большинстве случаев используют автоинкремент при внутренней адресации. Таким образом, например, можно передать любой объем данных в память RAM, адресованную как ведомое устройство-приемник. При этом данные сохраняются в адресах, следующих один за другим.

С таким же успехом можно последовательно считывать данные из памяти RAM, когда она адресована ведущим устройством как ведомый передатчик, а тактирование осуществляется в соответствии с количеством считываемых байтов данных.

Адрес ведомого устройства также подтверждается этим устройством с помощью бита квитирования. Если ведущее устройство после адресации получает отрицательное квитирование, то оно может заключить, что ведомое устройство или вообще отсутствует, или в настоящий момент с ним невозможно установить связь (например, оно занято обработкой заданий, критическими с точки зрения времени).

Примеры обмена данными между Master и Slave представлены на рис. 8.9.

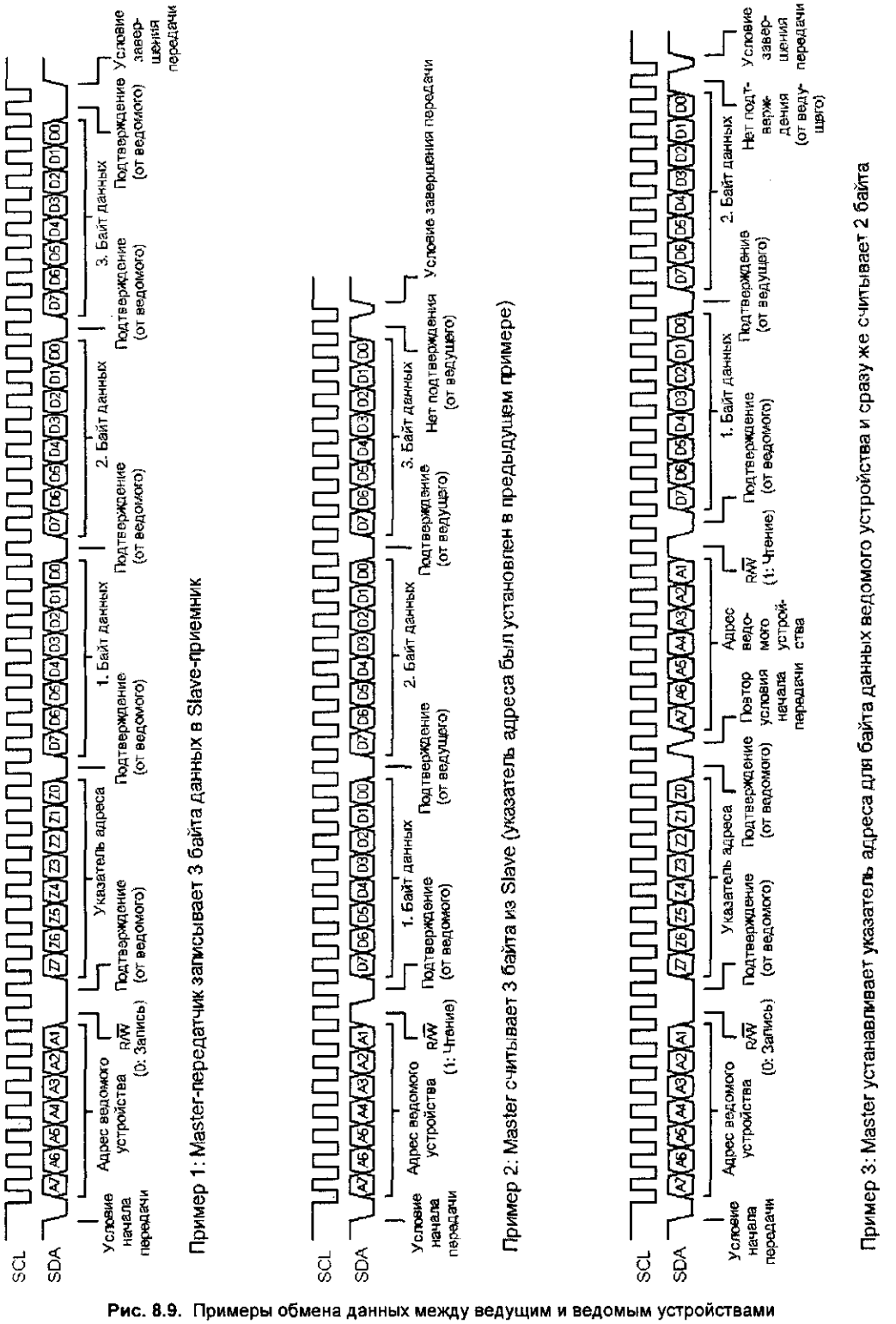


Рис. 8.9. Примеры обмена данных между ведущим и ведомым устройствами

Особые случаи

При передаче данных могут возникнуть некоторые особые ситуации, которые должны быть корректно обработаны аппаратным обеспечением шины I^2C.

Повторение условия начала передачи

В зависимости от условий, возникающих после передачи полного байта данных (включая бит подтверждения), ведущее устройство может повторить условие начала передачи. Для подсоединенного ведомого устройства это чревато тем, что оно отказывается от передачи данных и больше не адресуется, хотя шина остается занятой, а ведущее устройство в состоянии передать новый адрес ведомого устройства.

После возникновения условия начала передачи всегда должен следовать адрес ведомого устройства.

Это может относиться как к тому же ведомому устройству, так и к другим. Таким образом, можно обслуживать или несколько ведомых устройств одно за другим, или для одного ведомого устройства только изменить направление передачи данных на обратное. В этом случае есть одно преимущество: в течение рассматриваемого отрезка времени в работу шины не может вмешаться никакое другое ведущее устройство.

Чтение–модификация–запись

Рассмотрим следующую ситуацию. Байт данных считывается извне без перерыва, меняет свое значение, а затем опять записывается. Еще одно устройство, которое хотело бы обратиться к этому же байту данных, "видит" не отдельные шаги, а только результат.

Исследуем этот процесс на конкретном примере. Пусть ведущее устройство обращается на чтение к блоку памяти RAM с постоянным адресом $10_d = A_h = 1010_b$ и переменным адресом $7_d = 111_b$ ведомого устройства, желая считать байт данных, хранящийся по адресу 20_d, изменить его и опять записать.

Для этого необходимо выполнить следующие действия.

1. Указатель адреса в памяти RAM устанавливается на $20_d = 14_h = 0001\ 0100_b$. Для этого ведущее устройство создает условие начала передачи, адресует ведомое устройство как приемник и передает ему значение указателя адреса:
   - условие начала передачи;
   - первый байт: адрес ведомого устройства + бит "Запись" = $1010\ 1110_b$;
   - подтверждение от ведомого устройства;
   - второй байт: указатель адреса = $0001\ 0100_b$;
   - подтверждение от ведомого устройства.
2. Вслед за этим выполняется считывание байта данных по адресу 20_d:
   - повторение условия начала передачи;
   - первый байт: адрес ведомого устройства + бит "Чтение" = $1010\ 1111_b$;

- подтверждение от ведомого устройства;
- второй байт: считывается, например, байт данных $F0_h$ ($1111\ 0000_b$) по адресу 20_d;
- нет подтверждения от ведущего устройства (окончание операции чтения).

3. Ведущее устройство теперь изменяет считанный байт данных и хотело бы после этого опять записать результат в память RAM по адресу 20_d. После выполнения операции чтения в действии 2, благодаря функции автоматического инкремента памяти RAM, внутренний адрес увеличен до 21_d. По этой причине ведущее устройство перед повторной записью байта данных должно выполнить сброс указателя адреса RAM в 20_d:
   - изменение байта данных с $F0_h$ на $0F_h$;
   - повторяется условие начала передачи;
   - первый байт: адрес ведомого устройства + бит "Запись" = $1010\ 1110_b$;
   - подтверждение от ведомого устройства;
   - второй байт: указатель адреса $0001\ 0100_b$;
   - подтверждение от ведомого устройства;
   - третий байт: измененный байт данных 0000 1111 b по адресу 20_d;
   - подтверждение от ведомого устройства.
4. Ведущее устройство теперь может или разблокировать шину, создав условие завершения передачи, или продолжить процесс дальше, опять повторив условие начала передачи с последующей адресацией.

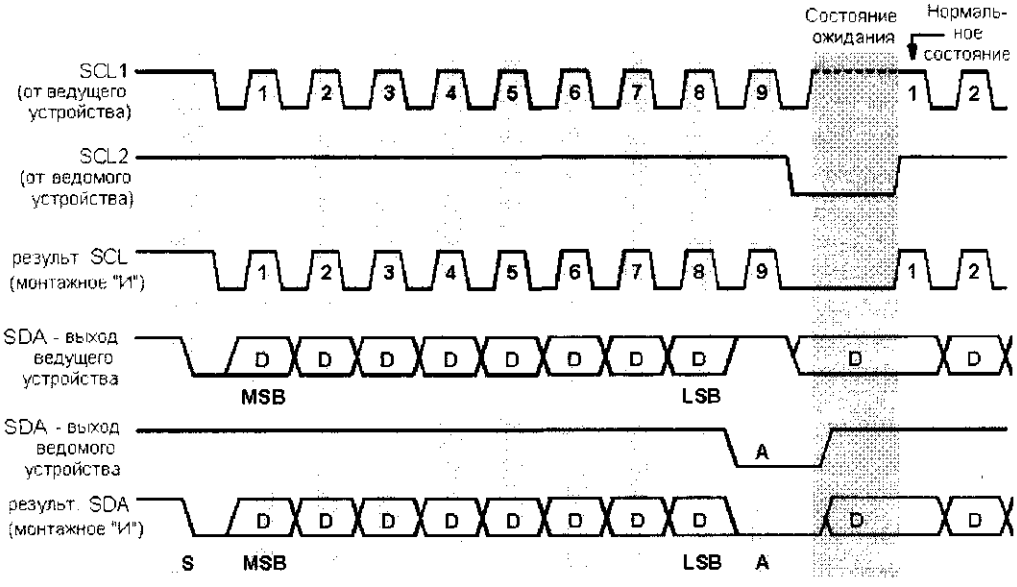
Задержка такта

Непосредственно после передачи разряда подтверждения следует фаза низкого уровня тактирующего сигнала. Приемник независимо от того, является ли он ведущим или ведомым устройством, вследствие открытого коллектора может перевести тактовую шину в состояние низкого уровня и удержать ее в этом состоянии. В результате передатчик будет переведен в состояние ожидания. Он прерывает передачу до тех пор, пока тактовая шина SCL опять не окажется в состоянии высокого уровня. Так образуется особый вид взаимообмена, называемый "Handshaking" — "рукопожатие" (обмен с подтверждением связи), который позволяет обрабатывать принятые данные даже медленнодействующему приемнику (рис. 8.10).

Синхронизация такта

Наряду с задержкой такта после передачи полного байта данных дополнительно может выполняться синхронизация таким образом, что приемник "растягивает" период низкого уровня тактирующего сигнала. Эта особенность, например, полезна в том случае, когда приемником является микропроцессор, обслуживающий шину I^2C с помощью программного обеспечения, который после приема должен обрабатывать каждый бит в отдельности. В таком случае такт будет синхронизирован с тактом ведущего устройства, в результате чего будет получен ре-

зультирующий такт, фаза низкого уровня которого имеет длину, соответствующую самому медленнодействующему, а фаза высокого уровня — самому быстродействующему устройству (рис. 8.11).



S: Условие начала

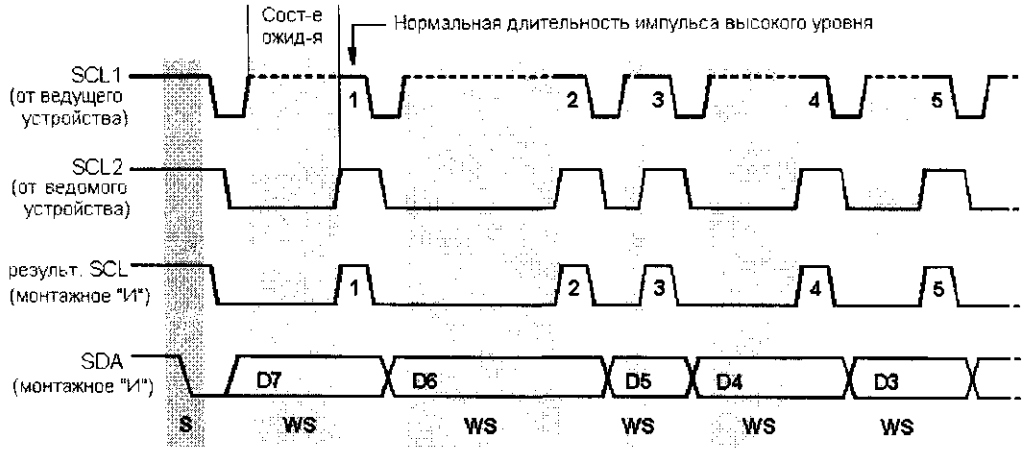
D: Бит данных (не может быть изменен, когда SCL = выс. ур.)

A: Подтверждение (от приемника о получении байта)

MSB: Старший значащий бит

LSB: Младший значащий бит

Рис. 8.10. Вставка состояния ожидания при передаче данных от ведущего устройства к ведомому



S: Условие начала

WS: Состояние ожидания, выработанное ведомым устройством

Рис. 8.11. Уменьшение скорости передачи данных посредством задержки такта

Арбитраж шины при работе с несколькими ведущими устройствами

Как видно из рис. 8.3, магистральные шины, наряду с выходными драйверами, имеют также и входной каскад, в котором устройство сравнивает сгенерированное внутреннее состояние с фактически присутствующим на шине. Эта особенность используется для арбитража шины (предоставление доступа к общей шине в режиме работы нескольких ведущих устройств).

В связи с тем, что шина I^2C занимается асинхронно (то есть, не в точно установленный момент времени), может возникнуть случай, когда два или несколько ведущих устройств в один и тот же момент времени создают условие начала передачи. Каждое ведущее устройство на основании сигнала обратной связи через входной каскад создает впечатление, что именно оно является единственным передатчиком в данный момент времени. В связи с тем, что для тактового сигнала выполняется синхронизация, то это "впечатление" остается до тех пор, пока все передатчики передают в точности одну и ту же битовую комбинацию. Разногласие возникает только тогда, когда ведущее устройство, ожидающее появления на шине сигнала высокого уровня, вместо него обнаруживает сигнал низкого уровня. С этого момента данное ведущее устройство отзывает свою передачу данных, а другое или все остальные ведущие устройства по прежнему работают с шиной, поскольку не замечают возникшей проблемы. Таким образом ведущие устройства одно за другим освобождают шину до тех пор, пока наконец останется единственное ведущее устройство, которое "выиграло" процесс арбитража.

Легко заметить, что из-за открытого коллектора арбитраж всегда выигрывает то ведущее устройство, которое последовательно передает больше сигналов низкого уровня (то есть, нулевые биты). В связи с тем, что первый подлежащий передаче байт данных — это адрес ведомого устройства, решение в процессе арбитража принимается еще при адресации, если к ведомому устройству хотят обратиться несколько ведущих устройств. Для этого случая обращение для записи (бит направления передачи данных = 0) опять имеет приоритет перед обращением для чтения. Таким образом, наивысшим приоритетом обладает адрес 0 ведомого устройства при обращении для записи ($0000\ 0000_b$); обозначаемый понятием "общий вызов".

Также может возникнуть особый случай, когда два ведущих устройства одновременно адресуют одно и то же ведомое устройство, передают ему те же самые данные и совместно генерируют условие завершения передачи. При этом ни одно ведущее устройство "не замечает" каких-либо действий другого ведущего устройства.

Если ведущее устройство проигрывает арбитраж еще при выдаче адреса ведомого устройства, то оно немедленно переключается в режим работы ведомого приемника и считывает дальше данные с шины. Благодаря этому, устройство в состоянии отреагировать в том же цикле, когда оставшееся на шине ведущее устройство обращается по его адресу.

Задача пользователя — выбрать действия, необходимые после проигрыша арбитража. Обычно после разблокирования шины повторяется весь процесс передачи данных, начиная с первого условия начала передачи.

Синхронизация шины I^2C

В связи с тем, что в случае I^2C речь идет о синхронной шине, к ее синхронизации не предъявляется каких-либо особых требований, кроме одного: не должны быть превышены минимальные значения времени T_{High} = 4 мкс для сигнала высокого уровня и T_{Low} = 4,7 мкс для сигнала низкого уровня для тактирующих сигналов в линии SCL.

Исчерпывающее описание характеристик и параметров шины I^2C можно найти на Web-сайте компании Philips по адресу, указанному в конце книги.

Обращение микроконтроллеров семейства AVR к шине I^2C

В отличие от последовательного интерфейса SPI, микроконтроллеры семейства AVR не оснащены аппаратным интерфейсом для обращения к шине I^2C. Однако, благодаря высокому допустимому значению тактовой частоты, микроконтроллеры семейства AVR могут взаимодействовать с шиной I^2C с помощью программного обеспечения, что показано в главе 16.

9 Интегрированный аналоговый компаратор

Интегрированный аналоговый компаратор используется во всех микроконтроллерах базовой серии семейства AVR. Он сравнивает входное напряжение на своем неинвентирующем входе AIN0 с входным напряжением на инвентирующем входе AIN1. Как только напряжение на неинвентирующем входе AIN0 станет больше, чем на инвентирующем AIN1, на выходе ACO устанавливается лог. 1.

Входные порты входов AIN0 и AIN1 для четырех представителей базовой серии семейства AVR перечислены в табл. 9.1.

Таблица 9.1. Входные порты входов AIN0 и AIN1 для четырех представителей базовой серии AVR

Входы компаратора	*AT90S1200*	*AT90S2313*	*AT90S4414*	*AT90S8515*
AIN0	Порт B / Разряд 0	Порт B / Разряд 0	Порт B / Разряд 2	Порт B / Разряд 2
AIN1	Порт B / Разряд 1	Порт B / Разряд 1	Порт B / Разряд 3	Порт B / Разряд 3

Выходной сигнал компаратора может быть применен для срабатывания функции захвата на входе T/C1. Выход ACO компаратора через входной мультиплексор Mux1 напрямую связан со схемой подавления сигналов помех функции захвата T/C1, когда управляющий разряд ACIC регистра управления и состояния ACSR в аналоговом компараторе содержит лог. 1 (см. рис. 4.7). Кроме того, через аналоговый компаратор возможен вызов прерывания.

С помощью разрядов ACIS1 и ACIS0 регистра ACSR аналогового компаратора можно выбрать способ вызова прерывания на выходе ACO компаратора: по ниспадающему/нарастающему фронту сигнала или при смене уровня сигнала. Блок-схема аналогового компаратора микроконтроллеров семейства AVR показана на рис. 9.1.

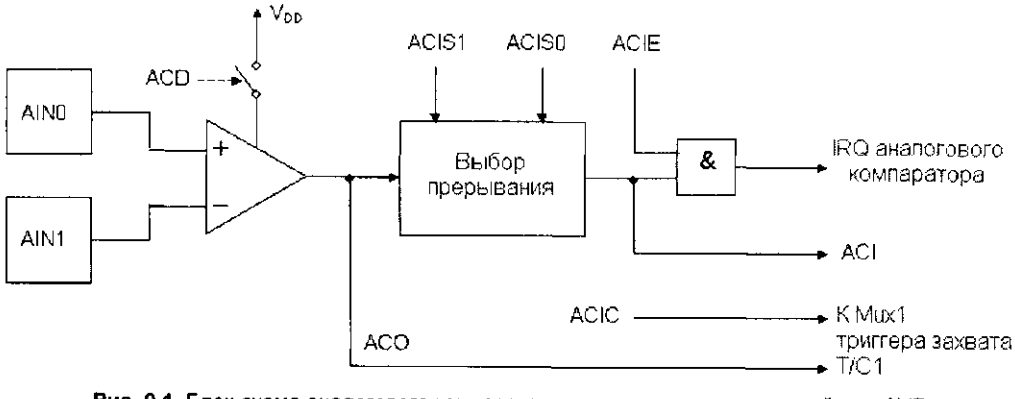


Рис. 9.1. Блок-схема аналогового компаратора микроконтроллеров семейства AVR

Регистр управления и состояния ACSR

Регистр управления и состояния ACSR (**C**ontrol and **S**tatus **R**egister) аналогового компаратора расположен в области ввода/вывода по адресу $08 (по адресу $28 в RAM). После поступления сигнала сброса он инициализируется значением $00. В моделях AT90S8515, AT90S4414 и AT90S2313 используются только разряды 0-5 и разряд 7 регистра ACSR. Разряды 0–4 доступны для чтения и записи. Разряд 5 (выход ACO компаратора) можно только считывать. Разряд 6 зарезервирован компанией Atmel и также доступен только для чтения (всегда содержит лог. 0).

В микроконтроллере AT901200 используются только разряды 0, 1, 3–5 и разряд 7 регистра ACSR. Разряды 0, 1, 3 и 4 доступны для чтения и записи. Разряду 5 соответствует выход ACO компаратора. Разряды 6 и 2 зарезервированы компанией Atmel и доступны только для чтения (всегда содержат лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$08 ($28)	ACD	–	ACO	ACI	ACIE	ACIC *)	ACIS1	ACIS0	ACSR

*) Отсутствует в микроконтроллере AT90S1200

Когда разряд **ACD** (**A**nalog **C**omparator **D**isable — аналоговый компаратор отключен) установлен в лог. 1, питание аналогового компаратора отключено. Это обычно применяют для того, чтобы еще больше снизить потребление тока в "спящем" режиме, когда нет необходимости в возврате через прерывание аналогового компаратора (см. также раздел "Спящие" режимы центрального процессора" главы 3).

Когда должен быть изменен разряд ACD, то вначале необходимо заблокировать прерывание от аналогового компаратора посредством сброса разряда разрешения ACIE в регистре ACSR, поскольку в противном случае в результате изменения разряда ACD непреднамеренно может быть вызвано прерывание.

Разряд **ACO** (**A**nalog **C**omparator **O**utput — выход аналогового компаратора) напрямую связан с выходом аналогового компаратора.

Флаг **ACI** (**A**nalog **C**omparator **I**nterrupt Flag — флаг прерываний аналогового компаратора) устанавливается в лог. 1, когда наступает событие, определенное разрядами ACIS1 и ACIS0 в регистре ACSR и, вследствие этого, вызывает прерывание от аналогового компаратора.

В том случае, когда в регистре состояния установлен флаг общего разрешения прерываний, а в регистре ACSR — разряд ACIE, в результате установки флага ACI выполнение программы разветвляется по соответствующим адресам обработки прерываний от аналогового компаратора (адрес $00C для микроконтроллеров AT90S8515 и AT90S4414, адрес $00A для микроконтроллера AT90S2313, адрес $003 для микроконтроллера AT90S1200). Как только начинает выполняться подпрограмма обработки прерывания, может быть выполнен аппаратный сброс флага ACI. Альтернативно этому, флаг ACI может быть сброшен посредством записи лог. 1 в разряд 4 регистра ACSR.

Когда разряд **ACIE** (**A**nalog **C**omparator **I**nterrupt **E**nable — разрешение прерывания от аналогового компаратора) и разряд общего разрешения прерываний в ре-

гистре состояния SREG установлены в лог. 1, то аналоговый компаратор разблокирован. В противном случае он будет заблокирован.

Когда разряд **ACIC** (**A**nalog **C**omparator **I**nput **C**apture Enable — разрешение захвата на входе аналогового компаратора) установлен в лог. 1, то выход аналогового компаратора будет соединен через мультиплексор Mux1 со входом схемы подавления помех при захвате на входе таймера-счетчика T/C1 (см. рис. 4.7). Если разряд ACIC сброшен, то мультиплексор Mux1 соединяет вывод порта ICP со входом схемы подавления помех.

Для того чтобы можно было вызвать прерывание захвата, оно должно быть разрешено разрядом TICIE1 в регистре TIMSK (см. раздел "Сброс и обработка прерываний" главы 3).

Разряды **ACIS1** и **ACIS0** устанавливают вид события на выходе аналогового компаратора, которое должно вызвать прерывание работы аналогового компаратора. Возможные значения разрядов ACIS1 и ACIS0 показаны в табл. 9.2.

Таблица 9.2. Вид событий для вызова прерывания аналогового компаратора

Разряд ACIS1	*Разряд ACIS0*	*Вид прерывания аналогового компаратора*
0	0	Прерывание при изменении состояния выхода
0	1	Зарезервировано
1	0	Прерывание по ниспадающему фронту на выходе аналогового компаратора
1	1	Прерывание по нарастающему фронту на выходе аналогового компаратора

10 Порты ввода/вывода

Использование портов ввода/вывода в различных микроконтроллерах базовой серии семейства AVR показано в табл. 10.1

Таблица 10.1. Использование портов ввода/вывода в различных моделях базовой серии AVR

Порт	*AT90S1200*	*AT90S2313*	*AT90S4414*	*AT90S8515*
Порт A	–	–	Есть	Есть
Порт B	Есть	Есть	Есть	Есть
Порт C	–	–	Есть	Есть
Порт D	Есть (7 разрядов)	Есть (7 разрядов)	Есть	Есть

Все порты ввода/вывода микроконтроллеров базовой серии семейства AVR являются 8-разрядными и двунаправленными. Исключение составляет только 7-разрядный порт D моделей AT90S1200 и AT90S2313. Здесь пришлось пренебречь разрядом PD7, поскольку в распоряжении есть только 20 точек подсоединения к корпусу. Каждый вывод порта может быть индивидуально сконфигурирован как вход или выход, причем в случае функционировании в качестве входа к нему может быть по выбору подключено подтягивающее сопротивление (рис. 10.1).

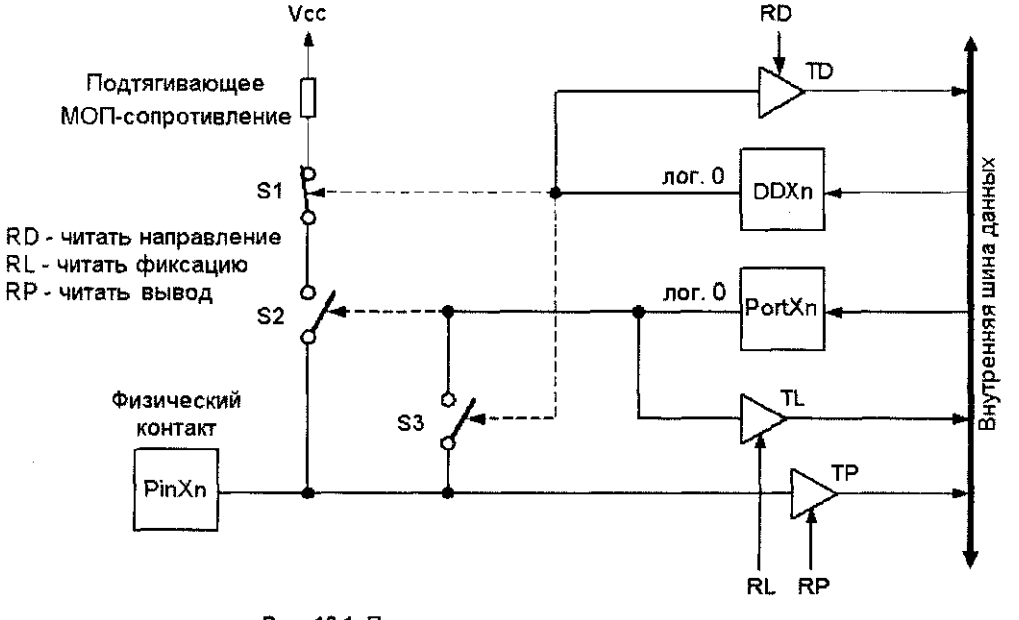


Рис. 10.1. Принципиальная схема вывода порта

На рис. 10.1 показана схема, по которой построены порты микроконтроллеров семейства AVR. Буква "X" в обозначении вывода регистра порта и регистра направления передачи данных используется вместо обозначения порта, то есть, вме-

сто нее можно поставить буквы "A", "B", "C" или "D". Номера разрядов 0–7 внутри регистра представлены буквой "n".

Положения ключей S1–S3, показанные на рис. 10.1, соответствуют сигналам низкого уровня на выходах регистров DDXn и PortXn.

Обращение к каждому из портов осуществляется через по трем различным адресам в области ввода/вывода:

1. Регистр направления передачи данных (DDX на рис. 10.1) определяет назначение вывода порта: вход или выход.
   - Если разряд "n" в регистре DDXn содержит сигнал низкого уровня, то соответствующий вывод сконфигурирован как вход. Ключ S3 (замыкающий контакт) в этом случае разомкнут и отделяет регистр PortXn от вывода PinXn. Ключ S1 (размыкающий контакт) при этом закрыт и подключает подтягивающее сопротивление к выводу PinXn, если ключ S2 замкнут посредством лог. 1 на выходе регистра PortXn.
   - Если разряд "n" в регистре DDXn содержит сигнал высокого уровня, то соответствующий вывод сконфигурирован как выход. Ключ S3 (замыкающий контакт) в этом случае замкнут и отделяет регистр PortXn от вывода PinXn. Ключ S1 (размыкающий контакт) при этом разомкнут и отделяет подтягивающее сопротивление от вывода PinXn, независимо от положения ключа S2 и значения на выходе порта PortXn.
2. В регистр порта PortX (см. рис. 10.1) в том случае, если вывод выполняет роль выхода, записывается значение, подлежащее выводу. В том случае, когда вывод выполняет роль входа, ключ S2 замкнут, и подтягивающее сопротивление может быть подключено к входному контакту.
3. Логический уровень на выводе PinXn может быть считан по третьему адресу. Схема управления в этом случае вырабатывает сигнал считывания RP (**R**ead **P**in — читать вывод), который через драйвер TP подключает вывод непосредственно к внутренней шине данных.

Все возможные конфигурации для направления передачи данных и подтягивающего сопротивления порта перечислены в табл. 10.2.

Таблица 10.2. Конфигурация направления передачи данных и подтягивающего сопротивления порта

Регистр DDXn	*Регистр PortXn*	*Ввод/вывод*	*Подтягивающее сопротивление*	*Описание*
0	0	Ввод	Нет	Высокоомный вход
0	1	Ввод	Да	Подтяг. сопротивление нагружает вход
1	0	Вывод	Нет	Двухтактный выход: лог. 0
1	1	Вывод	Нет	Двухтактный выход: лог. 1

Регистр направления передачи данных и регистр порта доступны для чтения и записи, а информация на выводах аппаратного обеспечения может быть только считана. Для того чтобы выполнить операцию чтения, схема управления выдает сигналы RD (**R**ead **D**irection — читать направление), RL (**R**ead **L**atch — читать фиксацию) и, соответственно, RP, которые через драйверы TD, TL и TP передают соответствующее значение на внутреннюю шину данных.

Выходной усилитель-формирователь порта микроконтроллера семейства AVR при низком уровне сигнала на выходе в состоянии принимать входные токи силой до 20 мА и благодаря этому, например, напрямую управлять светодиодами, подключенными к питающему напряжению.

Большинство портов альтернативно применяются для выполнения каких-либо особых функций, например, для мультиплицированной шины адресов и данных внешней памяти RAM, для входов таймеров, прерываний, аналоговых компараторов, последовательных интерфейсов SPI, интерфейсов UART, выходов таймеров и т.д. Эти функции подробно описаны в руководстве того или иного модуля.

Автоматическое переключение направления передачи данных происходит не при всех особых функциях. В таких случаях пользователь должен самостоятельно сконфигурировать регистр направления передачи данных.

На рис. 10.2 на примере вывода RXEN интерфейса UART показано автоматическое переключение функции порта для PD0 как представителя всех особых функций.

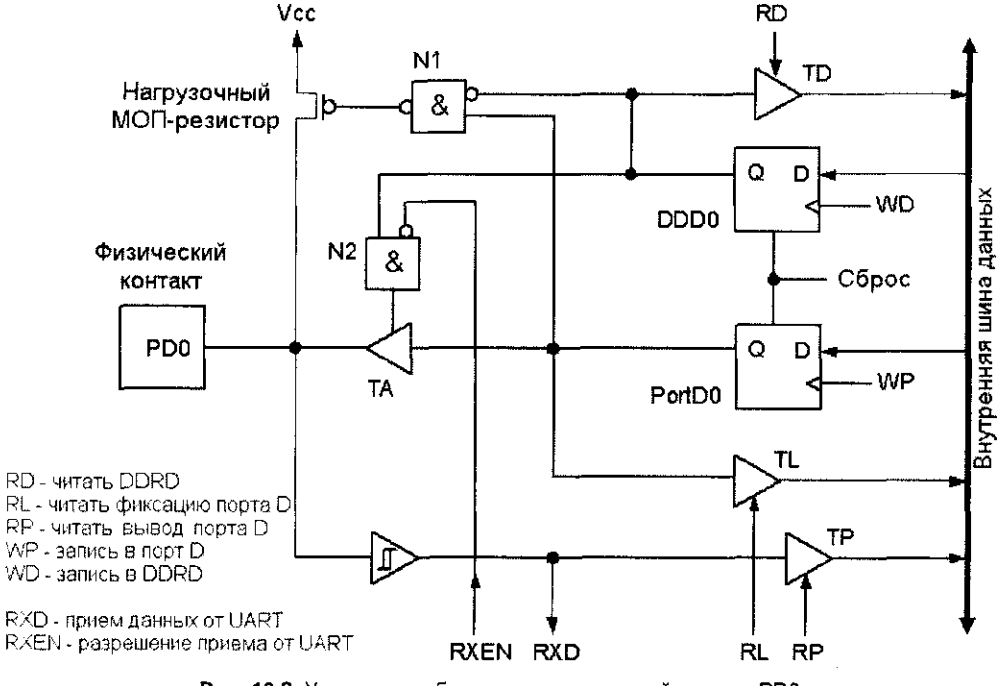


Рис. 10.2. Упрощенная блок-схема соединений вывода PD0

Выход вентиля "НЕ-И" N1, выполняющего функцию ключей S1 и S2, показанных на рис. 10.1, только тогда переходит в состояние лог. 0, когда в соответствии с табл. 10.2 триггер PortD0 содержит высокий уровень, а триггер DDD0 — низкий уровень. Низкий уровень на выходе N1 включает подтягивающее МОП-сопротивление — МОП-транзистор с высоким канальным сопротивлением (приблизительно 35–150 кОм) во включенном состоянии.

Лог. 1 на выходе DDD0 через выход вентиля N2 освобождает выходной драйвер TA и посредством этого подключает выход PortD0 к выводу PD0, как только

выход N2 будет разблокирован с помощью лог. 0 на внутренней точке подсоединения RXEN. RXEN — это разряд в управляющем регистре UCR интерфейса UART, который разблокирует/блокирует приемник интерфейса UART. Если разряд RXEN находится в состоянии высокого уровня, то низкий уровень на выходе N2 переключат выходной усилитель-формирователь TA в высокоомное состояние и таким образом отделяет триггер PortD0 от вывода PD0.

Вывод PD0 также связан напрямую со входом RxD схемы распознавания данных интерфейса UART.

Порт A

Порт A присутствует только в моделях AT90S4414 и AT90S8515. Наряду с общими функциями ввода/вывода, в качестве особой функции через этот порт осуществляется управление работой мультиплексированной шины адресов и данных, если к микроконтроллеру семейства AVR подключено внешнее запоминающее устройство RAM. На этот случай порт A оснащен внутренними подтягивающими сопротивлениями. Работа с портом A при выполнении этой альтернативной функции подробно описано в разделе "Внешняя память SRAM" главы 3.

Когда порт A с помощью разряда SRE регистра MCUCR переключен на выполнение альтернативной функции, его регистр направления передачи данных DDRA соответствующим образом переписывается.

Регистр данных порта A

Регистр данных PORTA расположен в области ввода/вывода по адресу $1B (по адресу $3B в памяти RAM) и доступен для чтения и записи. После поступления сигнала сброса он инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$1B ($3B)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA

Регистр направления передачи данных DDRA

Регистр направления передачи данных DDRA расположен в области ввода/вывода по адресу $1A (по адресу $3A в памяти RAM) и доступен для чтения и записи. После поступления сигнала сброса он инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$1A ($3A)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA

Адреса порта A — выводы PINA

Здесь речь идет не о регистре, а о прямом доступе по некоторому адресу через выводы порта A. При считывании порта A получают содержимое внутреннего регистра PortA, а при считывании информации с выводов PINA получают логический уровень, который в настоящий момент присутствует на выводах порта A (см. рис. 10.1).

Обращение к выводам PINA порта A осуществляется через область ввода/вывода по адресу $19 (по адресу $39 в памяти RAM). Они доступны только для чтения. После поступления сигнала сброса выводы PINA находятся в высокоомном состоянии.

Разряд	7	6	5	4	3	2	1	0	
$19 ($39)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA

Порт B

Порт B присутствует во всех представителях базовой серии семейства AVR. Наряду с выполнением общих функций ввода/вывода, он также выполняет также и некоторые особые функции. Занятость порта B в отношении этих альтернативных функций для разных представителей микроконтроллеров базовой серии семейства AVR различна и задается явным образом при описании регистра данных порта для каждого микроконтроллера.

Регистр данных порта B

Регистр данных порта B расположен в области ввода/вывода по адресу $18 (по адресу $38 в памяти RAM) и доступен для чтения и записи. После поступления сигнала сброса он инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$18 ($38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
AT90S1200	SCK	MISO	MOSI	–	–	–	AIN1	AIN0	Особ. функ.
AT90S2313	SCK	MISO	MOSI	–	OC1	–	AIN1	AIN0	Особ. функ.
AT90S4414	SCK	MISO	MOSI	/SS	AIN1	AIN0	T1	T0	Особ. функ.
AT90S8515	SCK	MISO	MOSI	/SS	AIN1	AIN0	T1	T0	Особ. функ.

SCK — тактовый выход в случае, если ведущим устройством является последовательный интерфейс SPI, и тактовый вход в случае, если этот интерфейс является ведомым. Если интерфейс SPI сконфигурирован как ведомое устройство, то вывод SCK является входом независимо от направления передачи данных, установленного в регистре DDB7. Подтягивающее сопротивление в этом случае по-прежнему может быть подключено и отключено с помощью регистра порта PORTB7. Если интерфейс SPI сконфигурирован как ведущее устройство, то направление передачи данных для вывода SCK задается регистром DDB7. Более подробно эта функция описана в главе 7.

В микроконтроллерах моделей AT90S1200 и AT90S2313 последовательный интерфейс SPI для пользователя недоступен. Здесь вывод SCK служит только в качестве тактового входа при последовательном программировании.

MISO — информационный вход в случае использования последовательного интерфейса SPI в качестве ведущего устройства, и информационный выход в случае, если этот интерфейс применяется в качестве ведомого. Если интерфейс SPI

сконфигурирован для рабjты в качестве ведущего устройства, то вывод MISO является входом независимо от направления передачи данных, установленного в регистре DDB6. Подтягивающее сопротивление в этом случае по-прежнему может быть подключено и отключено с помощью регистра порта PORTB6. Если интерфейс SPI сконфигурирован для работы в качестве ведомого устройства, то направление передачи данных для вывода MISO будет задано регистром DDB6. Более подробно эта функция описана в главе 7.

В микроконтроллерах моделей AT90S1200 и AT90S2313 последовательный интерфейс SPI для пользователя недоступен. Здесь вывод MISO служит только в качестве информационного выхода при последовательном программировании.

MOSI — информационный выход в случае использования последовательного интерфейса SPI в качестве ведущего устройства, и информационный вход в случае, если этот интерфейс применяется в качестве ведомого. Если интерфейс SPI сконфигурирован для работы в качестве ведомого устройства, то вывод MOSI является входом независимо от направления передачи данных, установленного в регистре DDB5. Подтягивающее сопротивление в этом случае по-прежнему может быть подключено и отключено с помощью регистра порта PORTB5. Если интерфейс SPI сконфигурирован для работы в качестве ведущего устройства, то направление передачи данных для вывода MOSI будет задано регистром DDB5. Более подробно эта функция описана в главе 7.

В микроконтроллерах моделей AT90S1200 и AT90S2313 последовательный интерфейс SPI для пользователя недоступен. Здесь вывод MOSI служит только в качестве информационного входа при последовательном программировании.

/SS — вход выбора в том случае, если последовательный интерфейс SPI является ведомым. Если интерфейс SPI сконфигурирован для работы в качестве ведомого устройства, то вывод /SS является входом независимо от направления передачи данных, установленного в регистре DDB4. При наличии на этом выводе сигнала низкого уровня ведомое устройство является активным. Подтягивающее сопротивление в этом случае по-прежнему может быть подключено и отключено с помощью регистра порта PORTB4. Если интерфейс SPI сконфигурирован для работы в качестве ведущего устройства, то направление передачи данных через вывод /SS задается регистром DDB4. Более подробно эта функция описана в главе 7.

AIN1 — инвертирующий вход аналогового компаратора. В интересах высокого входного сопротивления в этом случае рекомендуется отключить подтягивающее сопротивление. Более подробно эта функция описана в главе 9.

AIN0 — неинвертирующий вход аналогового компаратора. В интересах высокого входного сопротивления в этом случае рекомендуется отключить подтягивающее сопротивление. Более подробно эта функция описана в главе 9.

OC1 — выход функции сравнения таймера/счетчика T/C1 в микроконтроллере AT90S2313. В этом случае вывод PB3 должен быть сконфигурирован для работы в качестве выхода (регистр DDB3 содержит лог. 1). Более подробно эта функция рассматривается в "16-разрядный таймер/счетчик T/C1" главы 4.

T1 — тактовый вход для T/C1. Более подробно эта функция рассматривается в разделе "16-разрядный таймер/счетчик T/C1" главы 4.

T0 — тактовый вход для T/C0. Более подробно эта функция рассматривается в разделе "8-разрядный таймер/счетчик T/C0" главы 4.

Регистр направления передачи данных DDRB

Регистр направления передачи данных DDRB расположен в области ввода/вывода по адресу $17 (по адресу $37 в памяти RAM) и доступен для чтения и записи. После поступления сигнала сброса этот регистр инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$17 ($37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB

Адреса порта B — выводы PINB

Здесь речь идет не о регистре, а о прямом доступе по некоторому адресу через выводы порта B. При считывании порта B получают содержимое внутреннего регистра PortB, а при считывании информации с выводов PINB получают логический уровень, который в настоящий момент присутствует на выводах порта B (см. рис. 10.1).

Обращение к выводам PINB порта B осуществляется через область ввода/вывода по адресу $16 (по адресу $36 в памяти RAM). Они доступны только для чтения. После поступления сигнала сброса выводы PINB находятся в высокоомном состоянии.

Разряд	7	6	5	4	3	2	1	0	
$16 ($36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB

Порт C

Порт C используется только в микроконтроллерах моделей AT90S4414 и AT90S8515. Наряду с общими функциями ввода/вывода, в качестве особой функции выполняется вывод через порт C адресного байта более высокого значения адресной шины, когда к микроконтроллеру семейства AVR подсоединена внешняя память RAM. На этот случай порт C оснащен внутренними подтягивающими сопротивлениями. Работа с портом C при выполнении этой альтернативной функции подробно описано в разделе "Внешняя память SRAM" главы 3.

Когда порт C с помощью разряда SRE регистра MCUCR переключен на выполнение альтернативной функции, его регистр направления передачи данных DDRC соответствующим образом переписывается.

Регистр данных порта C

Регистр данных PORTC расположен в области ввода/вывода по адресу $15 (по адресу $35 в памяти RAM) и доступен для чтения и записи. После поступления сигнала сброса он инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$15 ($35)	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC

Регистр направления передачи данных DDRC

Регистр направления передачи данных DDRC расположен в области ввода/вывода по адресу $14 (адрес $34 в RAM) и доступен для чтения и записи.

После поступления сигнала сброса он инициализируется значением $00.

Разряд	7	6	5	4	3	2	1	0	
$14 ($34)	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC

Адреса порта C — выводы PINC

Здесь речь идет не о регистре, а о прямом доступе по некоторому адресу через выводы порта C. При считывании порта C получают содержимое внутреннего регистра PortC, а при считывании информации с выводов PINC получают логический уровень, который в настоящий момент присутствует на выводах порта C (см. рис. 10.1).

Обращение к выводам PINC порта C осуществляется через область ввода/вывода по адресу $13 (по адресу $33 в памяти RAM). Они доступны только для чтения. После поступления сигнала сброса выводы PINC находятся в высокоомном состоянии.

Разряд	7	6	5	4	3	2	1	0	
$13 ($33)	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC

Порт D

Порт D предоставляется всеми представителями микроконтроллеров базовой серии семейства AVR, однако в моделях AT90S1200 и AT90S2313 его ширина составляет только 7 разрядов, поскольку из-за наличия на корпусе только 20 точек подсоединения от разряда PD7 пришлось отказаться.

Наряду с выполнением общих функций ввода/вывода, порт D также выполняет некоторые особые функции. Занятость порта D в отношении этих альтернативных функций для разных представителей микроконтроллеров базовой серии семейства AVR различна и задается явным образом при описании регистра данных порта для каждого микроконтроллера.

Регистр данных порта D

Регистр данных порта D расположен в области ввода/вывода по адресу $12 (по адресу $32 в RAM) и доступен для чтения и записи. После поступления сигнала сброса этот регистр инициализируется значением $00. В моделях AT90S1200 и AT90S2313 регистр PORTD7 доступен только чтения (всегда содержит лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$12 ($32)	PORTD7 *)	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
AT90S1200	–	–	–	T0	–	INT0	–	–	Особ. функ.
AT90S2313	–	ICP	T1	T0	INT1	INT0	TxD	RxD	Особ. функ.
AT90S4414	/RD	/WR	OC1A	–	INT1	INT0	TxD	RxD	Особ. функ.
AT90S8515	/RD	/WR	OC1A	–	INT1	INT0	TxD	RxD	Особ. функ.

*) отсутствует в моделях AT90S1200 и AT90S2313

/RD — управляющий сигнал для обращения на чтение к внешней памяти RAM. Подсоединение внешней памяти RAM к микроконтроллерам семейства AVR подробно описано в разделе "Внешняя память SRAM" главы 3.

/WR — управляющий сигнал для обращения на запись к внешней памяти RAM. Подсоединение внешней памяти RAM к микроконтроллерам семейства AVR подробно описано в в разделе "Внешняя память SRAM" главы 3.

ICP — вход для функции захвата таймера/счетчика T/C1 в микроконтроллере AT90S2313. Более подробно эта функция рассматривается в "16-разрядный таймер/счетчик T/C1" главы 4.

OC1A — выход для функции сравнения или функции ШИМ таймера/счетчика T/C1 в микроконтроллерах AT90S4414 и AT90S9515. В этом случае вывод PD5 должен быть сконфигурирован как выход (регистр DDD5 содержит лог. 1). Более подробно эта функция рассматривается в "16-разрядный таймер/счетчик T/C1" главы 4.

T1 — тактовый вход для T/C1. Более подробно эта функция рассматривается в "16-разрядный таймер/счетчик T/C1" главы 4.

T0 — тактовый вход для T/C0. Более подробно эта функция рассматривается в "8-разрядный таймер/счетчик T/C0" главы 4.

INT1 — вход для внешнего прерывания 1. Более подробно эта функция описана в разделе "Сброс и обработка прерываний" главы 3.

INT0 — вход для внешнего прерывания 0. Более подробно эта функция описана в разделе "Сброс и обработка прерываний" главы 3.

TxD — информационный выход приемопередатчика UART. Более подробно эта функция описана в главе 6.

RxD — информационный вход приемопередатчика UART. Более подробно эта функция описана в главе 6.

Регистр направления передачи данных DDRD

Регистр направления передачи данных порта DDRD расположен в области ввода/вывода по адресу $11 (по адресу $31 в RAM) и доступен для чтения и записи. После поступления сигнала сброса он инициализируется значением $00. В микроконтроллерах AT90S1200 и AT90S2313 регистр DDD7 доступен только для чтения (всегда содержит лог. 0).

Разряд	7	6	5	4	3	2	1	0	
$11 ($31)	DDD7 *)	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD

*) отсутствует в микроконтроллерах AT90S1200 и AT90S2313

Адреса порта D — выводы PIND

Здесь речь идет не о регистре, а о прямом доступе по некоторому адресу через выводы порта D. При считывании порта D получают содержимое внутреннего регистра PortD, а при считывании информации с выводов PIND получают логический уровень, который в настоящий момент присутствует на выводах порта D (см. рис. 10.1).

Обращение к выводам PIND порта B осуществляется через область ввода/вывода по адресу $10 (по адресу $30 в памяти RAM). Они доступны только для чтения. В микроконтроллерах AT90S1200 и AT90S2313 вывод PIND7 отсутствует и всегда принимается как лог. 0. После поступления сигнала сброса выводы PIND находятся в высокоомном состоянии.

Разряд	7	6	5	4	3	2	1	0	
$10 ($30)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND

*) отсутствует в микроконтроллерах AT90S1200 и AT90S2313

Выход с открытым коллектором

Несмотря на то, что выходной усилитель-формирователь микроконтроллеров семейства AVR выполнен в виде двухтактного каскада, с помощью порта можно образовать также и выход с открытым коллектором. Для этой цели в фиксатор порта PortXn постоянно записывается лог. 0, а выход подключается с помощью разряда направления передачи данных DDXn (см. рис. 10.1).

Ключ S2, благодаря лог. 0 в регистре PortXn, всегда разомкнут, и подтягивающее сопротивление постоянно отключено независимо от значения в регистре направления передачи данных DDXn. Если регистр DDXn содержит лог. 0, то выходной усилитель-формирователь, который на рис. 10.1 символически показан в виде ключа S3, будет находиться в высокоомном состоянии, а выход будет вести себя как каскад с открытым коллектором с отключенным транзистором.

Если разряд DDxn в регистре направления передачи данных содержит лог. 1, то выходной усилитель-формирователь активен (ключ S3 замкнут), и переключает лог. 0 порта PortXn на выход. В результате выход ведет себя так как каскад с открытым коллектором с подключенным транзистором.

Соответствующие условия показаны на рис. 10.3.

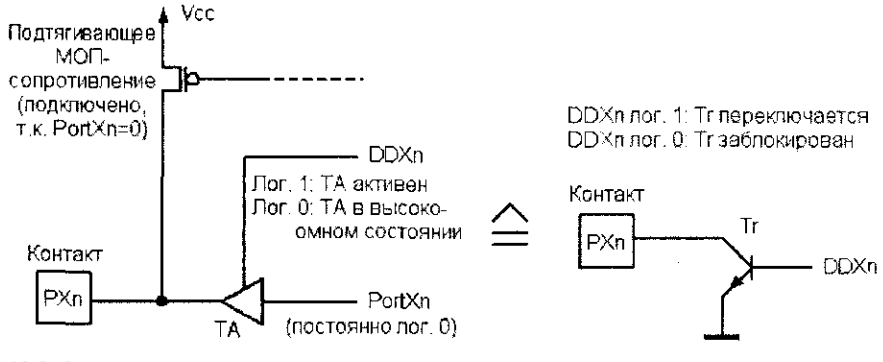


Рис. 10.3. Символическое представление порта как выходного каскада с открытым коллектором

11 Программирование памяти

Наряду с памятью программ и внутренней памятью EEPROM, во всех микроконтроллерах AVR реализованы также различные механизмы ограничения возможности программирования и сохранения запрограммированных данных.

Разряды блокировки памяти программ LB1 и LB2

Посредством программирования в лог. 0 разрядов LB1 и LB2 для микросхем микроконтроллеров семейства AVR могут быть активизированы защитные механизмы, указанные в табл. 11.1.

Таблица 11.1. Механизмы защиты микросхем микроконтроллеров семейства AVR

Режим	*LB1*	*LB2*	*Защитная функция*
1	1	1	Никаких ограничений со стороны защитной функции
2	0	1	Запоминающие устройства с памятью Flash и EEPROM заблокированы для дальнейшего программирования. В параллельном режиме также больше не могут программироваться и разряды предохранения! По этой причине разряды предохранения всегда должны быть запрограммированы раньше разрядов блокировки
3	0	0	Как и в режиме 2, но дополнительно невозможно выполнять операции считывания из памяти

Если будет запрограммирован только один разряд блокировки LB2, то никакая защитная функция установлена не будет, поскольку с помощью разряда блокировки LB1 прежде должен быть защищен от записи кристалл, после чего можно установить защиту от считывания. В незапрограммированном состоянии (состояние поставки) оба разряда содержат лог. 1.

Стирание запрограммированных разрядов блокировки возможно только с помощью функции стирания чипа, которая, впрочем, стирает весь блок в целом.

Разряды предохранения RCEN, FSTRT и SPIEN

Разряд RCEN (только в микроконтроллере AT90S1200):

- лог. 0 — выработка тактовых импульсов с частотой около 1 МГц для процессора осуществляется с помощью внутреннего RC-осциллятора;
- лог. 1 — выработка тактовых импульсов для процессора осуществляется через выводы XTAL1 и XTAL2 (состояние поставки).

Подробности изложены в разделе "Генерирование такта системной синхронизации с помощью контура RC-осциллятора" главы 2.

Разряд FSTRT (отсутствует в микроконтроллере AT90S1200):

- лог. 0 — выбирается режим с сокращенным временем запуска;

- лог. 1 — выбирается нормальное время запуска (состояние поставки).

Подробности изложены в разделе "Сброс и обработка прерываний" главы 3.

Разряд SPIEN:

- лог. 0 — разрешен последовательный режим программирования (состояние поставки);
- лог. 1 — последовательный режим программирования заблокирован.

Подробности изложены ниже в соответствующем разделе.

К разрядам предохранения нельзя обратиться ни в последовательном режиме программирования, ни через команду стирания кристалла! Эти разряды можно изменить исключительно в параллельном режиме программирования.

Байты сигнатуры

Для однозначной идентификации микросхемы компания Atmel предоставляет возможность считывания трех байтов сигнатуры. Это возможно как при параллельном, так и при последовательном режиме. Три байта сигнатуры расположены в отдельном адресном пространстве. Их значения представлены в табл. 11.2.

Таблица 11.2. Значения байтов сигнатуры

Адрес	Байт данных	Значение
$000	$1E	Идентифицирует компанию Atmel как изготовителя
$001	$90	512x16-разрядная Flash-память (модель AT90S1200)
	$91	1024x16-разрядная Flash-память (модель AT90S2313)
	$92	2048x16-разрядная Flash-память (модель AT90S4414)
	$93	4096x16-разрядная Flash-память (модель AT90S8515)
$002	$01	Всегда содержит $01, если значение байта по адресу $001 находится между $90 и $93

Стереть байты сигнатуры невозможно. В третьем режиме блокировки нельзя считать байты сигнатуры в последовательном режиме программирования. Попытка сделать это даст результат в виде $00, $01 и $02.

Процесс программирования

В состоянии поставки как запоминающее устройство типа Flash, так и внутренняя память EEPROM микроконтроллеров семейства AVR стерты, то есть, все ячейки памяти содержат $FF и готовы к процессу программирования.

Для программирования памяти компания Atmel предоставляет на выбор параллельный или последовательный режим программирования. В случае параллельного режима программирования в распоряжении имеются все возможности программирования, а при ограниченном последовательном режиме нет необходимости в каком-то особом напряжении программирования, и микросхема может быть запрограммирована обычным образом даже в составе схемы.

Параллельный режим программирования

Для этого процесса требуется вспомогательное напряжение номиналом +12 В (+11,5–12,5 В). Впрочем, оно нагружается очень незначительным током (< 250 мкА) и потому может быть легко получено с помощью простой схемы накачки заряда, если его нельзя получить как рабочее напряжение с помощью встроенных аппаратных средств.

На рис. 11.1 показано подсоединение микроконтроллера семейства AVR для выполнения параллельного программирования (стрелки указывают направление передачи данных).

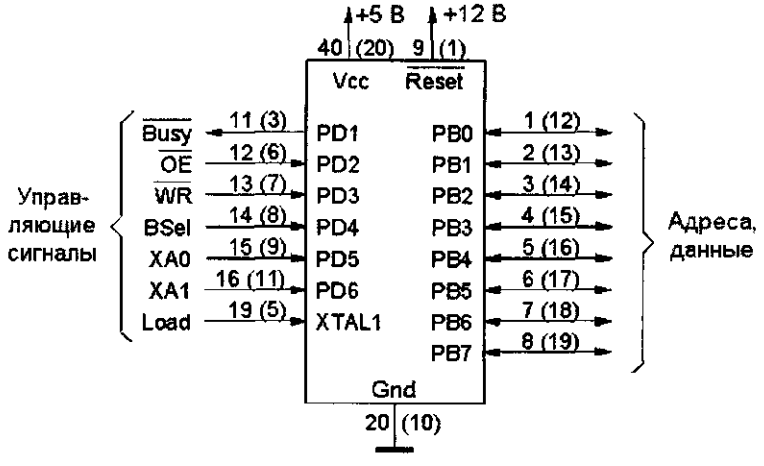


Рис. 11.1. Подсоединение микроконтроллера семейства AVR для параллельного программирования

Указанная на рисунке нумерация выводов относится к 40-полюсному корпусу типа PDIP (**P**lastic **D**ual-**I**nline **P**ackage — пластмассовый корпус с двухрядным расположением выводов), а обозначения выводов, взятые в скобки, — к 20-полюному корпусу типа PDIP. Управляющие сигналы на рис. 11.1 имеют следующее назначение:

- /Busy:
  - лог. 0 — выполняется программирование;
  - лог. 1 — микросхема готова к получению нового командного слова;
- /OE (**O**utput **E**nable — разрешение вывода) — активизирует каскад выходного усилителя-формирователя при считывании (проверке) данных;
- /WR — сигнал записи, запускает процесс программирования;
- BSel (**B**yte **S**elect — выбор байта):
  - лог. 0 — указывает на младший байт адреса или слова данных;
  - лог. 1 — указывает на старший байт адреса или слова данных;
- XA0, XA1 — оба разряда определяют действие, выполняемое при поступлении на вход XTAL1 положительного импульса загрузки (Load). Функции сигналов XA0 и XA1 в процессе программирования и проверки, представлены в табл. 11.3.

Таблица 11.3. Функции разрядов XA0, XA1

XA1	XA0	Функция
0	0	Загрузить старший (BSel = 1) или младший (BSel = 0) байт адреса памяти Flash или EEPROM
0	1	Загрузить старший (BSel = 1) или младший (BSel = 0) байт данных
1	0	Загрузить командное слово
1	1	Никаких действий

Перед поступлением импульса низкого уровня /WR или /OE должно быть загружено одно из командных слов (табл. 11.4), определяющих операцию.

Таблица 11.4. Командные слова для выполнения действий программирования и проверки

Командное слово	Функция
$1000\ 0000_b$	Стереть всю информацию на кристалле
$0100\ 0000_b$	Программирование разрядов предохранения
$0010\ 0000_b$	Программирование разрядов блокировки
$0001\ 0000_b$	Программирование флэш-памяти
$0001\ 0001_b$	Программирование памяти EEPROM
$0000\ 1000_b$	Считывание байтов сигнатуры
$0000\ 0100_b$	Считывание разрядов блокировки и предохранения
$0000\ 0010_b$	Чтение флэш-памяти
$0000\ 0011_b$	Чтение папяти EEPROM

Назначение разрядов в составе командного слова указано в табл. 11.5.

Таблица 11.5. Назначение отдельных разрядов командного слова

Разряд	Функция	
0	Лог. 0:	Обращение к флэш-памяти
	Лог. 1:	Обращение к памяти EEPROM
1	Лог. 1:	Считывание байта данных из флэш-памяти (если разряд 0 = 0) или из памяти EEPROM (если разряд 0 = 1)
2	Лог. 1:	Разряды блокировки и предохранения в считанном байте. Разряды этого байта данных имеют следующее значение: 7 — разряд блокировки 1; 6 — разряд блокировки 2; 5 — разряд предохранения SPIEN; 0 — разряд предохранения RCEN/FSTRT
3	Лог. 1:	Считывание байтов сигнатуры
4	Лог. 1:	Запрограммировать байт во флэш-память (если разряд 0 = 0) или память EEPROM (если разряд 0 = 1)
5	Лог. 1:	Программирование разрядов блокировки. Биты в переданном байте данных: 1 — разряд блокировки 1; 2 — разряд блокировки 2 (для программирования разряда блокировки соответствующий разряд должен содержать лог. 0)
6	Лог. 1:	Программирование разрядов предохранения. Биты в переданном байте: 5 — разряд SPIEN; 0 — разряд RCEN/FSTRT. Для программирования разряда предохранения соответствующий разряд должен быть в состоянии лог. 0, а для стирания — в состоянии лог. 1
7	Лог. 1:	Стирание информации с кристалла (полное стирание данных из памяти Flash и EEPROM, а также разрядов блокировки; разряды предохранения остаются без изменений)

Полностью процесс параллельного программирования показан на рис. 11.2.

***Рис. 11.2.** Процесс параллельного программирования*

Для того чтобы можно было начать собственно процесс программирования, микросхема должна быть переведена в соответствующий режим. Последовательность требуемых для этого действий показана на рис. 11.3.

Рис. 11.3. Последовательность действий для перевода микросхемы в режим программирования

Перед программированием новых данных необходимо стереть те данные, которые присутствуют в микросхеме. Команда "Стирание кристалла" сначала стирает все данные, хранящиеся в памяти Flash и EEPROM, а затем — также и разряды блокировки. На разряды предохранения эта команда не влияет. Ход процедуры стирания данных с кристалла показан на рис. 11.4.

Рис. 11.4. Последовательность действий для полного стирания данных на кристалле

После того как микросхема переведена в состояние программирования и данные в памяти были стерты, можно начинать собственно процесс программирования. Для этого все байты данных, подлежащие записи в память, последовательно, байт за байтом, "прожигаются" во флэш-памяти. Командное слово для программирования флэш-памяти (10_h и, соответственно, $0001\ 0000_b$ по табл. 11.5) должно быть загружено только один раз: перед программированием первого байта данных. Это командное слово остается действительным до тех пор, пока не будет загружено другое командное слово.

Старший байт адреса, по которому должно быть запрограммировано слово данных, также остается действительным до тех пор, пока оно не будет изменено. Благодаря этому, только изменяя младший байт адреса, можно последовательно программировать целый блок, состоящий максимум из 256 слов данных, если они находятся все находятся в пределах одной "страницы" флэш-памяти (то есть, все имеют одинаковый старший байт адреса). Байты, содержащие значение FF_h при программировании можно "перепрыгнуть", поскольку такое значение содержат все стертые ячейки памяти.

Ход процесса параллельного программирования показан на рис. 11.5.

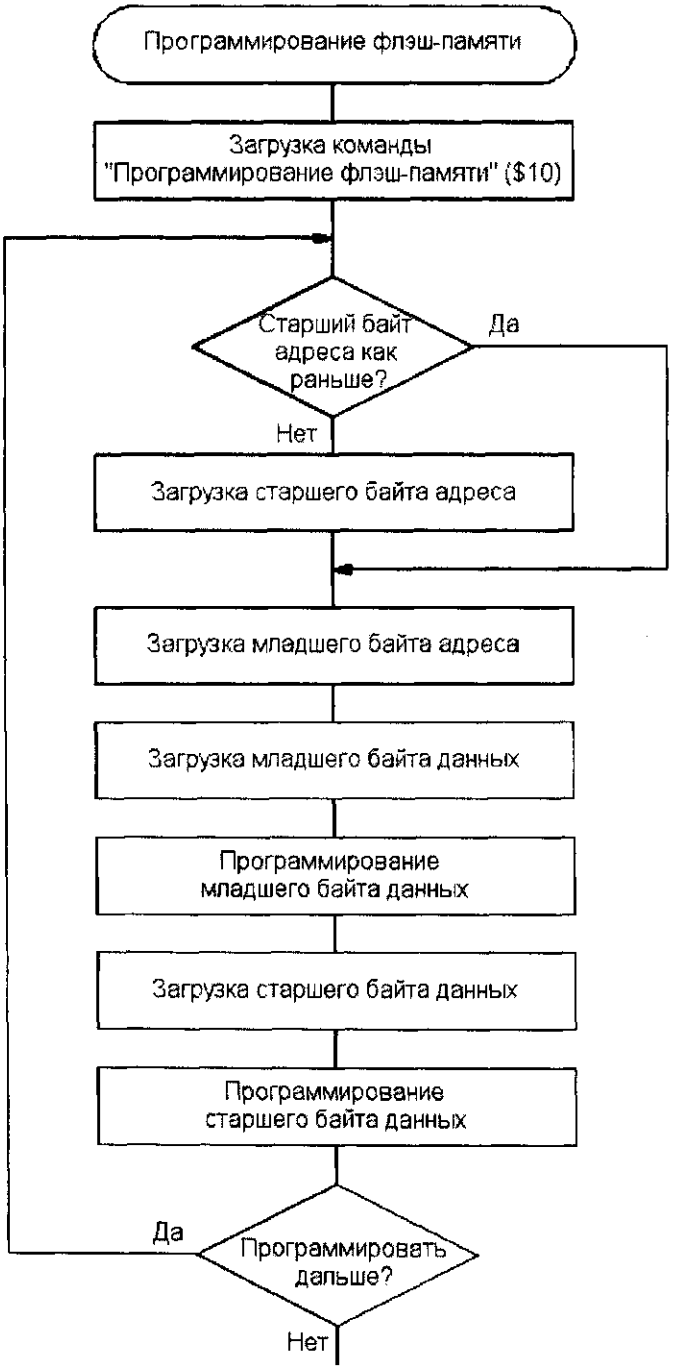


Рис. 11.5. Последовательность действий для программирования флэш-памяти

На рис. 11.6 – рис. 11.9 показана последовательность действий, которые необходимо выполнить для загрузки команд, адресов и данных, а также для самого процесса программирования.

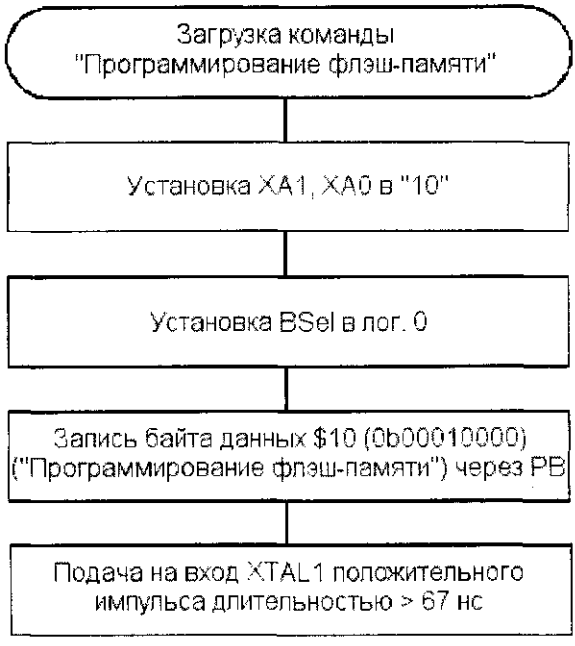


Рис. 11.6. Последовательность действий для загрузки команды программирования

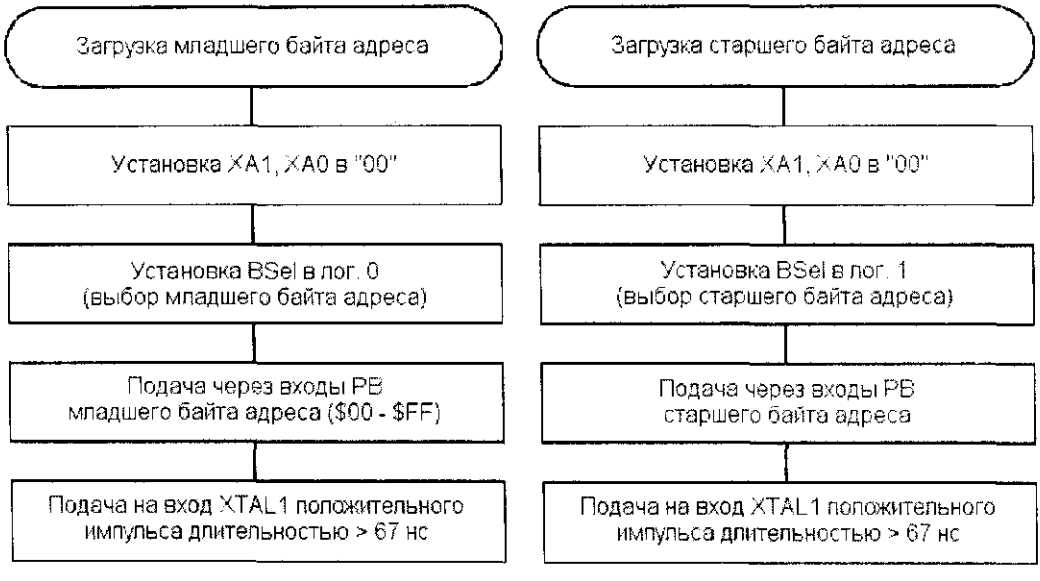


Рис. 11.7. Последовательность действий для загрузки старшего и младшего байтов адреса

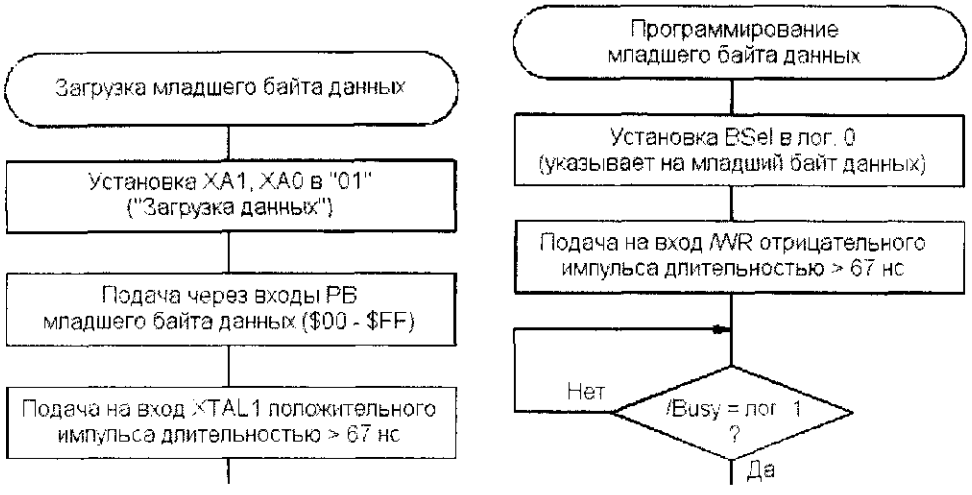


Рис. 11.8. Последовательность действий для загрузки и программирования младшего байта данных

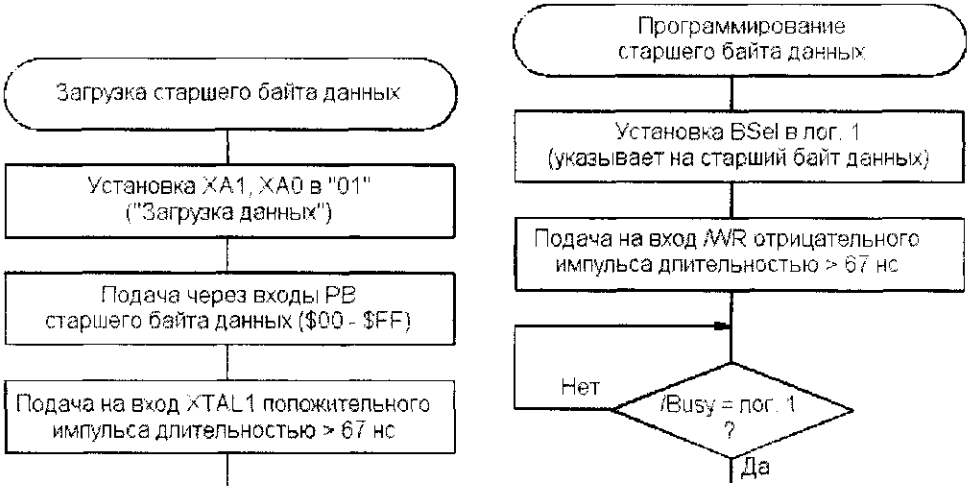


Рис. 11.9. Последовательность действий для загрузки и программирования старшего байта данных

В целом ход процесса программирования во флэш-память в параллельном режиме одного слова показан на рис. 11.10.

Считывание запрограммированных данных из флэш-памяти

Если требуется проверить запрограммированные данные после окончания процесса программирования (верификация) или считать неизвестное содержимое микросхемы микроконтроллера семейства AVR, то это можно сделать с помощью командного слова $02 (см. табл. 11.4), если только не запрограммированы разряды блокировки LB1 и LB2.

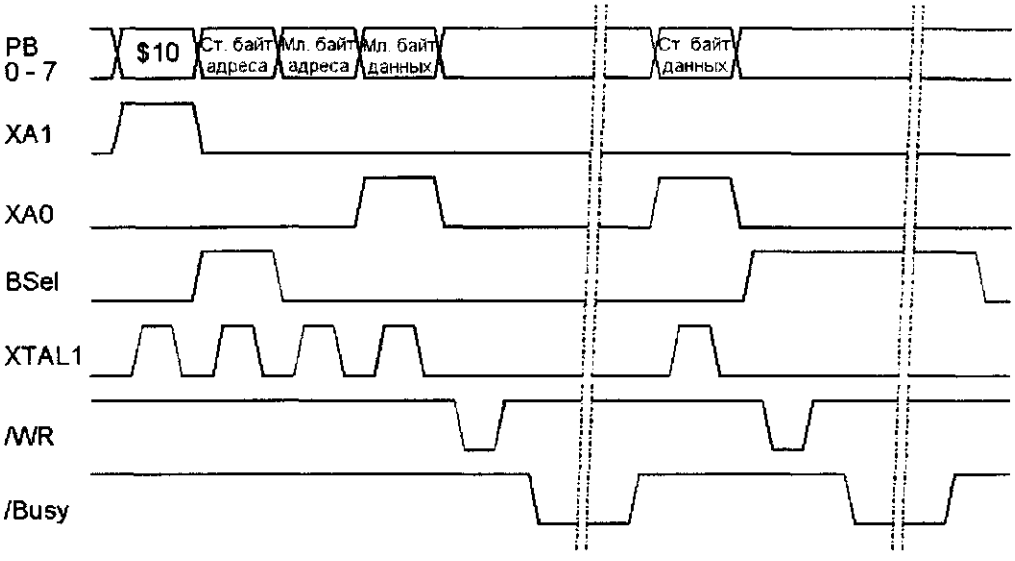


Рис. 11.10. Ход процесса параллельного программирования (/RESET = +12 D, /OE = лог. 1):

Последовательность действий для считывания содержимого флэш-памяти показана на рис. 11.11. Для этого микросхема уже должна находиться в параллельном режиме программирования (см. рис. 11.3).

Программирование памяти EEPROM

Для того, чтобы запрограммировать память EEPROM, необходимо применить следующий алгоритм (отдельные действия аналогичны показанным в примере программирования флэш-памяти).

1. Загрузка команды 0001 0001_b (11_h) в соответствии с табл. 11.4. Ход действий аналогичен загрузке команды программирования (см. рис. 11.6).
2. Загрузить старший байт адреса в соответствии с рис. 11.7 (только в микроконтроллере AT90S8515).
3. Загрузить младший байт адреса в соответствии с рис. 11.7.
4. Загрузить младший байт данных ($00–$FF) в соответствии с рис. 11.8.
5. Запрограммировать младший байт данных в соответствии с рис. 11.8.

Чтение памяти EEPROM

Последовательность действий для чтения памяти EEPROM аналогична последовательности для чтения флэш-памяти.

1. Загрузка команды 0001 0001_b (11_h) в соответствии с табл. 11.4. Ход действий аналогичен загрузке команды программирования (см. рис. 11.6).
2. Загрузить старший байт адреса в соответствии с рис. 11.7 (только в микроконтроллере AT90S8515).
3. Загрузить младший байт адреса в соответствии с рис. 11.7.
4. Сигналы управления /OE и BSel установить в 0. Байт данных из памяти EEPROM появляется на выводах порта B.

5. Посредством установки управляющего сигнала /OE в 1 каскад выходного усилителя-формирователя порта B возвращается в высокоомное состояние.

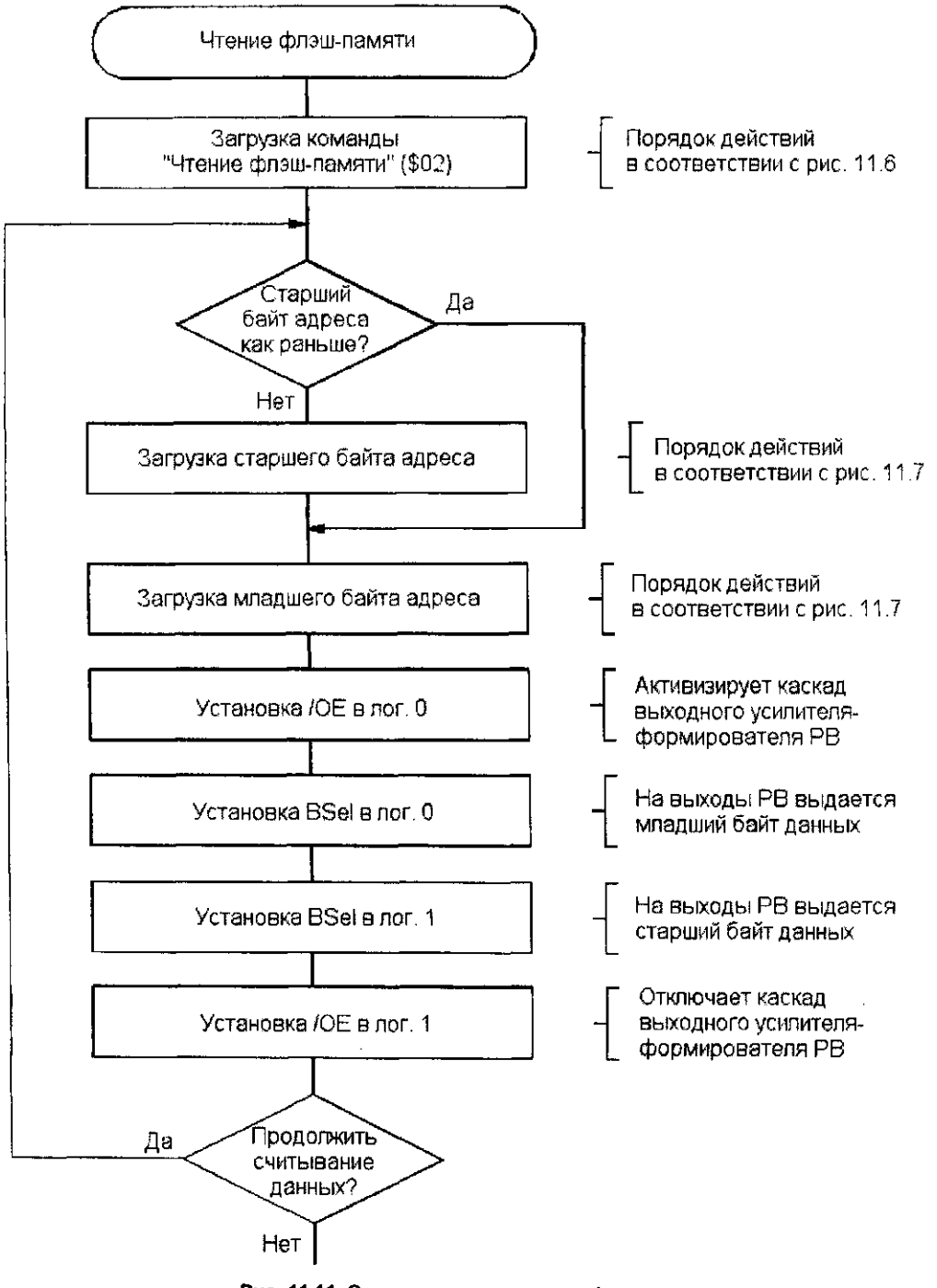


Рис. 11.11. Считывание содержимого флэш-памяти

Программирование разрядов предохранения

Для того чтобы запрограммировать разряды предохранения, применяется следующий алгоритм (отдельные действия аналогичны показанным в примере программирования флэш-памяти).

1. Загрузка команды 0100 0000_b (40_h) в соответствии с табл. 11.4. Ход действий аналогичен загрузке команды программирования (см. рис. 11.6).
2. Загрузить (младший) байт данных в соответствии с рис. 11.8.
   - лог. 0 в разряде 5 программирует, а лог. 1 стирает разряд предохранения SPIEN;
   - в микроконтроллере AT90S1200 лог. 0 в разряде 0 программирует, а лог. 1 стирает разряд предохранения RCEN;
   - в микроконтроллерах AT90S2313, AT90S4414 и AT90S8515 лог. 0 в разряде 0 программирует, а лог. 1 стирает разряд предохранения FSTRT.

   Все остальные разряды байта данных должны находиться в состоянии лог. 1, поскольку они зарезервированы компанией Atmel.
3. Программирование разрядов предохранения осуществляется посредством подачи на вход /WR импульса низкого уровня длительностью 1,5 мс (допустимыми сигналы длительностью от 1,0 до 1,8 мс.

Продолжительность процесса программирования разрядов предохранения не отображается с помощью сигнала /Busy!

Программирование разрядов блокировки

Для того, чтобы запрограммировать разряды блокировки, с помощью которых реализована защита интеллектуальной собственности от плагиаторов или просто защита микросхемы от случайного повторного программирования (см. табл. 11.1), должен быть применен представленный ниже алгоритм (отдельные действия аналогичны показанным в примере программирования флэш-памяти).

1. Загрузить команду 0010 0000_b (20_h) в соответствии с табл. 11.4. Ход действий аналогичен загрузке команды программирования (см. рис. 11.6).
2. Загрузить (младший) байт данных в соответствии с рис. 11.8. Лог. 0 в разряде 2 программирует разряд блокировки LB2, а лог. 0 в разряде 1 программирует разряд блокировки LB1. Все остальные разряды байта данных должны содержать лог. 1, поскольку они зарезервированы компанией Atmel.
3. Запрограммировать младший байт данных в соответствии с рис. 11.8.

Как уже упоминалось ранее, разряды блокировки могут быть опять стерты только по команде стирания кристалла, однако после этого будут также стерты и данные в памяти Flash и EEPROM.

Считывание разрядов предохранения и блокировки

Для считывания разрядов предохранения и блокировки применяют следующий алгоритм.

1. Загрузить команду 0000 0100_b (04_h) в соответствии с табл. 11.4. Ход действий аналогичен действиям при загрузке команды программирования (см. рис. 11.6).
2. Установить сигнал управления /OE в 0, а BSel — в 1. Состояние разрядов предохранения и блокировки появится в порту B:
   - разряд 7 = 0 — запрограммирован разряд блокировки LB1;
   - разряд 6 = 0 — запрограммирован разряд блокировки LB2;
   - разряд 5 = 0 — запрограммирован разряд предохранения SPIEN;
   - разряд 0 = 0 — запрограммированы разряды предохранения FSTRT/RCEN.
3. Посредством установки сигнала /OE в 1 каскад выходного усилителя формирователя порта B возвращается в высокоомное состояние.

Считывание байтов сигнатурны

Отдельные действия аналогичны показанным в примере программирования флэш-памяти.

1. Загрузить команду 0000 1000_b (08_h) в соответствии с табл. 11.4. Ход действий аналогичен последовательности при загрузке команды программирования (см. рис. 11.6).
2. Загрузить младший байт адресов \$00 – \$02 в соответствии с рис. 11.7.
3. Сигналы управления /OE и BSel установить в 0. Адресованный байт сигнатуры появится на порту B.
4. Посредством установки сигнала /OE в 1 каскад выходного усилителя-формирователя порта B возвращается в высокоомное состояние.

Командный байт необходимо загружать только перед считыванием первого байта сигнатуры.

Параметры параллельного режима программирования

Различные параметры параллельного режима программирования проиллюстрированы на рис. 11.12 и описаны в табл. 11.6.

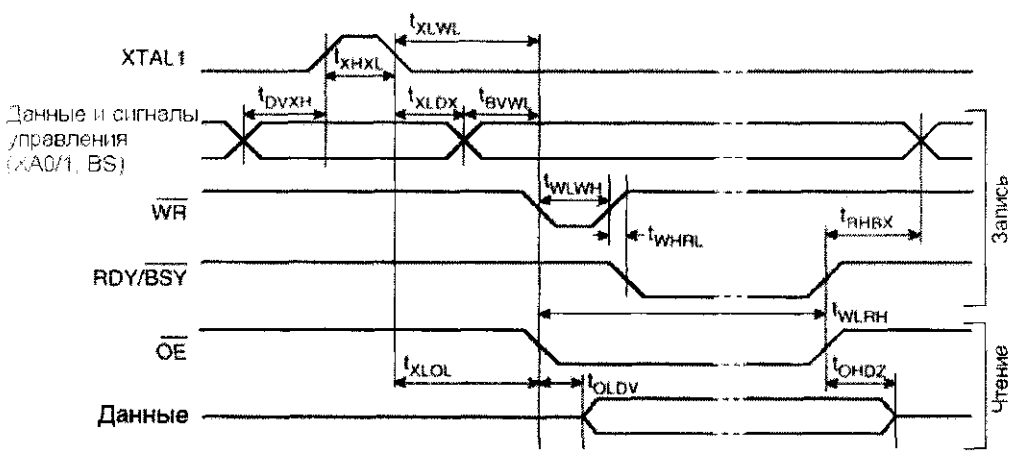


Рис. 11.12. Временные параметры в параллельном режиме программирования

Таблица 11.6. Данные, характеризующие параллельный режим программирования при $T_A = 25°C \pm 10\%$, $V_{CC} = 5\ В \pm 10\%$

Обозначение	Параметр	Мин.	Типичн.	Макс.	Ед. изм.
V_{PP}	Напряжение разрешения программирования	11,5		12,5	В
I_{PP}	Ток разрешения программирования			250	мА
t_{DVXH}	Время установки данных и сигналов управления до появления сигнала XTAL1 высокого уровня	67			нс
t_{XHXL}	Большая ширина импульса XTAL1	67			нс
t_{XLDX}	Время удержания данных и сигналов управления после появления сигнала XTAL1 низкого уровня	67			нс
t_{XLWL}	Время между сигналом XTAL1 низкого уровня и /WR низкого уровня	67			нс
t_{BVWL}	Время между сигналом проверки BS и /WR низкого уровня	67			нс
t_{RHBX}	Время удержания сигнала BS после появления высокого уровня сигнала RDY/BSY	67			нс
t_{WLWH}	Малая ширина импульса /WR (1)	67			нс
t_{WHRL}	Время между сигналом /WR высокого уровня и RDY/BSY низкого уровня		20		нс
t_{WLRH}	Время между сигналом /WR низкого уровня и RDY/BSY высокого уровня (2)	0,5	0,7	0,9	мс
t_{XLOL}	Время между сигналом XTAL1 низкого уровня и /OE низкого уровня	67			нс
t_{OLDV}	Время между сигналом /OE низкого уровня и проверкой данных		20		нс
t_{OHDZ}	Время между сигналом /OE высокого уровня и переходом информационных выходов в высоокомное состояние			20	нс
t_{WLWH_CE}	Малая ширина импульса /WR для стирания кристалла	5	10	15	мс
t_{WLWH_PFB}	Малая ширина импульса /WR для программирования разрядов предохранения	1	1,5	1,8	мс

(1) На стирание данных микросхемы должно затрачиваться время t_{WLWH_CE}, а для программирования разрядов предохранения — t_{WLWH_PFB}.

(2) Если t_{WLWH} больше, чем t_{WLRH}, то не будет генерироваться сигнал занятости /Busy.

Последовательный режим программирования

Наряду с программированием в параллельном режиме пользователь также имеет возможность ввода своих данных в последовательном режиме программирования через интерфейс SPI методом "прожига". Программирующим прибором при этом является интерфейс SPI, работающий в режиме ведущего устройства, а подлежащий программированию микроконтроллер семейства AVR выступаетег в качестве ведомого устройства. Последовательный интерфейс состоит из линий SCK, MOSI (вход) и MISO (выход). По нарастающему фронту импульсов в линии SCK биты данных записываются в подлежащий программированию микрокон-

троллер семейства AVR, а по ниспадающему фронту — считываются (рис. 11.13). Подробнее интерфейс SPI описан в главе 7.

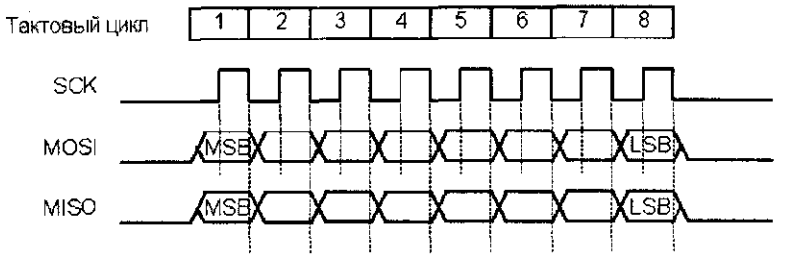


Рис. 11.13. Временная диаграмма передачи данных в режиме последовательного программирования

Последовательное программирование доступно также и в двух микроконтроллерах семейства AVR, в которых не используется интерфейс SPI: в моделях модели AT90S1200 и AT90S2313 (рис. 11.14).

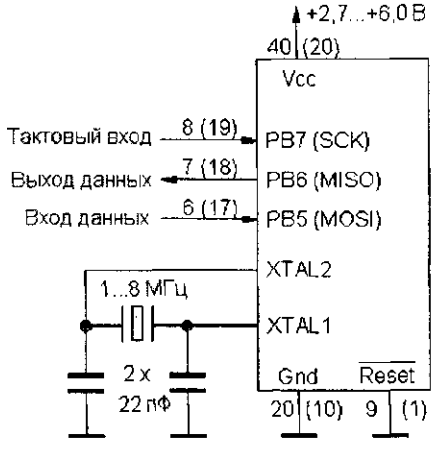


Рис. 11.14. Монтаж микроконтроллера семейства AVR для последовательного программирования

В качестве альтернативы подаче сигналов XTAL1 и XTAL2, показанной на рис. 11.14, в качестве системного такта на вход XTAL1 может быть подан внешний тактовый сигнал. Вход XTAL2 в этом случае остается открытым.

Фазы высокого и низкого уровней последовательных тактовых сигналов SCK по данным изготовителя, указанным в спецификации, должны составлять как минимум два периода такта системной синхронизации для микроконтроллеров AT90S8515, AT90S4414 и AT90S2313. В микроконтроллере AT90S1200 фаза высокого уровня должна продолжаться как минимум один период, а фаза низкого уровня — четыре периода такта системной синхронизации.

Память типов Flash и EEPROM в микроконтроллерах семейства AVR имеет различные диапазоны адресов (табл. 11.7).

Таблица 11.7. Диапазоны адресов памяти типов Flash и EEPROM

	AT90S1200	*AT90S2313*	*AT90S4414*	*AT90S8515*
Флэш-память	0000h ... 01FFh	0000h ... 03FFh	0000h ... 07FFh	0000h ... 0FFFh
Память EEPROM	0000h ... 003Fh	0000h ... 007Fh	0000h ... 00FFh	0000h ... 01FFh

Алгоритм последовательного программирования

Все команды в последовательном режиме программирования имеют собственный формат, состоящий из четырех байтов (табл. 11.8).

Таблица 11.8. Команды алгоритма последовательного программирования

Команда	*Байт 1*	*Байт 2*	*Байт 3*	*Байт 4*	*Операция*
Разрешение программирования	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Устанавливает последовательный режим программирования
Стирание кристалла	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Стирание памяти Flash, EEPROM и разрядов блокировки
Считывание флэш-памяти (память программ)	0010 **H**000	xxxx **aaaa**	**bbbb bbbb**	**oooo oooo**	Считывает старший (**H** = 1) или младший (**H** =0) байт по адресу **a:b** флэш-памяти
Запись во флэш- память	0100 **H**000	xxxx **aaaa**	**bbbb bbbb**	**iiii iiii**	Программирует старший (**H** = 1) или младший (**H** =0) байт **i** по адресу **a:b** во флэш-памяти
Считывание памяти EEPROM	1010 0000	xxxx xxx**a**	**bbbb bbbb**	**oooo oooo**	Считывает байт данных **o** по адресу **a:b** в памяти EEPROM
Запись в память EEPROM	1100 0000	xxxx xxx**a**	**bbbb bbbb**	**iiii iiii**	Программирует байт данных **i** по адресу **a:b** в памяти EEPROM
Запись разрядов блокировки	1010 1100	111x x**21**x	xxxx xxxx	xxxx xxxx	"0" в разрядах **2**, **1** программирует разряды блокировки
Считывание байта сигнатуры	0011 0000	xxxx xxxx	xxxx xx**bb**	**oooo oooo**	Считывает байт сигнатуры **o** по адресу **b**

При этом:

a — старший байт адреса
b — младший байт адреса
o — считанный байт данных
i — записанный байт данных
H = 0 — младший байт, H = 1 — старший байт
1 — Разряд блокировки LB1
2 — Разряд блокировки LB1
x — любое значение

При работе в режиме блокировки 3 (запрограммированы оба разряда блокировки) байты сигнатуры не могут быть считаны.

Адреса **a** и **b** в зависимости от объема памяти могут иметь и меньше позиций (см. табл. 11.7).

Первый байт содержит собственно код команды, характеризующий выполняемую операцию, а также целевое запоминающее устройство, по отношению к которому должна быть выполнена эта операция. Второй и третий байты данных содержат требуемый адрес в целевом запоминающем устройстве, а четвертый образует байт данных, который может быть двунаправленным (то есть, его можно как передавать, так и считывать).

Целевой микроконтроллер семейства AVR, как правило, пересылает обратно принятый ранее байт во время следующего тактового цикла. Такой возврат только что принятого байта наглядно демонстрирует представленный ниже пример двух команд, следующих одна за другой: "Разрешение" и "Считывание сигнатуры".

Команда	*Вход MOSI AVR*	*Выход MISO AVR*
Разрешение последовательного режима программирования	$AC $53 $xx $yy	$zz $AC $53 $xx
Читать байт сигнатуры по адресу $00	$30 $nn $00 $mm	$yy $30 $nn $1E

Байт $yy, принятый последним по команде разрешения, будет передан обратно как первый байт следующей команды "Чтение сигнатуры".

Некоторые команды считывают байт из целевого запоминающего устройства. Этот байт всегда передается как четвертый в последовательности команд, как это показано в примере считывания байта сигнатуры.

Прежде, чем начать процесс собственно программирования, необходимо перевести микросхему в состояние последовательного программирования. Если микроконтроллер семейства AVR перешел в **состояние последовательного программирования**, то можно обращаться к памяти, если только этому не противодействуют защитные механизмы, установленные в режиме блокировки.

Во избежание возникновения ошибок при программировании и верификации, в последовательном режиме программирования необходимо придерживаться следующей процедуры.

1. **Включение питания**. Рабочее напряжение подается между входом V_{cc} и "землей", в то время как сигналы сброса (/RESET) и SCK должны находиться в состоянии лог. 0. В том случае, когда между входами XTAL1 и XTAL2 не подключен никакой кварц, на вход XTAL1 необходимо подать внешний тактовый сигнал. Если нет возможности гарантировать, что в течение всей процедуры включения питания линия SCK будет находиться в состоянии лог. 0, то вход /RESET необходимо установить в лог. 1 как минимум на два периода такта системной синхронизации (XTAL1), после чего в линии SCK будет установлен сигнал низкого уровня.
2. **Переход в последовательный режим программирования**. После включения питания необходимо выждать как минимум 20 мс прежде, чем на вход MOSI микроконтроллера семейства AVR будет передана команда разрешения последовательного режима программирования (см. табл. 11.8).
3. **Проверка последовательного соединения**. Если последовательное соединение было установлено правильно, то микроконтроллер семейства AVR отправляет обратно только что принятый второй байт команды разрешения ($53), в то же время принимая третий байт. В любом случае все четыре байта

команды разрешения должны быть переданы — даже если третий байт не был передан обратно надлежащим образом. В случае потери синхронизации между передатчиком и приемником (например, если не был распознан тактовый импульс) на линию SCK выдается положительный импульс, чтобы сместить границы передачи на один разряд, а затем повторяется команда разрешения. Если после 32 попыток синхронизация не была достигнута, то можно сделать вывод о подсоединении дефектного микроконтроллера.

4. После выполнения команды стирания кристалла необходимо выждать время t_{WD_ERASE} (см. табл. 11.9), а затем на вход сброса /RESET подать импульс высокого уровня длительностью как минимум 2 периода такта системной синхронизации и продолжить процедуру с действия 2.
5. Подлежащие программированию данные последовательно, байт за байтом, вместе с соответствующим адресом упаковываются в команду записи в память Flash или EEPROM и передаются микроконтроллеру AVR.
6. Если информация в микроконтроллере AVR была стерта с помощью команды стирания кристалла, то байты, содержащие значение FF_h, при программировании пропускаются, поскольку во всех стертых ячейки памяти содержится именно такое значение.
7. При последовательном программировании памяти EEPROM нет необходимости удалять всю информацию из микросхемы с помощью команды стирания кристалла, поскольку в автоматический последовательный процесс записи интегрирован цикл стирания. Он предварительно стирает адресованный байт, который должен быть запрограммирован в EEPROM.
8. После записи байта данных в память Flash или EEPROM необходимо выждать время t_{WD_PROG} (см. табл. 11.9) прежде, чем можно будет начать пересылку следующей команды записи. Альтернативно, для определения окончания текущего процесса программирования также может использоваться метод опроса (см. следующий подраздел).
9. После того как были запрограммированы все байты, линия сброса /RESET может быть переведена в состояние высокого уровня, в результате чего происходит переход в обычный режим работы.
10. Альтернативно действию 9, последовательный режим программирования также можно реализовать и через последовательность "Отключение питания". Для этого на входе XTAL1 устанавливают лог. 0, если между XTAL1 и XTAL2 не подключен какой-нибудь кварц, а вход сброса /RESET устанавливают в лог. 1. После этого рабочее напряжение V_{cc} может быть отключено.

Таблица 11.9. Время ожидания t_{WD_ERASE} после поступления команды стирания кристалла, а также t_{WD_PROG} после программирования байта данных

	$V_{cc} = 3,2$ В	$V_{cc} = 3,6$ В	$V_{cc} = 4,0$ В	$V_{cc} = 5,0$ В
t_{WD_ERASE}	18 мс	14 мс	12 мс	8 мс
t_{WD_PROG}	9 мс	7 мс	6 мс	4 мс

Опрос для установления окончания процесса программирования

Сигнал занятости (/Busy), указывающий в параллельном режиме на продолжение текущего процесса программирования, в последовательном режиме программирования не используется. По этой причине пользователь прежде, чем начать программирование следующего байта, может только или выждать время t_{WD_PROG} (см. табл. 11.9), что, конечно же, значительно повышает надежность, или применить метод опроса.

При методе опроса только что запрограммированный адрес опять считывается до тех пор, пока не будет получен обратно корректно запрограммированный байт. Как только это произойдет, можно начинать программирование следующего байта.

Если запущен процесс программирования памяти EEPROM, то во время автоматического цикла стирания будет считываться адрес $80 (исключение — адрес $00 в микроконтроллере AT90S1200) и адрес $7F (исключение — $FF в микроконтроллере AT90S1200). При программировании флэш-памяти программ автоматический цикл стирания отсутствует, и во время всего цикла программирования обратно получают адрес $7F (исключение — адрес $FF в микроконтроллере модели AT90S1200).

Метод опроса применяется со всеми значениями, кроме $7F ($FF в модели AT90S1200) в случае флэш-памяти, а также $80 и $7F ($00 и $FF в микроконтроллере AT90S1200) в случае памяти EEPROM. Если эти значения должны быть запрограммированы, то необходимо выждать время t_{WD_PROG}, прежде, чем можно будет приступить к передаче следующей команды записи.

Перечень возвращенных значений при опросе во время текущего процесса программирования показан в табл. 11.10.

Таблица 11.10. Возвращенные значения при опросе во время текущего процесса программирования

	AT90S8515	*AT90S4414*	*AT90S2313*	*AT90S1200*
Стереть память EEPROM	$80	$80	$80	$00
Запрограммировать память EEPROM	$7F	$7F	$7F	$FF
Запрограммировать флэш-память	$7F	$7F	$7F	$FF

Параметры последовательного режима программирования

Временные параметры последовательного режима программирования проиллюстрированы рис. 11.15 и описаны в табл. 11.11.

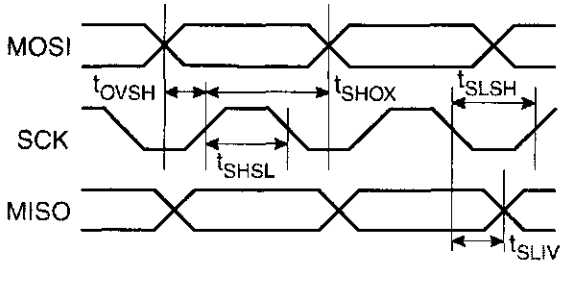


Рис. 11.15. Временная диаграмма последовательного режима программирования

Таблица 11.11. Характеристики последовательного режима программирования при T_A = –40...+85 °C и напряжении V_{CC} = 2,7...6,0 В (если не указаны никакие другие сведения)

Обознач.	Параметр	Мин.	Типичн.	Макс.	Ед. изм.
$1/t_{CLCL}$	Частота осциллятора (V_{cc} = 2,7 – 4 В)	0		4	МГц
t_{CLCL}	Период осциллятора (V_{cc} = 2,7 – 4 В)	250			нс
$1/t_{CLCL}$	Частота осциллятора (V_{cc} = 4 – 6 В)	0		8	МГц
t_{CLCL}	Период осциллятора (V_{cc} = 4 – 6 В)	125			нс
t_{SHSL}	Длительность импульса высокого уровня на линии SCK	2 t_{CLCL}			нс
t_{SLSH}	Длительность импульса низкого уровня на линии SCK	2 t_{CLCL}			нс
t_{OVSH}	Время установки входа MOSI, пока на линии SCK высокий уровень сигнала	t_{CLCL}			нс
t_{SHOX}	Время удержания входа MOSI после того как на линии SCK установился высокий уровень сигнала	2 t_{CLCL}			нс
t_{SLIV}	Время низкого уровня на линии SCK, пока действительно значение на входе MISO	10	16	32	нс

12 Система команд

Количество команд в различных моделях базовой серии семейства AVR:

- AT90S1200 — 89;
- AT90S2313 — 118;
- AT90S4414 — 118;
- AT90S8515 — 118.

В RICS-архитектурах, по сравнению с традиционными CISC-архитектурами, используются сокращенные наборы команд, что приводит к уменьшению объема программного кода и более быстрому решению поставленных задач. По этой причине, когда важна скорость, нередко предпочитают именно модули микроконтроллеров RISC.

В сравнении с обычными RISC-микроконтроллерами, компания Atmel реализовала в своем семействе AVR расширенную систему команд, благодаря чему кода становится еще меньше, а быстродействие — еще выше. Были реализованы CISC-подобные команды, однако без потерь в RISC-производительности и потребляемой мощности.

Система команд микроконтроллера AT90S8515 составляет целых 118 команд, в то время как другие RISC-микроконтроллеры того же уровня предоставляют в среднем лишь 50–60 машинных команд. Благодаря малому времени выполнения команд из своего расширенного набора, микроконтроллеры AVR решают задачи в 4–16 раз быстрее многих своих конкурентов.

Благодаря тому, что в большинстве микроконтроллеров AVR реализованы 32 рабочих регистра, каждый из которых напрямую связан с АЛУ, многие операции выполняются за один период такта системной синхронизации.

Представленный ниже пример демонстрирует, насколько эффективнее, по сравнению с традиционными CISC-контроллерами, решается задача в микроконтроллерах AVR с помощью одного лишь накапливающего сумматора.

Задача: V1 = ((V1 .and. 84_h) + (V2 .eor. V3)) .or. 80_h.

AVR-код	*CISC-код*
`eor v2,v3`	`MOV A,V2`
`andi v1,$84`	`EOR A,V3`
`add v1,v2`	`MOV TMP,A`
`ori v1,$80`	`MOV A,V1`
	`AND A,#84h`
	`ADD A,TMP`
	`OR A,#80h`
	`MOV V1,A`
4 машинных слова (8 байт)	10–18 байт
4 такта осциллятора	48–96 тактов осциллятора

Опишем некоторые обозначения и сокращения, которые будут использованы в дальнейшем при описании системы команд:

- **SREG** — регистр состояния в области ввода/вывода (адрес $3F);
- **C** — флаг переноса (разряд 0 регистра SREG);
- **Z** — нулевой флаг (разряд 1 регистра SREG);
- **N** — флаг отрицательного результата (разряд 2 регистра SREG);
- **V** — флаг переполнения при вычислениях в дополнительных кодах (разряд 3 регистра SREG);
- **S** — флаг знака (разряд 4 регистра SREG); S = N ⊕ V;
- **H** — флаг половинного переноса (разряд 5 регистра SREG);
- **T** — флаг копирования (разряд 6 регистра SREG);
- **I** — флаг общего разрешения прерываний (разряд 7 регистра SREG);
- **Rd** — регистр назначения в регистровом файле;
- **Rr** — передающий регистр;
- **R** — результат выполнения некоторой команды;
- **Kn** — n-битная константа (например, K8 = 1 байт);
- **k** — адресная константа для работы со счетчиком команд (PC);
- **b** — идентификатор некоторого разряда в рабочем регистре или в регистре ввода/вывода (3 бита);
- **s** — указывает на некоторый разряд в регистре состояния SREG;
- **X, Y, Z** — указатель при косвенной адресации (X = R27:R26, Y = R29:R28, Z = R31:R30);
- **Rdl** — младший байт регистровой пары Rdh:Rdl;
- **Rdh** — старший байт регистровой пары Rdh:Rdl;
- **P** — адрес некоторого порта ввода/вывода;
- **q** — смещение при косвенной адресации (6 бит);
- **Стек** — область памяти для хранения адреса возврата или регистр промежуточного хранения данных;
- **SP** — указатель стека при адресации стека;
- **X** — обозначает зарезервированные разряды в коде операции, устанавливаемые ассемблером в “0”;
- ✂ — пометка возле команды, отсутствующей в системе команд микроконтроллера AT90S1200.

Разряды условий (флаги) микроконтроллеров AVR

Без понимания назначения флагов составлять программы просто невозможно, поэтому мы первым делом выполним обзор разрядов условий в регистре состояния SREG.

В микроконтроллерах семейства AVR для обозначения результата выполнения той или иной операции используются флаги, входящие в состав регистра состояния SREG. Их всего восемь, и каждый из них может распознаваться особыми командами, определяющими последующий ход выполнения программы (например, `brcs`, `brhc`, `brtc`, `brie` и т.д.).

При входе в подпрограмму обработки прерывания содержимое регистра состояния рекомендуется сохранять, а после выхода из этой подпрограммы — опять восстанавливать, чтобы прерванная программа продолжила работу с гарантированно корректными разрядами условий.

Разряд 0 — C (флаг переноса)

Этот флаг указывает на переполнение в результате выполнения какой-либо арифметической или логической операции. Флаг переноса устанавливается различными командами и может опрашиваться напрямую. К операциям, изменяющим этот флаг, относятся сложение, вычитание, циклические сдвиги и некоторые логические операции. Например, переполнение в старшем разряде может возникать при сложении двух однобайтовых чисел:

	$3E_h$	=	$0011\ 1110_b$	=	62_d
	$+\ D7_h$	=	$+\ 1101\ 0111_b$	=	$+\ 215_d$
Результат:	$1]\ 15_h$	=	$1]\ 0001\ 0101_b$	=	277_d

↳ Переполнение → Флаг переноса = 1

В результате сложения возникает переполнение в старшем разряде, и устанавливается флаг переноса. В случае сложения, флаг C устанавливается, если возникает перенос в самом старшем разряде, в случае же вычитания он устанавливается в том случае, если в этом разряде возникает "заем". Этот принцип важен для арифметических операций, которые выполняются над данными, состоящими из нескольких байтов.

Рассмотрим пример вычитания числа $FEDC ($65244_d$) из числа $89AB ($35243_d$).

1. Шаг 1. Вычитание байта $DC из байта $AB:

`sub`	AB_h	=	$1010\ 1011_b$
	$-\ DC_h$	=	$-\ 1101\ 1100_b$
	$1]\ CF_h$	=	$1]\ 1100\ 1111_b$

Переполнение указывает на заем из старшего разряда.

2. Шаг 2. Вычитание байта $FE из байта $89, принимая во внимание заем, возникший при выполнении предыдущей операции:

`sbc`	89_h	=	$1000\ 1001_b$
	$-\ FE_h$	=	$-\ 1111\ 1110_b$
Перенос	$-\ 1_h$	=	$-\ 0000\ 0001_b$
	$1]\ 8A_h$	=	$1]\ 1000\ 1010_b$

Переполнение опять указывает на заем из старшего (отсутствующего) разряда и таким образом дает отрицательный результат:

$89AB – $FEDC = 1] $8ACF = -30001_d

Микроконтроллеры семейства AVR выполняют вычитание с помощью команд `sub`, `subi`, `sbiw`, `sbc`, `sbci`, `cp`, `cpc` и `cpi`. Тот же результат можно получить, если вместо использования этих команд складывать числа в дополнительных кодах, однако в таком случае будет отличаться установка флага переноса.

Рассмотрим еще один пример. Результат **-3** можно получить или с помощью сложения (команда `add`) чисел **+12** и **-15**, или же с помощью вычитания (команда `sub`) числа **+15** из числа **+12**. Для этих операций отрицательный результат определяют по флагам S и N.

`add`	$0C_h$	=	$0000\ 1100_b$	=	12_d
	$+ (-0F)_h$	=	$+ 1111\ 0001_b$	=	$+ (-15)_d$
Результат:	$0]\ FD_h$	=	$0]\ 1111\ 1101_b$	=	-3_d

Переполнения нет → флаг C сброшен.

`sub`	$0C_h$	=	$0000\ 1100_b$	=	12_d
	$- 0F_h$	=	$- 0000\ 1111_b$	=	$- 15_d$
Результат:	$1]\ FD_h$	=	$1]\ 1111\ 1101_b$	=	-3_d

"Заем" → флаг C установлен!

Возможность получить одинаковые результаты как в случае прибавления отрицательного, так и в случае вычитания положительного числа, возмещает отсутствие в наборе команд семейства AVR команды `addi` (сложение восьмиразрядных констант). Вычитание отрицательных значений дает такой же результат, что и при сложении положительных.

В следующем примере к содержимому регистра r16 прибавляется константа +5:

команда `subi r16, -5` замещает отсутствующую команду `addi r16, 5`.

Если в этом примере перед выполнением операции регистр r16 содержал $00, то после выполнения команды он будет содержать +5. Однако при этом будет установлен флаг переноса, хотя при обычном прибавлении значения +5 он установлен не был бы. Аналогичная ситуация возникает, если использовалось исходное значение $FF (= -1_d). В этом случае после выполнения операции +5 регистр r16 будет содержать +4, однако флаг переноса будет сброшен, хотя при обычном сложении +5 и $FF он был бы установлен. Поэтому, вследствие подобной потери флага переноса, может быть не распознано переполнение!

Существует еще один вариант: константа вначале загружается в какой-либо регистр, а затем содержимое обоих регистров складывается с помощью команды `add`. Если речь идет о такой константе как 1, то лучше просто воспользоваться командой `inc`.

Также, возможно сложение положительных 16-разрядных констант с помощью вычитания отрицательных значений. В представленном ниже примере значение $89AB складывается с содержимым регистров r15 и r16:

```
subi r15, low(-0x89ab)          ; вычитаем младший байт константы
sbci r16, high(-0x89ab)         ; вычитаем старший байт константы
```

Разряд 1 — Z (нулевой флаг)

Этот разряд условий всегда устанавливается, если результат какой-либо арифметической или логической операции равен нулю. В противном случае, флаг Z сбрасывается. Рассмотрим это на примере сложения двух чисел.

$2E_h$	=	$0010\ 1110_b$	=	46_d
$+\ D2_h$	=	$+\ 1101\ 0010_b$	=	$+\ 210_d$
$1]\ 00_h$	=	$1]\ 0000\ 0000_b$	=	256_d

Переполнение → флаг C = 1 ↱ ↳ Результат = 0 → флаг Z = 1

Разряд 2 — N (флаг отрицательного результата)

Флаг N указывает на отрицательный результат выполнения арифметической или логической операции. Для представления отрицательных чисел в микроконтроллерах AVR используется дополнение до двух или точное дополнение. Дополнение до двух образуется инвертированием всех разрядов N-разрядного двоичного числа с последующим прибавлением 1. В положительных числах (включая 0), старший разряд (бит 7) содержит 0, а в отрицательных, соответственно, — 1. Таким образом, наибольшее значение N-разрядного отрицательного числа — 2^{N-1}, а положительного числа — $2^{N-1} - 1$. Следовательно разрядности N = 8 (1 байт) соответствует диапазон значений -128...+127.

Разряд 3 — V (переполнение при вычислениях в дополнительных кодах)

Флаг V поддерживает арифметику дополнительных кодов. Он устанавливается, если в результате выполнения соответствующей операции над числами в дополнительном коде возникает переполнение. В противном случае, этот флаг сброшен.

При сложении двух положительных чисел в дополнительном коде результат обязательно должен получиться положительным (бит 7 = 0). Если после выполнения операции сложения знаковый разряд указывает на отрицательный результат (бит 7 = 1), то произошло переполнение.

Это же справедливо и для обратного случая сложения двух отрицательных чисел, когда результат обязательно должен получиться отрицательным (бит 7 = 1). Если же знаковый разряд указывает на положительный результат (бит 7 = 0), то это говорит о переполнении.

При выполнении арифметических операций над числами, представленными в дополнительном коде, переполнение возникает в двух случаях.

- Оба операнда положительные (бит 7 = 0), а при сложении возникает переполнение из разряда 6 в знаковый разряд, то есть, разряд 7 принимает значение 1. Например:

62_h	=	$0110\ 0010_b$	=	98_d
$+ 6F_h$	=	$+ 0110\ 1111_b$	=	$+ 111_d$
$0]\ D1_h$	=	$0]\ 1101\ 0001_b$	=	209_d

Флаг переноса = 0 ↵ ↳ Знаковый разряд изменен → флаг V = 1

- Оба операнда отрицательные (бит 7 = 1), а после сложения знаковый разряд положительный, то есть, разряд 7 принимает значение 0. Например:

$9E_h$	=	$1001\ 1110_b$	=	-98_d
$+ 91_h$	=	$+ 1001\ 0001_b$	=	$+ (-111)_d$
$1]\ 2F_h$	=	$1]\ 0010\ 1111_b$	=	-209_d

Флаг переноса = 1 ↵ ↳ Знаковый разряд изменен → флаг V = 1

Если операнды имеют разные знаки или настолько малы, что в результате сложения не превышают границ диапазона -128...+127, переполнение не возникает, и флаг V сброшен.

Разряд 4 — S (флаг знака)

S = N ⊕ V — логическая поразрядная операция "Исключающее ИЛИ" N и V.

Флаг S может использоваться для определения фактического знака результата выполнения арифметической операции. Если в результате операции не возникает переполнения при вычислениях в дополнительных кодах (флаг V = 0), то с помощью логической операции "Исключающее ИЛИ" в качестве флага знака принимается значение флага N. Если же флаг V = 1, то флаг знака принимает инвертированное значение флага N, чтобы изменить знак результата, ранее измененный в результате переполнения. Повторное инвертирование знака устанавливает его в первоначальное значение.

Если установлен флаг V, то в первом случае флаг S устанавливается в 0, что соответствует положительному результату (N ⊕ V = 1 ⊕ 1 = 0), а во втором — в 1, что соответствует отрицательному результату (N ⊕ V = 0 ⊕ 1 = 1).

Разряд 5 — H (флаг половинного переноса)

Флаг половинного переноса указывает на переполнение в младшем полубайте (разряды 0–3 байта). Он устанавливается, если возникает перенос из младшего полубайта в старший. В противном случае, флаг H сброшен.

Флаг половинного переноса особенно важен при обработке упакованных двоично-десятичных (BCD) значений. В представленном ниже примере устанавливается флаг половинного переноса H и сбрасывается флаг переноса C:

$2E_h$	=	$0010\ 1110_b$	=	46_d
$+ 74_h$	=	$+ 0111\ 0100_b$	=	$+ 116_d$
$0]\ A2_h$	=	$0]\ 1010\ 0010_b$	=	162_d

↳ Флаг H = 1, перенос из младшего полубайта.

Разряд 6 — T (флаг копирования)

Флаг T представляет собой буфер, который программист может использовать на свое усмотрение. С помощью команды `bst` выбранный разряд некоторого регистра можно скопировать в разряд T регистра состояния. Обратную операцию позволяет выполнить команда `bld`.

Разряд 7 — I (общее разрешение прерываний)

Седьмой разряд регистра состояния должен быть установлен (лог. 1) в том случае, если должны быть разрешены прерывания как таковые. Разрешение отдельных прерываний реализовано с помощью дополнительных регистров GIMSK и TIMSK. Если разряд I регистра состояния сброшен (лог. 0), то все прерывания запрещены, независимо от содержимого отдельных регистров управления.

В случае возникновения прерывания, разряд I аппаратно сбрасывается в лог. 0. Команда `reti`, с помощью которой обычно осуществляется выход из подпрограмм обработки прерываний, опять устанавливает разряд I в лог. 1, разрешая тем самым последующие прерывания.

Условный переход в программе

Все команды условного перехода, по сути, основаны на командах `brbc` и `brbs`, считывающих некоторый флаг из регистра SREG. Тем не менее, для повышения наглядности программ, компания Atmel определила три группы команд условного перехода:

- без учета знака (табл. 12.1);
- с учетом знака (табл. 12.2);
- прямого опроса флагов (табл. 12.3).

Рассмотрим эти группы подробнее.

Таблица 12.1. Команды условного перехода без учета знака

Условие	Пример операции	Вызов	Пояснения
Rd > Rr ?	`cp Rr,Rd`	`brlo *)`	Переход, если Rr < Rd (C = 1)
Rd ≥ Rr ?	`cp Rd,Rr`	`brsh/brcc`	Переход, если Rd ≥ Rr (C = 0)
Rd = Rr ?	`cp Rd,Rr`	`breq`	Переход, если Rd = Rr (Z = 1)
Rd ≤ Rr ?	`cp Rr,Rd`	`brsh *)`	Переход, если Rr ≤ Rd (C = 0)
Rd < Rr ?	`cp Rd,Rr`	`brlo/brcs`	Переход, если Rd < Rr (C = 1)

*) Не существует флага для определения условия A > B, а только для случая B < A. Таким образом, необходимо выяснить: больше ли содержимое регистра A содержимого регистра B (в таком случае, их следует поменять местами и повторить проверку) или же B < A. В результате выполнения операции B – A флаг переноса устанавливается только в том случае, если B меньше A. Этим также исключается случай B = A, и следовательно при C = 1 всегда справедливо условие A > B.

Условие A ≤ B также проверяется не напрямую. Вместо него используется условие B ≥ A. Если это условие справедливо, то при выполнении операции B – A флаг переноса сбрасывается. Следовательно, при C = 0 всегда выполняется условие A ≤ B.

Таблица 12.2. Команды условного перехода с учетом знака

Условие	*Пример операции*	*Вызов*	*Пояснения*
Rd > Rr ?	`cp Rr,Rd`	`brlt **)`	Переход, если Rr < Rd (S = 1)
Rd ≥ Rr ?	`cp Rd,Rr`	`brge`	Переход, если Rd ≥ Rr (S = 0)
Rd = Rr ?	`cp Rd,Rr`	`breq`	Переход, если Rd = Rr (Z = 1)
Rd ≤ Rr ?	`cp Rr,Rd`	`brge **)`	Переход, если Rr ≤ Rd (S = 0)
Rd < Rr ?	`cp Rd,Rr`	`brlt`	Переход, если Rd < Rr (S = 1)

**) Не существует флага для определения условия A > B, а только для случая B < A. Таким образом, необходимо выяснить: больше ли содержимое регистра A содержимого регистра B (в таком случае, их следует поменять местами и повторить проверку) или же B < A. В результате выполнения операции B – A флаг знака устанавливается только в том случае, если B меньше A. Этим также исключается случай B = A, и следовательно при S = 1 всегда справедливо условие A > B.

Условие A ≤ B также проверяется не напрямую. Вместо него используется условие B ≥ A. Если это условие справедливо, то при выполнении операции B – A флаг знака сбрасывается. Следовательно, при S = 0 всегда выполняется условие A ≤ B.

Таблица 12.3. Команды прямого опроса флагов

Условие	*Вызов*	*Пояснения*
C ?	`brcs (brcc)`	Переход, если C = 1 (C = 0)
Z ?	`breq (brne)`	Переход, если Z = 1 (Z = 0)
N ?	`brim (brpl)`	Переход, если N = 1 (N = 0)
V ?	`brvs (brvc)`	Переход, если V = 1 (V = 0)

Обзор команд микроконтроллеров AVR

Команды микроконтроллеров семейства AVR перечислены, в соответствии с группами, в табл. 12.4 – табл. 12.8.

Таблица 12.4. Команды пересылки

	Мнемоника	*Операнды*	*Операция*	*Влияние на флаги*	*Слов (байт)*	*Тактовых циклов*	*Описание*
	mov Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd ← Rr	–	1 (2)	1	Копирование содержимого регистра Rr в регистр Rd
	ldi Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd ← K8	–	1 (2)	1	Загрузка в регистр Rd 8-битной константы K8
✂	ld Rd, X	0 ≤ d ≤ 31	Rd ← (X)	–	1 (2)	2	Загрузка в регистр Rd содержимого ячейки SRAM, адресуемой с помощью 16-битного указателя X

Таблица 12.4. Продолжение

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
✂	ld Rd, X+	0 ≤ d ≤ 31	Rd ← (X) X ← X+1	–	1 (2)	2	Загрузка в регистр Rd содержимого ячейки SRAM, адресуемой с помощью 16-битного указателя X, с последующим увеличением X на 1
✂	ld Rd, -X	0 ≤ d ≤ 31	X ← X–1 Rd ← (X)	–	1 (2)	2	Уменьшение 16-битного указателя X на 1 с последующей загрузкой в регистр Rd содержимого ячейки SRAM, адресуемой с помощью X
✂	ld Rd, Y	0 ≤ d ≤ 31	Rd ← (Y)	–	1 (2)	2	Загрузка в регистр Rd содержимого ячейки SRAM, адресуемой с помощью 16-битного указателя Y
✂	ld Rd, Y+	0 ≤ d ≤ 31	Rd ← (Y) Y ← Y+1	–	1 (2)	2	Загрузка в регистр Rd содержимого ячейки SRAM, адресуемой с помощью 16-битного указателя Y, с последующим увеличением Y на 1
✂	ld Rd, -Y	0 ≤ d ≤ 31	Y ← Y–1 Rd ← (Y)	–	1 (2)	2	Уменьшение 16-битного указателя Y на 1 с последующей загрузкой в регистр Rd содержимого ячейки SRAM, адресуемой с помощью Y
✂	ldd Rd, Y+q	0 ≤ d ≤ 31 0 ≤ q ≤ 63	Rd ← (Y+q)	–	1 (2)	2	Загрузка в регистр Rd содержимого ячейки SRAM, адресуемой с помощью суммы q и содержимого 16-битного указателя Y

Таблица 12.4. Продолжение

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
	ld Rd, Z	0 ≤ d ≤ 31	Rd ← (Z)	–	1 (2)	2	Загрузка в регистр Rd содержимого ячейки SRAM, адресуемой с помощью 16-битного указателя Z (в модели AT90S1200 — 8-бит-ного указателя Z)
✂	ld Rd, Z+	0 ≤ d ≤ 31	Rd ← (Z) Z ← Z+1	–	1 (2)	2	Загрузка в регистр Rd содержимого ячейки SRAM, адресуемой с помощью 16-битного указателя Z, с последующим увеличением Z на 1
✂	ld Rd, -Z	0 ≤ d ≤ 31	Z ← Z–1 Rd ← (Z)	–	1 (2)	2	Уменьшение 16-битного указателя Z на 1 с последующей загрузкой в регистр Rd содержимого ячейки SRAM, адресуемой с помощью Z
✂	ldd Rd, Z+q	0 ≤ d ≤ 31 0 ≤ q ≤ 63	Rd ← (Z+q)	–	1 (2)	2	Загрузка в регистр Rd содержимого ячейки SRAM, адресуемой с помощью суммы q и содержимого 16-битного указателя Z
✂	lds Rd, k16	0 ≤ d ≤ 31 0 ≤ k16 ≤ 65535	Rd ← (k16)	–	2 (4)	3	Загрузка в регистр Rd содержимого ячейки SRAM по 16-битному адресу k16
✂	st X, Rr	0 ≤ r ≤ 31	(X) ← Rr	–	1 (2)	2	Загрузка в ячейку SRAM, адресуемую с помощью 16-битного указателя X, содержимого регистра Rr

Таблица 12.4. Продолжение

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
✂	st X+, Rr	0 ≤ r ≤ 31	(X) ← Rr X ← X+1	–	1 (2)	2	Загрузка в ячейку SRAM, адресуемую с помощью 16-битного указателя X, содержимого регистра Rr с последующим увеличением X на 1
✂	st -X, Rr	0 ≤ r ≤ 31	X ← X–1 (X) ← Rr	–	1 (2)	2	Уменьшение 16-битного указателя X на 1 с последующей загрузкой в ячейку SRAM, адресуемую с помощью X, содержимого Rr
✂	st Y, Rr	0 ≤ r ≤ 31	(Y) ← Rr	–	1 (2)	2	Загрузка в ячейку SRAM, адресуемую с помощью 16-битного указателя Y, содержимого регистра Rr
✂	st Y+, Rr	0 ≤ r ≤ 31	(Y) ← Rr Y ← Y+1	–	1 (2)	2	Загрузка в ячейку SRAM, адресуемую с помощью 16-битного указателя Y, содержимого регистра Rr с последующим увеличением X на 1
✂	st -Y, Rr	0 ≤ r ≤ 31	Y ← Y–1 (Y) ← Rr	–	1 (2)	2	Уменьшение 16-битного указателя Y на 1 с последующей загрузкой в ячейку SRAM, адресуемую с помощью Y, содержимого регистра Rr
✂	std Y+q, Rr	0 ≤ r ≤ 31 0 ≤ q ≤ 63	(Y+q) ← Rr	–	1 (2)	2	Загрузка в ячейку SRAM, адресуемую с помощью суммы q и содержимого 16-битного указателя Y, содержимого регистра Rr

Таблица 12.4. Продолжение

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
	st Z, Rr	0 ≤ r ≤ 31	(Z) ← Rr	–	1 (2)	2	Загрузка в ячейку SRAM, адресуемую с помощью 16-битного указателя Z (в модели AT90S1200 — 8-битного указателя Z), содержимого регистра Rr
✂	st Z+, Rr	0 ≤ r ≤ 31	(Z) ← Rr Z ← Z+1	–	1 (2)	2	Загрузка в ячейку SRAM, адресуемую с помощью 16-битного указателя Z, содержимого регистра Rr с последующим увеличением Z на 1
✂	st -Z, Rr	0 ≤ r ≤ 31	Z ← Z–1 (Z) ← Rr	–	1 (2)	2	Уменьшение 16-битного указателя Z на 1 с последующей загрузкой в ячейку SRAM, адресуемую с помощью Z, содержимого регистра Rr
✂	std Z+q, Rr	0 ≤ r ≤ 31 0 ≤ q ≤ 63	(Z+q) ← Rr	–	1 (2)	2	Загрузка в ячейку SRAM, адресуемую с помощью суммы q и содержимого 16-битного указателя Z, содержимого регистра Rr
✂	sts k16, Rr	0 ≤ r ≤ 31 0 ≤ k16 ≤ 65535	(k16) ← Rr	–	2 (4)	3	Загрузка в ячейку SRAM по 16-битному адресу k16 содержимого регистра Rr
✂	lpm	–	R0 ← (Z)	–	1 (2)	3	Загрузка в регистр R0 содержимого ячейки памяти Flash-EPROM, адресуемой с помощью 16-битного указателя Z

Таблица 12.4. Окончание

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
	in Rd, P	0 ≤ d ≤ 31 0 ≤ P ≤ 63	Rd ← P	–	1 (2)	1	Загрузка в регистр Rd содержимого порта ввода/вывода P
	out P, Rr	0 ≤ r ≤ 31 0 ≤ P ≤ 63	P ← Rr	–	1 (2)	1	Загрузка в порт ввода/вывода P содержимого регистра Rr
✂	pop Rd	0 ≤ d ≤ 31	Rd ← Стек	–	1 (2)	2	Загрузка в регистр Rd содержимого ячейки стека, адресуемой с помощью указателя стека
✂	push Rr	0 ≤ r ≤ 31	Стек ← Rr	–	1 (2)	2	Загрузка в ячейку стека, адресуемую с помощью указателя стека, содержимого регистра Rr
	bst Rd, b	0 ≤ d ≤ 31 0 ≤ b ≤ 7	T ← Rd(b)	T	1 (2)	1	Копирование в флаг T разряда b регистра Rd
	bld Rd, b	0 ≤ d ≤ 31 0 ≤ b ≤ 7	Rd(b) ← T	–	1 (2)	1	Копирование флага T в разряд b регистра Rd

Таблица 12.5. Арифметические и логические команды

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
	add Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd ← Rd + Rr	H, S, V, N, Z, C	1 (2)	1	Содержимое регистра Rr прибавляется к содержимому регистра Rd
	adc Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd ← Rd + Rr + C	H, S, V, N, Z, C	1 (2)	1	Содержимое регистра Rr и флаг переноса прибавляются к содержимому регистра Rd
✂	adiw Rdl, K6	dl ∈ {24, 26, 28, 30} 0 ≤ K6 ≤ 63	Rdh:Rdl ← Rdh:Rdl+K6	S, V, N, Z, C	1 (2)	1	Константа K6 прибавляется к содержимому регистровой пары Rdh:Rdl

Таблица 12.5. Продолжение

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (бит)	Тактовых циклов	Описание
	sub Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd ← Rd – Rr	H, S, V, N, Z, C	1 (2)	1	Содержимое регистра Rr вычитается из содержимого регистра Rd
	subi Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd ← Rd – K8	H, S, V, N, Z, C	1 (2)	1	8-битная константа K8 вычитается из содержимого регистра Rd
	sbc Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd ← Rd – Rr – C	H, S, V, N, Z, C	1 (2)	1	Содержимое регистра Rr и флаг переноса вычитаются из содержимого регистра Rd
	sbci Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd ← Rd – K8 – C	H, S, V, N, Z, C	1 (2)	1	8-битная константа K8 и флаг переноса вычитаются из содержимого регистра Rd
✂	sbiw Rdl, K6	dl ∈ {24, 26, 28, 30} 0 ≤ K6 ≤ 63	Rdh:Rdl ← Rdh:Rdl–K6	S, V, N, Z, C	1 (2)	2	Константа K6 вычитается из содержимого регистровой пары Rdh:Rdl
	and Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd ← Rd ∧ Rr	S, V=0, N, Z	1 (2)	1	Объединение логической операцией "И" содержимого регистров Rd и Rr
	andi Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd ← Rd ∧ K8	S, V=0, N, Z	1 (2)	1	Объединение логической операцией "И" содержимого регистра Rd и 8-битной константы K8
	or Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd ← Rd ∨ Rr	S, V=0, N, Z	1 (2)	1	Объединение логической операцией "ИЛИ" содержимого регистров Rd и Rr
	ori Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd ← Rd ∨ K8	S, V=0, N, Z	1 (2)	1	Объединение логической операцией "ИЛИ" содержимого регистра Rd и 8-битной константы K8

Таблица 12.5. Продолжение

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
	eor Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd ← Rd ⊕ Rr	S, V=0, N, Z	1 (2)	1	Объединение логической операцией "Исключающее ИЛИ" содержимого регистров Rd и Rr
	com Rd	0 ≤ d ≤ 31	Rd ← $FF – Rd	S, V=0, N, Z, C=1	1 (2)	1	Реализация дополнения до единицы содержимого регистра Rd
	neg Rd	0 ≤ d ≤ 31	Rd ← $00 – Rd	H, S, V, N, Z, C	1 (2)	1	Реализация дополнения до двух содержимого регистра Rd
	sbr Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd ← Rd ∨ K8	S, V=0, N, Z	1 (2)	1	Установка в регистре Rd разрядов, заданных с помощью K8
	cbr Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd ← Rd ∧ ($FF – K8)	S, V=0, N, Z	1 (2)	1	Сброс в регистре Rd разрядов, заданных с помощью K8
	inc Rd	0 ≤ d ≤ 31	Rd ← Rd +1	S, V, N, Z	1 (2)	1	Прибавление 1 к содержимому регистра Rd
	dec Rd	0 ≤ d ≤ 31	Rd ← Rd – 1	S, V, N, Z	1 (2)	1	Вычитание 1 из содержимого регистра Rd
	tst Rd	0 ≤ d ≤ 31	Rd ← Rd ∧ Rd	S, V=0, N, Z	1 (2)	1	Проверка, является ли содержимое регистра Rd нулевым или отрицательным
	clr Rd	0 ≤ d ≤ 31	Rd ← Rd ⊕ Rd	S=0, V=0, N=0, Z=1	1 (2)	1	Очистка регистра Rd
	ser Rd	16 ≤ d ≤ 31	Rd ← $FF	–	1 (2)	1	Загрузка в регистр Rd значения $FF
	cp Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd – Rr	H, S, V, N, Z, C	1 (2)	1	Сравнение содержимого регистров Rd и Rr (содержимое обоих регистров не меняется)

Таблица 12.5. Окончание

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
	cpc Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	Rd – Rr – C	H, S, V, N, Z, C	1 (2)	1	Сравнение содержимого регистров Rd и Rr с учетом флага переноса
	cpi Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd – K8	H, S, V, N, Z, C	1 (2)	1	Сравнение содержимого регистра Rd с 8-битной константой K8
	cpse Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	если Rd =Rr, то PC←PC+2(3) иначе PC←PC+1	–	1 (2)	1/2/3	Содержимое регистров Rd и Rr сравнивается, и в том случае, если Rd=Rr, следующая команда пропускается

Таблица 12.6. Команды перехода

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
	rjmp k	-2K ≤ k ≤ 2K	PC←PC+k+1	–	1 (2)	2	Относительный переход по адресу в пределах диапазона от PC–2K до PC+2K (слов)
✂	ijmp	–	PC←Z(15–0)	–	1 (2)	2	Косвенный переход по адресу, на который указывает указатель Z
✂	jmp k	0 ≤ k ≤ 4M	PC ← k	–	2 (4)	3	Абсолютный переход по адресу k
	rcall k	-2K ≤ k ≤ 2K	PC←PC+k+1	–	1 (2)	3	Вызов подпрограммы в пределах диапазона от PC–2K до PC+2K (слов)
✂	icall	–	PC←Z(15–0)	–	1 (2)	3	Косвенный вызов подпрограммы, на которую указывает указатель Z
✂	call k	0 ≤ k ≤ 4M	PC ← k	–	2 (4)	4	Вызов подпрограммы по адресу k
	ret	–	PC ← Стек	–	1 (2)	4	Возврат из подпрограммы, адрес извлекается из стека

Таблица 12.6. Продолжение

Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
reti	–	PC ← Стек	I = 1	1 (2)	4	Возврат из подпрограммы обработки прерывания по адресу, извлеченному из стека; устанавливается флаг общего разрешения прерываний
cpse Rd, Rr	0 ≤ d ≤ 31 0 ≤ r ≤ 31	если Rd =Rr, то PC←PC+2(3) иначе PC←PC+1	–	1 (2)	1/2/3	Содержимое регистров Rd и Rr сравнивается, и в том случае, если Rd=Rr, следующая команда пропускается
sbrc Rr, b	0 ≤ r ≤ 31 0 ≤ b ≤ 7	если Rr(b)=0, то PC←PC+2(3) иначе PC←PC+1	–	1 (2)	1/2/3	Если разряд b регистра Rr содержит лог. 0, то следующая команда пропускается
sbrs Rr, b	0 ≤ r ≤ 31 0 ≤ b ≤ 7	если Rr(b)=1, то PC←PC+2(3) иначе PC←PC+1	–	1 (2)	1/2/3	Если разряд b регистра Rr содержит лог. 1, то следующая команда пропускается
sbic P, b	0 ≤ P ≤ 31 0 ≤ b ≤ 7	если I/O(b)=0, то PC←PC+2(3) иначе PC←PC+1	–	1 (2)	1/2/3	Если разряд b регистра ввода/вывода P содержит лог. 0, то следующая команда пропускается
sbis P, b	0 ≤ P ≤ 31 0 ≤ b ≤ 7	если I/O(b)=1, то PC←PC+2(3) иначе PC←PC+1	–	1 (2)	1/2/3	Если разряд b регистра ввода/вывода P содержит лог. 1, то следующая команда пропускается
brbc s, k7	0 ≤ s ≤ 7 -64 ≤ k7 ≤ +63	если SREG(s)=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если разряд s регистра SREG содержит лог. 0, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Таблица 12.6. Продолжение

Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
brbs s, k7	0 ≤ s ≤ 7 -64 ≤ k7 ≤ +63	если SREG(s)=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если разряд s регистра SREG содержит лог. 1, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
brne k7	-64 ≤ k7 ≤ +63	если Z=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если сброшен нулевой флаг, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
breq k7	-64 ≤ k7 ≤ +63	если Z=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если установлен нулевой флаг, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
brcc k7	-64 ≤ k7 ≤ +63	если C=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если сброшен флаг переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
brcs k7	-64 ≤ k7 ≤ +63	если C=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если установлен флаг переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
brsh k7	-64 ≤ k7 ≤ +63	если C=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если сброшен флаг переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Таблица 12.6. Продолжение

Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
brlo k7	-64 ≤ k7 ≤ +63	если C=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если установлен флаг переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
brpl k7	-64 ≤ k7 ≤ +63	если N=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если сброшен флаг отрицательного результата, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
brmi k7	-64 ≤ k7 ≤ +63	если N=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если установлен флаг отрицательного результата, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
brge k7	-64 ≤ k7 ≤ +63	если S=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если сброшен флаг знака, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
brlt k7	-64 ≤ k7 ≤ +63	если S=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если установлен флаг знака, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
brhc k7	-64 ≤ k7 ≤ +63	если H=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если сброшен флаг половинного переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Таблица 12.6. Продолжение

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
	brhs k7	-64 ≤ k7 ≤ +63	если H=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если установлен флаг половинного переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
	brtc k7	-64 ≤ k7 ≤ +63	если T=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если сброшен флаг копирования, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
	brts k7	-64 ≤ k7 ≤ +63	если T=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если установлен флаг копирования, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
	brvc k7	-64 ≤ k7 ≤ +63	если V=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если сброшен флаг переполнения, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
	brvs k7	-64 ≤ k7 ≤ +63	если V=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если установлен флаг переполнения, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
	brid k7	-64 ≤ k7 ≤ +63	если I=0, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если сброшен флаг общего разрешения прерываний, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Таблица 12.6. Окончание

Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
brie k7	-64 ≤ k7 ≤ +63	если I=1, то PC←PC+k7+1 иначе PC←PC+1	–	1 (2)	1 (2)	Если установлен флаг общего разрешения прерываний, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Таблица 12.7. Команды поразрядных операций

Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
lsl Rd	0 ≤ Rd ≤ 31	Rd(n+1)←Rd(n), C ← Rd(7), Rd(0) ← 0	H, S, V, N, Z, C	1 (2)	1	Все разряды регистра Rd сдвигаются на одну позицию влево; флагу переноса присваивается значение разряда 7, а в разряд 0 записывается лог. 0
lsr Rd	0 ≤ Rd ≤ 31	Rd(n)←Rd(n+1), C ← Rd(0), Rd(7) ← 0	S, V, N=0, Z, C	1 (2)	1	Все разряды регистра Rd сдвигаются на одну позицию вправо; флагу переноса присваивается значение разряда 0, а в разряд 7 записывается лог. 0
rol Rd	0 ≤ Rd ≤ 31	Rd(n+1)←Rd(n), Rd(0) ← C, C ← Rd(7)	H, S, V, N, Z, C	1 (2)	1	Все разряды регистра Rd сдвигаются на одну позицию влево; в разряд 0 записывается флаг переноса, после чего разряд 7 сохраняется как флаг переноса

Таблица 12.7. Продолжение

	Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
	ror Rd	0 ≤ Rd ≤ 31	Rd(n)←Rd(n+1), Rd(7) ← C, C ← Rd(0)	S, V, N, Z, C	1 (2)	1	Все разряды регистра Rd сдвигаются на одну позицию вправо; в разряд 7 записывается флаг переноса, после чего разряд 0 сохраняется как флаг переноса
	asr Rd	0 ≤ Rd ≤ 31	Rd(n)←Rd(n+1), n = 0…6, C ← Rd(0)	S, V, N, Z, C	1 (2)	1	Разряды 0…6 регистра Rd сдвигаются на одну позицию вправо; разряд 0 сохраняется как флаг переноса; разряд 7 не изменяется
	swap Rd	0 ≤ Rd ≤ 31	Rd(0…3) ↔ Rd(4…7)	–	1 (2)	1	Старший и младший полубайты регистра Rd меняются местами
	bclr s	0 ≤ s ≤ 7	SREG(s) ← 0	I, T, H, S, V, N, Z, C	1 (2)	1	Сброс разряда s в регистре SREG
	bset s	0 ≤ s ≤ 7	SREG(s) ← 1	I, T, H, S, V, N, Z, C	1 (2)	1	Установка разряда s в регистре SREG
	sbr Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd ← Rd ∨ K8	S, V=0, N, Z	1 (2)	1	Установка в регистре Rd разрядов, заданных с помощью K8
	cbr Rd, K8	16 ≤ d ≤ 31 0 ≤ K8 ≤ 255	Rd ← Rd ∧ ($FF – K8)	S, V=0, N, Z	1 (2)	1	Сброс в регистре Rd разрядов, заданных с помощью K8
	cbi P, b	0 ≤ P ≤ 31 0 ≤ b ≤ 7	I/O(P,b) ← 0	–	1 (2)	2	Запись лог. 0 в разряд b регистра ввода/вывода P
	sbi P, b	0 ≤ P ≤ 31 0 ≤ b ≤ 7	I/O(P,b) ← 1	–	1 (2)	2	Запись лог. 1 в разряд b регистра ввода/вывода P
	bst Rd, b	0 ≤ d ≤ 31 0 ≤ b ≤ 7	T ← Rd(b)	T	1 (2)	1	Копирование в флаг T разряда b регистра Rd

Таблица 12.7. Продолжение

Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
bld Rd, b	0 ≤ d ≤ 31 0 ≤ b ≤ 7	Rd(b) ← T	–	1 (2)	1	Копирование флага T в разряд b регистра Rd
clc	–	C ← 0	C = 0	1 (2)	1	Сброс флага переноса в регистре SREG
sec	–	C ← 1	C = 1	1 (2)	1	Установка флага переноса в регистре SREG
cln	–	N ← 0	N = 0	1 (2)	1	Сброс флага отрицательного результата в регистре SREG
sen	–	N ← 1	N = 1	1 (2)	1	Установка флага отрицательного результата в регистре SREG
clz	–	Z ← 0	Z = 0	1 (2)	1	Сброс нулевого флага в регистре SREG
sez	–	Z ← 1	Z = 1	1 (2)	1	Установка нулевого флага в регистре SREG
cli	–	I ← 0	I = 0	1 (2)	1	Сброс флага общего разрешения прерываний в регистре SREG
sei	–	I ← 1	I = 1	1 (2)	1	Установка флага общего разрешения прерываний в регистре SREG
cls	–	S ← 0	S = 0	1 (2)	1	Сброс флага знака в регистре SREG
ses	–	S ← 1	S = 1	1 (2)	1	Установка флага знака в регистре SREG
clv	–	V ← 0	V = 0	1 (2)	1	Сброс флага переполнения в регистре SREG
sev	–	V ← 1	V = 1	1 (2)	1	Установка флага переполнения в регистре SREG

Таблица 12.7. Окончание

Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
clt	–	T ← 0	T = 0	1 (2)	1	Сброс флага копирования в регистре SREG
set	–	T ← 1	T = 1	1 (2)	1	Установка флага копирования в регистре SREG
clh	–	H ← 0	H = 0	1 (2)	1	Сброс флага половинного переноса в регистре SREG
seh	–	H ← 1	H = 1	1 (2)	1	Установка флага половинного переноса в регистре SREG

Таблица 12.8. Другие команды

Мнемоника	Операнды	Операция	Влияние на флаги	Слов (байт)	Тактовых циклов	Описание
nop	–	–	–	1 (2)	1	Команда без определенной функции (нет операции)
sleep	–	CPU ← "Спящий" режим	–	1 (2)	1	Переход в "спящий" режим (см. главу 3)
wdr	–	Сброс сторожевого таймера	–	1 (2)	1	См. главу 5

Описание команд микроконтроллеров AVR

ADC

Название: Add with Carry — сложение с учетом переноса
Синтаксис: `adc Rd, Rr`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Rd ← Rd + Rr + C
Операнды: $0 \le d \le 31; 0 \le r \le 31$
16-битный код операции: `0001 11rd dddd rrrr`
Влияние на флаги: H, S, V, N, Z, C
Описание: Содержимое регистра Rr и флаг переноса прибавляются к содержимому регистра Rd
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
;Сложение байтов r1:r0 и r3:r2
0c20 add r2,r0 ; Складываем младшие байты
1c31 adc r3,r1 ; Складываем старшие байты
               ; с учетом переноса
```

ADD

Название: Add without Carry — сложение без учета переноса

Синтаксис: `add Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd + Rr

Операнды: $0 \le d \le 31$; $0 \le r \le 31$

16-битный код операции: `0000 11rd dddd rrrr`

Влияние на флаги: H, S, V, N, Z, C

Описание: Содержимое регистра Rr прибавляется к содержимому регистра Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
;Сложение байтов r1:r0 и r3:r2
0c20 add r2,r0 ; Складываем младшие байты
1c31 adc r3,r1 ; Складываем старшие байты
               ; с учетом переноса
```

ADIW

Название: Add Immediate to Word — сложение непосредственного значения со словом

Синтаксис: `adiw Rdl, K6`

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: Rdh:Rdl ← Rdh:Rdl + K6

Операнды: dl ∈ {24, 26, 28, 30}; $0 \le K6 \le 63$

16-битный код операции: `1001 0110 KKdd KKKK`

Влияние на флаги: S, V, N, Z, C

Описание: Константа K6 прибавляется к содержимому регистровой пары Rdh:Rdl. Эта команда работает с четырьмя самыми старшими регистровыми парами регистрового файла, и главным образом используется в операциях с указателями

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
9601 adiw r24,1  ; Увеличиваем на 1
                 ; содержимое пары r25:r24
96ff adiw r30,63 ; Прибавляем 63 к
                 ; указателю Z (r31:r30)
```

AND

Название: Логическое "И"

Синтаксис: `and Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd ∧ Rr

Операнды: 0 ≤ d ≤ 31; 0 ≤ r ≤ 31

16-битный код операции: `0010 00rd dddd rrrr`

Влияние на флаги: S, V = 0, N, Z

Описание: Связывание логической операцией "И" содержимого регистров Rd и Rr. Результат записывается в регистр Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
2023 and r2,r3   ; r2 "И" r3
e001 ldi r16,1   ; Загружаем в r16 битовую
                 ; маску 0000 0001
2220 and r2,r16  ; Во все разряды r2, кроме
                 ; нулевого, записан лог. 0
```

ANDI

Название: Logical AND with Immediate — логическое "И" с непосредственным значением

Синтаксис: andi Rd, K8

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd ∧ K8

Операнды: 0 ≤ d ≤ 31; 0 ≤ K8 ≤ 255

16-битный код операции: 0111 KKKK dddd KKKK

Влияние на флаги: S, V = 0, N, Z

Описание: Связывание логической операцией "И" содержимого регистра Rd и 8-битной константы K8. Результат записывается в регистр Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
701f andi r17,$0F ; Старший полубайт r17
                  ; сбрасывается в лог. 0
7120 andi r18,$10 ; Все разряды r18, кроме
                  ; четвертого, - в лог. 0
7a3a andi r19,$AA ; В разряды 0, 2, 4, 6
                  ; r19 записывется лог. 0
```

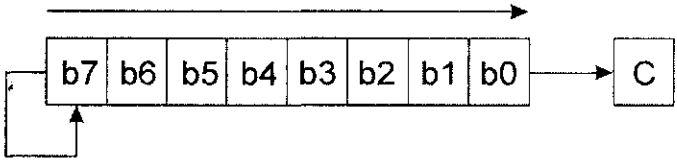
ASR

Название: Arithmetic Shift Right — арифметический сдвиг вправо

Синтаксис: `asr Rd`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция:



Rd(n) ← Rd(n+1), n = 0...6, C ← Rd(0)

Операнды: 0 ≤ Rd ≤ 31

16-битный код операции: `1001 010d dddd 0101`

Влияние на флаги: S, V, N, Z, C

Описание: Разряды 0...6 регистра Rd сдвигаются на одну позицию вправо; разряд 0 сохраняется как флаг переноса; разряд 7 не изменяется. Эта команда может использоваться для деления на два байтов со знаковым разрядом без изменения этого разряда. Флаг переноса может использоваться для округления результата

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
e110 ldi r17,$10 ; в r17 загружаем 16
9515 asr r17   ; делим содержимое r17 на 2
ef2c ldi r18,$FC ; в r18 загружаем -4
9525 asr r18 ; делим содержимое r18 на 2
```

BCLR

Название: Bit Clear in SREG — очистка разряда в SREG

Синтаксис: `bclr s`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: SREG(s) ← 0

Операнды: 0 ≤ s ≤ 7

16-битный код операции: `1001 0100 1sss 1000`

Влияние на флаги: I, T, H, S, V, N, Z, C

Описание: Сброс разряда s в регистре SREG

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
9488 bclr 0 ; сброс флага переноса
94f8 bclr 7 ; запрет всех прерываний
```

BLD

Название: Bit load from T-Flag in SREG to a Bit in Register — загрузка в разряд регистра флага T из регистра SREG

Синтаксис: `bld Rd, b`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd(b) ← T

Операнды: $0 \le d \le 31$; $0 \le b \le 7$

16-битный код операции: `1111 100d dddd 0bbb`

Влияние на флаги: Нет

Описание: Копирование флага T в разряд b регистра Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
fa12 bst r1,2 ; переносим разряд 2 из r1 в
              ; флаг T
f804 bld r0,4 ; копируем флаг T в разряд 4
              ; регистра r0
```

BRBC

Название: Branch if Bit in SREG is cleared — переход, если сброшен разряд в регистре SREG

Синтаксис: `brbc s, k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если SREG(s) = 0, то PC ← PC + k7 + 1, иначе PC ← PC +1

Операнды: $0 \le s \le 7$; $-64 \le k7 \le +63$

16-битный код операции: `1111 01kk kkkk ksss`

Влияние на флаги: Нет

Описание: Если разряд s регистра SREG содержит лог. 0, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (условие не исполняется) или 2 (условие исполняется)

Пример использования:

```
3045 cpi r20,5     ; Сравниваем r20 с 5
f401 brbc 1,NotEq ; Переход, если флаг Z=0
NotEq:
0000 nop ; Точка перехода, если (r20) ≠ 0
```

BRBS

Название: Branch if Bit in SREG is set — переход, если установлен разряд в регистре SREG

Синтаксис: `brbs s, k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если SREG(s) = 1, то PC ← PC + k7 + 1, иначе PC ← PC +1

Операнды: $0 \le s \le 7$; $-64 \le k7 \le +63$

16-битный код операции: `1111 00kk kkkk ksss`

Влияние на флаги: Нет

Описание: Если разряд s регистра SREG содержит лог. 1, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (условие не исполняется) или 2 (условие исполняется)

Пример использования:

```
fa03 bst r0,3     ; Копируем разряд 3
                  ; регистра r0 в флаг T
f006 brbs 6,BSet ; Переход, если флаг T=1
BSet:
0000 nop         ; Точка перехода, если T=1
```

BRCC

Название: Branch if Carry is cleared — переход, если сброшен флаг переноса

Синтаксис: `brcc k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг C = 0, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: $-64 \le k7 \le +63$ (дополнение до двух)

16-битный код операции: `1111 01kk kkkk k000`

Влияние на флаги: Нет

Описание: Если сброшен флаг переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг C = лог. 1) или 2 (флаг C = лог. 0)

Пример использования:

```
0f67 add r22,r23 ;Прибавляем (r23) к (r22)
f400 brcc NoCarry ; Переход, если C=0
NoCarry:
0000 nop ; Точка перехода, если C=0
```

BRCS

Название: Branch if Carry is set — переход, если установлен флаг переноса

Синтаксис: `brcs k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг C = 1, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 00kk kkkk k000`

Влияние на флаги: нет

Описание: Если установлен флаг переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг C = лог. 0) или 2 (флаг C = лог. 1)

Пример использования:

```
3348 cpi r20,56 ; Сравниваем r20 с 56
f000 brcs Cset  ; Переход, если флаг c=1
Cset:
0000 nop ; Точка перехода, если (r20)<56
```

BREQ

Название: Branch if Equal — переход, если равно

Синтаксис: `breq k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг Z = 1, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 00kk kkkk k001`

Влияние на флаги: Нет

Описание: Если установлен нулевой флаг, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов). Нулевой флаг после выполнения операции вычитания или сравнения устанавливается только в том случае, если результат равен $00 (то есть, содержимое обоих операндов равно)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг Z = лог. 0) или 2 (флаг Z = лог. 1)

Пример использования:

```
1401 cp r0,r1     ; Сравниваем (r0) и (r1)
f1f1 breq Gleich ; Переход, если Z = лог.1
.org 64
Gleich:
0000 nop  ; Точка перехода, если (r0)=(r1)
```

BRGE

Название: Branch if Greater or Equal — переход, если больше или равно (с учетом знака)

Синтаксис: `brge k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг S = 0, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 01kk kkkk k100`

Влияние на флаги: Нет

Описание: Если сброшен флаг знака, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов). Флаг знака после выполнения операции вычитания или сравнения сбрасывается только в том случае, если содержимое первого операнда со знаком больше или равно содержимого второго операнда со знаком

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг S = лог. 1) или 2 (флаг S = лог. 0)

Пример использования:

```
1401 cp r0,r1  ; Сравниваем (r0) и (r1)
f5ec brge GrGl ; Переход, если S = 0
.org 64
Gleich:
0000 nop ; Точка перехода, если (r0)>=(r1)
```

BRHC

Название: Branch if Half Carry Flag is cleared — переход, если сброшен флаг половинного переноса

Синтаксис: `brhc k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг H = 0, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 01kk kkkk k101`

Влияние на флаги: Нет

Описание: Если сброшен флаг половинного переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг H = лог. 1) или 2 (флаг H = лог. 0)

Пример использования:

```
f57d brhc HCclear ; Переход, если флаг H=0
.org 48
HCclear:
0000 nop ; Точка перехода
```

BRHS

Название: Branch if Half Carry Flag is set — переход, если установлен флаг половинного переноса

Синтаксис: `brhs k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг H = 1, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 00kk kkkk k101`

Влияние на флаги: Нет

Описание: Если установлен флаг половинного переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг H = лог. 0) или 2 (флаг H = лог. 1)

Пример использования:

```
f17d brhs HCset ; Переход, если флаг H=1
.org 48
HCset:
0000 nop ; Точка перехода
```

BRID

Название: Branch if Global Interrupt is Disabled — переход, если активен общий запрет прерываний

Синтаксис: `brid k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг I = 0, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 01kk kkkk k111`

Влияние на флаги: Нет

Описание: Если сброшен флаг I (все прерывания запрещены), то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг I = лог. 1) или 2 (флаг I = лог. 0)

Пример использования:

```
f5f7 brid IntDis ; Переход, если
                 ; прерывания запрещены
.org 63
IntDis:
0000 nop ; Точка перехода
```

BRIE

Название: Branch if Global Interrupt is Enabled — переход, если активно общее разрешение прерываний

Синтаксис: `brie k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг I = 1, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 00kk kkkk k111`

Влияние на флаги: Нет

Описание: Если установлен флаг I (общее разрешение прерываний), то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг I = лог. 0) или 2 (флаг I = лог. 1)

Пример использования:

```
f1f7 brie IntEna ; Переход, если
                 ; прерывания разрешены
.org 63
IntEna:
0000 nop ; Точка перехода
```

BRLO

Название: Branch if Lower — переход, если меньше (без учета знака)

Синтаксис: `brlo k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг C = 1, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 00kk kkkk k000`

Влияние на флаги: Нет

Описание: Если установлен флаг переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов). Флаг переноса в результате выполнения операции вычитания или сравнения может быть установлен только в том случае, если содержимое первого беззнакового операнда меньше, чем содержимое второго беззнакового операнда

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг C = лог. 0) или 2 (флаг C = лог. 1)

Пример использования:

```
27bb clr r27 ; (r27) = 0
Loop:
95b3 inc r27    ; Инкрементируем r27
30b5 cpi r27,5 ; Сравниваем (r27) с 5
f3e8 brlo Loop ; Переход, если (r27) < 5
0000 nop       ; Выход из цикла
```

BRLT

Название: Branch if Less Then — переход, если меньше (с учетом знака)

Синтаксис: `brlt k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг S = 1, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 00kk kkkk k100`

Влияние на флаги: Нет

Описание: Если установлен флаг знака, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов). Флаг знака в результате выполнения операции вычитания или сравнения может быть установлен только в том случае, если содержимое первого операнда со знаком меньше, чем содержимое второго операнда со знаком

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг S = лог. 0) или 2 (флаг S = лог. 1)

Пример использования:

```
1401 cp r0,r1  ; Сравниваем r0 и r1
f1f4 brlt Less ; Переход, если флаг S=1
.org 64
Less:
0000 nop ; Точка перехода, если (r0)<(r1)
```

BRMI

Название: Branch if Minus — переход, если минус

Синтаксис: `brmi k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг N = 1, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 00kk kkkk k010`

Влияние на флаги: нет

Описание: Если установлен флаг отрицательного результата, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг N = лог. 0) или 2 (флаг N = лог. 1)

Пример использования:

```
5034 subi r19,4    ; Вычитаем 4 из r19
f0b2 brmi Negativ ; Переход, если получен
                  ; отрицательный результат
.org 24
Negativ:
0000 nop ; Точка перехода
```

BRNE

Название: Branch if Not Equal — переход, если не равно
Синтаксис: `brne k7`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Если флаг Z = 0, то PC ← PC + k7 + 1, иначе PC ← PC + 1
Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)
16-битный код операции: `1111 01kk kkkk k001`
Влияние на флаги: Нет
Описание: Если сброшен нулевой флаг, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов). Нулевой флаг в результате выполнения операции вычитания или сравнения может быть сброшен только в том случае, если результат операции не равен $00 (то есть, содержимое двух операндов не равно)
Слов (байт): 1 (2)
Тактовых циклов: 1 (флаг Z = лог. 1) или 2 (флаг Z = лог. 0)
Пример использования:

```
27bb clr r27    ; (r27) = 0
Loop:
95b3 inc r27    ; Инкрементируем r27
30b5 cpi r27,5 ; Сравниваем (r27) с 5
f7e9 brne Loop ; Переход, если (r27) ≠ 5
0000 nop        ; Выход из цикла
```

BRPL

Название: Branch if Plus — переход, если плюс
Синтаксис: `brpl k7`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Если флаг N = 0, то PC ← PC + k7 + 1, иначе PC ← PC + 1
Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)
16-битный код операции: `1111 01kk kkkk k010`
Влияние на флаги: Нет
Описание: Если сброшен флаг отрицательного результата, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)
Слов (байт): 1 (2)
Тактовых циклов: 1 (флаг N = лог. 1) или 2 (флаг N = лог. 0)
Пример использования:

```
5034 subi r19,4    ; Вычитаем 4 из r19
f0b2 brpl Positiv ; Переход, если получен
                  ; положительный результат
.org 24
Positiv:
0000 nop ; Точка перехода
```

BRSH

Название: Branch if Same or Higher — переход, если больше или равно (без учета знака)

Синтаксис: `brsh k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг C = 0, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 01kk kkkk k000`

Влияние на флаги: Нет

Описание: Если сброшен флаг переноса, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов). Флаг переноса в результате выполнения операции вычитания или сравнения может быть сброшен только в том случае, если содержимое первого беззнакового операнда больше или равно содержимого второго беззнакового операнда

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг C = лог. 1) или 2 (флаг C = лог. 0)

Пример использования:

```
5034 subi r19,4 ; Вычитаем 4 из r19 f4b0
brsh SamHi      ; Переход, если (r19) >= 4
.org 24
SamHi:
0000 nop
```

BRTC

Название: Branch if T-Flag is Cleared – переход, если сброшен флаг T

Синтаксис: `brtc k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг T = 0, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 01kk kkkk k110`

Влияние на флаги: Нет

Описание: Если сброшен флаг T, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг T = лог. 1) или 2 (флаг T = лог. 0)

Пример использования:

```
fa35 bst r3,5 ; Копируем разряд 5 регистра
              ; r3 в флаг T
f4f6 brtc Tclear ; Переход, если флаг T=0
.org 32
Tclear:
0000 nop
```

BRTS

Название: Branch if T-Flag is Set — переход, если установлен флаг T

Синтаксис: `brts k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг T = 1, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 00kk kkkk k110`

Влияние на флаги: Нет

Описание: Если установлен флаг T, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг T = лог. 0) или 2 (флаг T = лог. 1)

Пример использования:

```
fa35 bst r3,5 ; Копируем разряд 5 регистра
              ; r3 в флаг T
f0f6 brts Tset ; Переход, если флаг T=1
.org 32
Tset:
0000 nop
```

BRVC

Название: Branch if Overflow is Cleared — переход, если сброшен флаг переполнения

Синтаксис: `brvc k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг V = 0, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 01kk kkkk k011`

Влияние на флаги: Нет

Описание: Если сброшен флаг переполнения, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг V = лог. 1) или 2 (флаг V = лог. 0)

Пример использования:

```
0c34 add r3,r4    ; Складываем r3 и r4
f533 brvc NoOver  ; Если нет переполнения,
                  ; то переходим
.org 40
NoOver:
0000 nop
```

BRVS

Название: Branch if Overflow is Set — переход, если установлен флаг переполнения

Синтаксис: `brvs k7`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если флаг V = 1, то PC ← PC + k7 + 1, иначе PC ← PC + 1

Операнды: -64 ≤ k7 ≤ +63 (дополнение до двух)

16-битный код операции: `1111 00kk kkkk k011`

Влияние на флаги: Нет

Описание: Если установлен флаг переполнения, то выполняется относительный переход по адресу в диапазоне от PC–64 до PC+63 (слов)

Слов (байт): 1 (2)

Тактовых циклов: 1 (флаг V = лог. 0) или 2 (флаг V = лог. 1)

Пример использования:

```
0c34 add r3,r4    ; Складываем r3 и r4
f133 brvs OverFl  ; Если есть переполнение,
                  ; то переходим
.org 40
OverFl:
0000 nop
```

BSET

Название: Bit Set in SREG — установлен разряд в регистре SREG

Синтаксис: `bset s`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: SREG(s) ← 1

Операнды: 0 ≤ s ≤ 7

16-битный код операции: `1001 0100 0sss 1000`

Влияние на флаги: I, T, H, S, V, N, Z, C

Описание: Установка разряда s в регистре SREG

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
9468 bset 6 ; Установка флага T
9478 bset 7 ; Общее разрешение прерывания
```

BST

Название: Bit Store from Bit in Register to T-Flag in SREG — сохранение разряда регистра как флага регистра SREG

Синтаксис: `bst Rd, b`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: T ← Rd(b)

Операнды: 0 ≤ d ≤ 31; 0 ≤ b ≤ 7

16-битный код операции: `1111 101d dddd Xbbb`

Влияние на флаги: T

Описание: Копирование в флаг T разряда b регистра Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
fa12 bst r1,2 ; Копируем разряд 2 регистра
              ; r1 в флаг T
f804 bld r0,4 ; Копируем флаг T в разряд 4
              ; регистра r0
```

CALL

Название: Дальний вызов подпрограммы

Синтаксис: `call k16 (k22)`

Реализована в моделях: Ни в одной модели базовой серии семейства AVR

Операция: PC ← k16 (k22)

Операнды: 0 ≤ k ≤ 64K (0 ≤ k ≤ 4M)

32-битный код операции: `1001 010k kkkk 111k kkkk kkkk kkkk kkkk`

Влияние на флаги: Нет

Описание: Вызов подрограммы по абсолютному адресу k16 (в микроконтроллерах с 16-разрядной шиной адреса) или k22 (в микроконтроллерах с 22-разрядной шиной адреса). Адрес возврата (также 16- или 22-разрядный) — команды, следующей после команды `call` — сохраняется в стеке

Слов (байт): 2 (4)

Тактовых циклов: 4

Пример использования:

```
.device AT90S8515
warning: 'CALL' not supported
000080 940e 0100 call UP1 ; Вызов UP1
000082 0000 nop    ; Продолжение программы
.org 256
UP1:
000100 9468 bset 6 ; Установка флага T
000101 9508 ret
```

CBI

Название: Clear Bit in I/O-Register — очистка разряда в регистре ввода/вывода

Синтаксис: `cbi P, b`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: I/O(P,b) ← 0

Операнды: $0 \le P \le 31$; $0 \le b \le 7$

16-битный код операции: `1001 1000 pppp pbbb`

Влияние на флаги: Нет

Описание: Запись лог. 0 в разряд b регистра ввода/вывода P. Эта команда может применяться только для регистров ввода/ вывода 0–31 ($00–$1F)

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:
```
9897 cbi $12,7 ; Запись лог. 0 в разряд 7
               ; порта D
```

CBR

Название: Clear Bit(s) in Register — очистка разряда(ов) в регистре

Синтаксис: `cbr Rd, K8`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd ∧ ($FF – K8)

Операнды: $16 \le d \le 31$; $0 \le K8 \le 255$

16-битный код операции: `0111 KKKK dddd KKKK`

Влияние на флаги: Нет

Описание: Сброс в регистре Rd разрядов, заданных с помощью K8, с применением логической операции "И" с дополненной константой K8. Эта команда применяется только для регистров r16–r31

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:
```
700f cbr r16,$F0 ;Сброс старшего полубайта
7f2e cbr r18,1   ;Сброс разряда 0 в r18
```

CLC

Название: Clear Carry Flag — сброс флага переноса
Синтаксис: `clc`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: C ← 0
Операнды: Нет
16-битный код операции: `1001 0100 1000 1000`
Влияние на флаги: C = 0
Описание: Сброс флага переноса в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
0c00 add r0,r0 ; Умножаем r0 на 2
9488 clc       ; Сбрасываем флаг переноса
```

CLH

Название: Clear Half Carry Flag — сброс флага половинного переноса
Синтаксис: `clh`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: H ← 0
Операнды: Нет
16-битный код операции: `1001 0100 1101 1000`
Влияние на флаги: H = 0
Описание: Сброс флага половинного переноса в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
94d8 clh ;Сброс флага половинного переноса
```

CLI

Название: Clear Global Interrupt Flag — сброс флага общего разрешения прерываний
Синтаксис: `cli`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: I ← 0
Операнды: Нет
16-битный код операции: `1001 0100 1111 1000`
Влияние на флаги: I = 0
Описание: Сброс флага общего разрешения прерываний в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
94f8 cli   ; Сброс флага I
```

CLN

Название: Clear Negative Flag — сброс флага отрицательного результата
Синтаксис: `cln`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: N ← 0
Операнды: Нет
16-битный код операции: `1001 0100 1010 1000`
Влияние на флаги: N = 0
Описание: Сброс флага отрицательного результата в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
94a8 cln    ; Сброс флага N
```

CLR

Название: Clear Register — очистка регистра
Синтаксис: `clr`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Rd ← Rd ⊕ Rd
Операнды: 0 ≤ d ≤ 31
16-битный код операции: `0010 01dd dddd dddd`
Влияние на флаги: S=0, V=0, N=0, Z=1
Описание: Благодаря применению логической операции "Исключающее ИЛИ" содержимого регистра самому к себе, все разряды этого регистра устанавливаются в лог. 0
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
2700 clr r16 ; Очищаем r16
27ff clr r31 ; Очищаем r31
```

CLS

Название: Clear Signed Flag — сброс флага знака
Синтаксис: `cls`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: S ← 0
Операнды: Нет
16-битный код операции: `1001 0100 1100 1000`
Влияние на флаги: S = 0
Описание: Сброс флага знака в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
0c23 add r2,r3 ; Прибавляем r3 к r2
94c8 cls       ; Сбрасываем флаг знака
```

CLT

Название: Clear Transfer Flag — сброс флага копирования
Синтаксис: `clt`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: T ← 0
Операнды: Нет
16-битный код операции: `1001 0100 1110 1000`
Влияние на флаги: T = 0
Описание: Сброс флага копирования в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
94e8 clt     ; Сброс флага копирования
```

CLV

Название: Clear Overflow Flag — сброс флага переполнения
Синтаксис: `clv`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: V ← 0
Операнды: Нет
16-битный код операции: `1001 0100 1011 1000`
Влияние на флаги: V = 0
Описание: Сброс флага переполнения в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
94b8 clv       ; Сброс флага переполнения
```

CLZ

Название: Clear Zero Flag — сброс нулевого флага
Синтаксис: `clz`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Z ← 0
Операнды: Нет
16-битный код операции: `1001 0100 1001 1000`
Влияние на флаги: Z = 0
Описание: Сброс нулевого флага в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
9498 clz       ; Сброс нулевого флага
```

COM

Название: One's Complement — дополнение до единицы

Синтаксис: `com Rd`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← $FF – Rd

Операнды: 0 ≤ d ≤ 31

16-битный код операции: `1001 010d dddd 0000`

Влияние на флаги: S, V = 0, N, Z, C = 1

Описание: Дополнение до единицы содержимого регистра Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
9440 com r4 ; Создаем дополнение до
            ; единицы содержимого r4
f0b1 breq Null ; Переход, если = 0
.org 24
Null:
0000 nop
```

CP

Название: Compare — сравнение

Синтаксис: `cp Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd – Rr

Операнды: 0 ≤ d ≤ 31; 0 ≤ r ≤ 31

16-битный код операции: `0001 01rd dddd rrrr`

Влияние на флаги: H, S, V, N, Z, C

Описание: Сравнение регистров Rd и Rr. При этом Rr вычитается из Rr без сохранения результата (содержимое Rd не изменяется). Флаги устанавливаются как при обычном вычитании, чтобы можно было проверить условие перехода

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
164f cp r4,r31      ;Сравниваем r4 и r31
f4b1 brne Ungleich ;Переход, если не равно
.org 24
Ungleich:
0000 nop
```

CPC

Название: Compare with Carry — сравнение с переносом

Синтаксис: `cpc Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd – Rr – C

Операнды: $0 \le d \le 31$; $0 \le r \le 31$

16-битный код операции: `0000 01rd dddd rrrr`

Влияние на флаги: H, S, V, N, Z, C

Описание: Содержимое регистров Rd и Rr сравнивается с помощью вычитания. При этом учитывается перенос, полученный в результате выполнения предыдущей операции. Содержимое регистров Rr и Rr не изменяется. Флаги можно использовать при условных переходах. Эта команда удобна для сравнения нескольких байтов

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
1440 cp r4,r0 ; Сравнение длинного целого
              ; r7:r4 с r3:r0
0451 cpc r5,r1 ; Учитывается перенос,
0462 cpc r6,r2 ; полученный при предыдущем
0473 cpc r7,r3 ; сравнении
f099 breq Gleich ; Переход, если равно
.org 24
Gleich:
0000 nop
```

CPI

Название: Compare with Immediate — сравнение с непосредственным значением

Синтаксис: `cpi Rd, K8`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd – K8

Операнды: $16 \le d \le 31$; $0 \le K8 \le 255$

16-битный код операции: `0011 KKKK dddd KKKK`

Влияние на флаги: H, S, V, N, Z, C

Описание: Содержимое регистра Rd с помощью вычитания сравнивается с 8-битной константой K8. При этом содержимое регистра не изменяется. Флаги можно использовать при условных переходах

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
3003 cpi r16,3   ; Сравниваем r16 с $03
f0b1 breq Gleich ; Переход, если равно
.org 24
Gleich:
0000 nop
```

CPSE

Название: Compare Skip if Equal — сравнение с пропуском следующей команды, если равно

Синтаксис: `cpse Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если Rd = Rr, то PC ← PC + 2(3), иначе PC ← PC + 1

Операнды: $0 \le d \le 31$; $0 \le r \le 31$

16-битный код операции: `0001 00rd dddd rrrr`

Влияние на флаги: Нет

Описание: Содержимое регистров Rd и Rr сравнивается с помощью вычитания, и в том случае, если Rd=Rr, следующая команда пропускается. Сами регистры и флаги остаются неизменными

Слов (байт): 1 (2)

Тактовых циклов: 1 — если Rd ≠ Rr;
2 — если Rd = Rr и далее следует команда из одного слова;
3 — если Rd = Rr и далее следует команда из двух слов

Пример использования:

```
9443 inc r4      ; Инкрементируем r4
1040 cpse r4,r0  ; Сравниваем r4 с r0
9441 neg r4      ; Будет выполнена только,
                 ; в том случае, если r4≠r0
0000 nop
```

DEC

Название: Декремент

Синтаксис: `dec Rd`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd – 1

Операнды: $0 \le d \le 31$

16-битный код операции: `1001 010d dddd 1010`

Влияние на флаги: S, V, N, Z

Описание: Из содержимого регистра Rd вычитается $01. Флаг переноса не изменяется, благодаря чему Rd можно использовать в качестве счетчика цикла при выполнении арифметических операций над несколькими байтами. При декрементировании беззнаковых операндов можно использовать только сравнения на равенство и неравенство. В случае значений, представленных в дополнительных кодах, условные переходы доступны все

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
e100 ldi r16,$10    ; Загружаем константу
                    ; $10 в регистр r16
Schleife:
0c12 add r1,r2      ; Прибавляем r2 к r1
950a dec r16        ; Декрементируем r16
f7e9 brne Schleife ; Переход, если r16 ≠ 0
0000 nop
```

EOR

Название: Exclusive OR — исключающее "ИЛИ"

Синтаксис: `eor Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd ⊕ Rr

Операнды: $0 \le d \le 31$; $0 \le r \le 31$

16-битный код операции: `0010 01rd dddd rrrr`

Влияние на флаги: S, V=0, N, Z

Описание: Объединение логической операцией "Исключающее ИЛИ" содержимого регистров Rd и Rr

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
2444 eor r4,r4   ; Обнуляем r4
2606 eor r0,r22 ; Объединяем r0 и r22
                ; операцией "Исключающее ИЛИ"
```

ICALL

Название: Indirect Call to a Subroutine — косвенный вызов подпрограммы
Синтаксис: `icall`
Реализована в моделях: AT90S2313, AT90S4414, AT90S8515
Операция: PC ← Z
Операнды: Нет
16-битный код операции: `1001 0101 XXXX 1001`
Влияние на флаги: Нет
Описание: Косвенный вызов подпрограммы, на которую указывает указатель Z (r31:r30). Адрес возврата (команда, следующая после команды `icall`), сохраняется в стеке
Слов (байт): 1 (2)
Тактовых циклов: 3
Пример использования:

```
.org 256
UP1:
000100 9468 bset 6 ; Устанавливаем флаг T
000101 9508 ret
.org 290
000122 e0f1 ldi r31,high(UP1)
000123 e0e0 ldi r30,low(UP1)
000124 9509 icall  ; Вызов UP1
000125 0000 nop    ; Продолжение программы
```

IJMP

Название: Indirect Jump — косвенный переход
Синтаксис: `ijmp`
Реализована в моделях: AT90S2313, AT90S4414, AT90S8515
Операция: PC ← Z
Операнды: Нет
16-битный код операции: `1001 0100 XXXX 1001`
Влияние на флаги: Нет
Описание: Косвенный переход по адресу, на который указывает указатель Z (r31:r30)
Слов (байт): 1 (2)
Тактовых циклов: 2
Пример использования:

```
.org 256
Label1:
000100 0000 nop ; Продолжение программы
.org 290
000118 e0f1 ldi r31,high(Label1)
000119 e0e0 ldi r30,low(Label1)
00011a 9409 icall ; Переход к метке Label1
00011b 0000 nop   ; Продолжение программы
```

IN

Название: Загрузка из порта ввода/вывода в регистр
Синтаксис: `in Rd, P`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Rd ← P
Операнды: $0 \le d \le 31$; $0 \le P \le 63$
16-битный код операции: `1011 0PPd dddd PPPP`
Влияние на флаги: Нет
Описание: Загрузка в регистр Rd содержимого порта ввода/вывода P
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
b3a6 in r26,$16  ; Считываем из порта B в
                 ; регистр r26
30a4 cpi r26,4   ; Проверяем r26 на
                 ; равенство $04
f169 breq Label1 ; Переход к метке Label1,
                 ; если равно
.org 48
Label1:
0000 nop         ; Продолжение программы
```

INC

Название: Инкремент
Синтаксис: `inc Rd`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Rd ← Rd + 1
Операнды: $0 \le d \le 31$
16-битный код операции: `1001 010d dddd 0011`
Влияние на флаги: S, V, N, Z
Описание: К содержимому регистра Rd прибавляется $01. Флаг переноса не изменяется, благодаря чему Rd можно использовать в качестве счетчика цикла при выполнении арифметических операций над несколькими байтами. При инкрементировании беззнаковых операндов можно использовать только сравнения на равенство и неравенство. В случае значений, представленных в дополнительных кодах, условные переходы доступны все
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
27aa clr r26      ; Обнуляем r26
Label1:
95a3 inc r26      ; Инкрементируем r26
                  ; ... Другие команды
34af cpi r26,$4F  ; Сравниваем r26 с $4F
f7e9 brne Label1  ; Переход к метке Label1,
                  ; если равно
0000 nop          ; Продолжение программы
```

JMP

Название: Long Jump — дальний переход

Синтаксис: `jmp k16 (k22)`

Реализована в моделях: Ни в одной модели базовой серии семейства AVR

Операция: PC ← k16 (k22)

Операнды: 0 ≤ k ≤ 64K (0 ≤ k ≤ 4M)

32-битный код операции: `1001 010k kkkk 110k kkkk kkkk kkkk kkkk`

Влияние на флаги: Нет

Описание: Переход по абсолютному адресу k16 (в микроконтроллерах с 16-разрядной шиной адреса) или k22 (в микроконтроллерах с 22-разрядной шиной адреса)

Слов (байт): 2 (4)

Тактовых циклов: 3

Пример использования:

```
.device AT90S8515
000000 2c10 mov r1,r0 ; Копируем r0 в r1
warning : 'JMP' not supported
000001 940c 0800 jmp Far ; Переход по
                         ; абсолютному адресу
.org 2048
Far:
000800 0000 nop    ; Продолжение программы
```

LD (LDD)

Косвенная загрузка из SRAM в регистр с помощью указателя X

Синтаксис: а) `ld Rd, X` (указатель X остается неизменным);
б) `ld Rd, X+` (после передачи указатель X инкрементируется);
в) `ld Rd, -X` (перед передачей указатель X декрементируется)

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: а) Rd ← (X);
б) Rd ← (X) / X ← X + 1;
в) X ← X – 1 / Rd ← (X)

Операнды: 0 ≤ d ≤ 31

16-битный код операции: а) `1001 000d dddd 1100`
б) `1001 000d dddd 1101`
в) `1001 000d dddd 1110`

Влияние на флаги: Нет

Описание: Байт из ячейки SRAM, адресуемой с помощью указателя X (r27:r26), загружается в регистр Rd. В случае (а) после выполнения операции содержимое X остается неизменным, в случае (б) оно увеличивается на 1, а в случае (в) — уменьшается на 1 еще до выполнения передачи. При наличии внешней памяти SRAM доступно все адресное пространство 64K

Слов (байт): 1 (2)
Тактовых циклов: 2
Пример использования:

```
27bb clr r27 ; Очищаем старший байт в X
e2a0 ldi r26,$20 ; Младший байт в X = 32
900d ld r0,X+ ; Загружаем в r0 байт по
              ; адресу $20 и инкрементируем X
901c ld r1,X ; Загружаем в r1 байт по
             ; адресу $21
e2a3 ldi r26,$23 ; Младший байт в X = 35
902c ld r2,X ; Загружаем в r2 байт по
             ; адресу $23
903e ld r3,-X ; Декрементируем X и
       ; загружаем в r3 байт по адресу $22
```

Косвенная загрузка из SRAM в регистр с помощью указателя Y

Синтаксис: а) `ld Rd, Y` (указатель Y остается неизменным);
б) `ld Rd, Y+` (после передачи указатель Y инкрементируется);
в) `ld Rd, -Y` (перед передачей указатель Y декрементируется);
г) `ldd Rd, Y+q` (указатель Y остается неизменным; q — смещение)

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: а) Rd ← (Y);
б) Rd ← (Y) / Y ← Y + 1;
в) Y ← Y - 1 / Rd ← (Y);
г) Rd ← (Y+q)

Операнды: $0 \le d \le 31$; $0 \le q \le 63$

16-битный код операции:
а) `1000 000d dddd 1000`
б) `1001 000d dddd 1001`
в) `1001 000d dddd 1010`
г) `10q0 qq0d dddd 1qqq`

Влияние на флаги: Нет

Описание: Байт из ячейки SRAM, адресуемой с помощью указателя Y (r29:r28), загружается в регистр Rd. В случае (а) после выполнения операции содержимое Y остается неизменным, в случае (б) оно увеличивается на 1, а в случае (в) — уменьшается на 1 еще до выполнения передачи. В случае (г) содерджимое Y не изменяется, но во время выполнения операции складывается с q. При наличии внешней памяти SRAM доступно все адресное пространство 64K

Слов (байт): 1 (2)
Тактовых циклов: 2
Пример использования:

```
27dd clr r29 ; Очищаем старший байт в Y
e2c0 ldi r28,$20 ; Младший байт в Y = 32
9009 ld r0,Y+ ; Загружаем в r0 байт по
```

```
                ; адресу $20 и инкрементируем Y
8018 ld r1,Y ; Загружаем в r1 байт по
                ; адресу $21
e2c3 ldi r28,$23 ; Младший байт в Y = 35
8028 ld r2,Y ; Загружаем в r2 байт по
                ; адресу $23
903a ld r3,-Y ; Декрементируем Y и
       ; загружаем в r3 байт по адресу $22
804a ldd r4,Y+2 ; Загружаем в r4 байт по
                   ; адресу $24
```

Косвенная загрузка из SRAM в регистр с помощью указателя Z

Синтаксис: а) `ld Rd, Z` (указатель Z остается неизменным);
б) `ld Rd, Z+` (после передачи указатель Z инкрементируется);
в) `ld Rd, -Z` (перед передачей указатель Z декрементируется);
г) `ldd Rd, Z+q` (указатель Z остается неизменным; q — смещение)

Реализована в моделях: [AT90S1200 *)], AT90S2313, AT90S4414, AT90S8515

Операция: а) Rd ← (Z);
б) Rd ← (Z) / Z ← Z + 1;
в) Z ← Z – 1 / Rd ← (Z);
г) Rd ← (Z+q)

Операнды: $0 \le d \le 31$; $0 \le q \le 63$

16-битный код операции: а) `1000 000d dddd 0000`
б) `1001 000d dddd 0001`
в) `1001 000d dddd 0010`
г) `10q0 qq0d dddd 0qqq`

Влияние на флаги: Нет

Описание: Байт из ячейки SRAM, адресуемой с помощью указателя Z (r31:r30), загружается в регистр Rd. В случае (а) после выполнения операции содержимое Z остается неизменным, в случае (б) оно увеличивается на 1, а в случае (в) — уменьшается на 1 еще до выполнения передачи. В случае (г) содерджимое Z не изменяется, но во время выполнения операции складывается с q. При наличии внешней памяти SRAM доступно все адресное пространство 64K

*) В случае модели AT90S1200 внутренняя память SRAM не используется, и с помощью указателя Z можно обращаться исключительно к рабочим регистрам из регистрового файла

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:

```
27ff clr r31 ; Очищаем старший байт в Z
e2e0 ldi r30,$20 ; Младший байт в Z = 32
9001 ld r0,Z+ ; Загружаем в r0 байт по
              ; адресу $20 и инкрементируем Z
8010 ld r1,Z ; Загружаем в r1 байт по
             ; адресу $21
e2e3 ldi r30,$23 ; Младший байт в Z = 35
8020 ld r2,Z ; Загружаем в r2 байт по
             ; адресу $23
9032 ld r3,-Z ; Декрементируем Z и
       ; загружаем в r3 байт по адресу $22
8042 ldd r4,Z+2 ; Загружаем в r4 байт по
                ; адресу $24
```

LDI

Название: Load Immediate — загрузка непосредственного значения

Синтаксис: `ldi Rd, K8`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← K8

Операнды: $16 \le d \le 31$; $0 \le K8 \le 255$

16-битный код операции: `1110 KKKK dddd KKKK`

Влияние на флаги: Нет

Описание: Загрузка в регистр Rd 8-битной константы K8

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
27ff clr r31      ; Очищаем старший байт
                  ; указателя Z
e2e0 ldi r30,$20  ; Младший байт Z = 32
95c8 lpm          ; Загружаем байт из флэш-
                  ; памяти в регистр r0
```

LDS

Название: Load Direct from SRAM — прямая загрузка из SRAM

Синтаксис: `lds Rd, k16`

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← (k16)

Операнды: $0 \le d \le 31$; $0 \le k16 \le 65535$

32-битный код операции: `1001 000d dddd 0000 kkkk kkkk kkkk kkkk`

Влияние на флаги: Нет

Описание: Байт, извлеченный по абсолютному адресу k16, загружается непосредственно в регистр Rd. При наличии внешней памяти SRAM доступно все адресное пространство 64K

Слов (байт): 2 (4)

Тактовых циклов: 3

Пример использования:

```
9020 ff00 lds r2,$FF00 ; Загружаем в r2
     ; байт, расположенный по адресу $FF00
0c21 add r2,r1 ; Прибавляем r1 к r2
9220 ff00 sts $FF00,r2 ; Результат обратно
```

LPM

Название: Load Program Memory — загрузка памяти программ

Синтаксис: `lpm`

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: R0 ← (Z)

Операнды: Нет

16-битный код операции: `1001 0101 110X 1000`

Влияние на флаги: Нет

Описание: Байт памяти программ, адресуемый с помощью содержимого указателя Z (r31:r30), загружается в регистр R0. Разряды 15...1 указателя Z адресуют 16-разрядную ячейку программной флэш-памяти. Разряд 0 указателя Z определяет, какой байт должен быть загружен: младший (разряд 0 = 0) или старший (разряд 0 = 1). При наличии внешней памяти SRAM доступно адресное пространство до 32K

Слов (байт): 1 (2)

Тактовых циклов: 3

Пример использования:

```
27ff clr r31      ; Очищаем старший байт
                  ; указателя Z
efe0 ldi r30,$F0 ; Младший байт Z = $F0
95c8 lpm    ; Записываем в r0 младший байт,
            ; считанный по адресу $0078
```

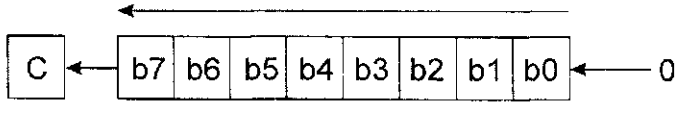
LSL

Название: Logical Shift Left — логический сдвиг влево

Синтаксис: `lsl Rd`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция:



Операнды: $0 \le d \le 31$

16-битный код операции: `0000 11dd dddd dddd ( = add Rd, Rd)`

Влияние на флаги: H, S, V, N, Z, C

Описание: Все разряды регистра Rd сдвигаются на одну позицию влево; флагу переноса присваивается значение разряда 7, а в разряд 0 записывается лог. 0

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования: `0c00 lsl r0 ; Умножение r0 на 2`

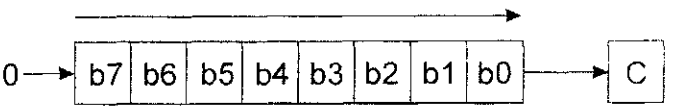
LSR

Название: Logical Shift Right — логический сдвиг вправо

Синтаксис: `lsr Rd`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция:



Операнды: $0 \le d \le 31$

16-битный код операции: `1001 010d dddd 0110`

Влияние на флаги: S, V, N=0, Z, C

Описание: Все разряды регистра Rd сдвигаются на одну позицию вправо; флагу переноса присваивается значение разряда 0, а в разряд 7 записывается лог. 0. Флаг переноса может использоваться для округления

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
9406 lsr r0       ; Делим r0 на 2
f408 brcc Label1 ; Переход, если флаг C=0
9403 inc r0 ; Округляем результат,если C=1
Label1:
0000 nop     ; Продолжение программы
```

MOV

Название: Copy Register — копирование регистра

Синтаксис: `mov Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rr

Операнды: $0 \le d \le 31$; $0 \le r \le 31$

16-битный код операции: `0010 11rd dddd rrrr`

Влияние на флаги: Нет

Описание: Копирование содержимого регистра Rr в регистр Rd. Содержимое регистра Rr остается неизменным

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
2d00 mov r16,r0  ; Копируем содержимое r0
                 ; в r16
2eff mov r15,r31 ; Копируем содержимое r31
                 ; в r15
```

MUL

Название: Multiply — умножение

Синтаксис: `mul Rd, Rr`

Реализована в моделях: Ни в одной модели базовой серии семейства AVR

Операция: R1:R0 ← Rd × Rr

Операнды: $0 \le d \le 31$; $0 \le r \le 31$

16-битный код операции: `1001 11rd dddd rrrr`

Влияние на флаги: C

Описание: Беззнаковое перемножение множителя Rr и множимого Rd. Результат записывается в регистровую пару R1:R0 (R1 — старший байт; R0 — младший байт). Разряд 15 результата копируется в флаг переноса

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:

```
.device AT90S8515
warning : 'MUL' not supported
9c65 mul r6,r5 ; Перемножаем r5 и r6
```

NEG

Название: Two's Complement — дополнение до двух

Синтаксис: `neg Rd`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← $00 – Rd

Операнды: $0 \le d \le 31$

16-битный код операции: `1001 010d dddd 0001`

Влияние на флаги: H, S, V, N, Z, C

Описание: Реализация дополнения до двух содержимого регистра Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
18b0 sub r11,r0 ; Вычитаем r0 из r11
f40a brpl Pos   ; Переход, если результат
                ; положительный
94b1 neg r11  ; Создаем дополнение до двух
              ; содержимого регистра r11
Pos:
0000 nop      ; Продолжение программы
```

NOP

Название: No Operation — нет операции

Синтаксис: `nop`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Нет

Операнды: Нет

16-битный код операции: `0000 0000 0000 0000`

Влияние на флаги: Нет

Описание: Эта команда задерживает на один такт ход программы

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
bb08 out $18,r16 ; Вывод в порт B
0000 nop ; Ожидаем один системный такт
bb18 out $18,r17 ; Еще один вывод в порт B
```

OR

Название: Logical OR — логическая операция "ИЛИ"

Синтаксис: `or Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: $Rd \leftarrow Rd \vee Rr$

Операнды: $0 \le d \le 31$; $0 \le r \le 31$

16-битный код операции: `0010 10rd dddd rrrr`

Влияние на флаги: S, V=0, N, Z

Описание: Объединение логической операцией "ИЛИ" содержимого регистров Rd и Rr. Результат записывается в регистр Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
2823 or r2,r3  ; r2 "ИЛИ" r3
e001 ldi r16,1 ; В регистр r16 загружаем
               ; битовую маску 0000 0001
2a20 or r2,r16 ; Устанавливаем разряд 0 в
               ; регистре r2
```

ORI

Название: Logical OR with Immediate — логическая операция "ИЛИ" с непосредственным значением

Синтаксис: `ori Rd, K8`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd ∨ K8

Операнды: 16 ≤ d ≤ 31; 0 ≤ K8 ≤ 255

16-битный код операции: `0110 KKKK dddd KKKK`

Влияние на флаги: S, V=0, N, Z

Описание: Объединение логической операцией "ИЛИ" содержимого регистра Rd и 8-битной константы K8. Результат записывается в регистр Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
601f ori r17,$0F ; Заполняем лог. 1
                 ; младший полубайт регистра r17
6120 ori r18,$10 ; Устанавливаем в лог. 1
                 ; разряд 4 регистра r18
6a3a ori r19,$AA ; Устанавливаем в лог. 1
       ; разряды 1, 3, 5, 7 в регистре r19
```

OUT

Название: Передача содержимого регистра в порт ввода/вывода

Синтаксис: `out P, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: P ← Rr

Операнды: 0 ≤ r ≤ 31; 0 ≤ P ≤ 63

16-битный код операции: `1011 1PPr rrrr PPPP`

Влияние на флаги: Нет

Описание: Передача в порт P содержимого регистра Rr

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
bb08 out $18,r16 ; Вывод в порт B
0000 nop         ; Ожидаем один системный такт
bb18 out $18,r17 ; Еще один вывод в порт B
```

POP

Название: Извлечение содержимого регистра из стека

Синтаксис: `pop Rd`

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Стек

Операнды: $0 \le d \le 31$

16-битный код операции: `1001 000d dddd 1111`

Влияние на флаги: Нет

Описание: Указатель стека увеличивается на 1, после чего байт в памяти SRAM, на который указывает этот указатель, записывается в регистр Rd

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:

```
000000 d3ff rcall UP1
.org 1024
UP1:
000400 b60f in r0,$3F  ; Копируем в r0
000401 920f push r0    ; Помещаем SREG в стек
000402 930f push r16   ; Помещаем r16 в стек
000403 0000 nop        ; Некоторый код
000404 910f pop r16    ; Восстанавливаем r16
000405 900f pop r0     ; SREG - опять в r0
000406 be0f out $3F,r0 ; Восстанавливаем
                       ; регистр SREG
000407 9508 ret
```

PUSH

Название: Размещение содержимого регистра в стеке

Синтаксис: `push Rd`

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: Стек ← Rd

Операнды: $0 \le r \le 31$

16-битный код операции: `1001 001d dddd 1111`

Влияние на флаги: Нет

Описание: Содержимое регистра Rd сохраняется в ячейке памяти SRAM, адресуемой с помощью указатель стека, после чего этот указатель уменьшается на 1

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:

```
000000 d3ff rcall UP1
.org 1024
UP1:
000400 b60f in r0,$3F  ; Копируем в r0
000401 920f push r0    ; Сохраняем SREG
000402 930f push r16   ; Сохраняем r16
000403 0000 nop        ; Некоторый код
000404 910f pop r16    ; Восстанавливаем r16
000405 900f pop r0     ; SREG - опять в r0
000406 be0f out $3F,r0 ; Восстанавливаем
                       ; регистр SREG
000407 9508 ret
```

RCALL

Название: Relative Call to a Subroutine — относительный вызов подпрограммы

Синтаксис: `rcall k12`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: PC ← PC + k12 + 1

Операнды: -2K ≤ k12 ≤ 2K

16-битный код операции: `1101 kkkk kkkk kkkk`

Влияние на флаги: Нет

Описание: Вызов подпрограммы, начало которой должно находиться в диапазоне PC ± 2K слов. Адрес возврата (команда, следующая после команды `rcall`), помещается в стек. Благодаря круговой адресации, команда `rcall` может использоваться без ограничений всеми представителями базовой серии семейства AVR

Слов (байт): 1 (2)

Тактовых циклов: 3

Пример использования:

```
.org $FF0
UP1:
000ff0 9468 bset 6 ; Установка флага T
000ff1 9508 ret
.org $01A
00001a dfd5 rcall UP1 ; Вызов UP1
00001b 0000 nop ; Продолжение программы
```

RET

Название: Return from Subroutine — возврат из подпрограммы

Синтаксис: `ret`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: PC ← Стек

Операнды: Нет

16-битный код операции: `1001 0101 0XX0 1000`

Влияние на флаги: Нет

Описание: Выход из подпрограммы. Указатель стека увеличивается на 1, после чего соответствующий байт из SRAM копируется в старший байт счетчика команд. Затем указатель стека еще раз увеличивается на 1 и соответствующий байт из SRAM копируется в младший байт счетчика команд

Слов (байт): 1 (2)

Тактовых циклов: 4

Пример использования:

```
000000 d3ff rcall UP1
000001 0000 nop ; Продолжение программы
.org 1024
UP1:
000400 930f push r16 ; Помещаем в стек r16
000401 0000 nop ; Выполнение подпрограммы
000402 910f pop r16 ; Восстанавливаем r16
000403 9508 ret
```

RETI

Название: Return from Interrupt — возврат из подпрограммы обработки прерывания

Синтаксис: `reti`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: PC ← Стек

Операнды: Нет

16-битный код операции: `1001 0101 0XX1 1000`

Влияние на флаги: I=1

Описание: Возврат из подпрограммы обработки прерывания. Указатель стека увеличивается на 1, после чего байт из памяти SRAM, на который указывает этот указатель, копируется в старший байт счетчика команд. Затем указатель стека еще раз увеличивается на 1 и соответствующий байт из SRAM копируется в младший байт счетчика команд. Устанавливается флаг общего разрешения прерываний

Слов (байт): 1 (2)

Тактовых циклов: 4

Пример использования:

```
ExtInt:
930f push r16 ; Помещаем в стек r16
0000 nop      ; Некоторая подпрограмма
              ; обработки прерывания
910f pop r16  ; Восстанавливаем r16
9518 reti     ; Возврат из подпрограммы и
          ; установка разрешения прерываний
```

RJMP

Название: Relative Jump — относительный переход

Синтаксис: `rjmp k12`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: PC ← PC + k12 + 1

Операнды: -2K ≤ k12 ≤ 2K

16-битный код операции: `1100 kkkk kkkk kkkk`

Влияние на флаги: Нет

Описание: Относительный переход по адресу, который должен находиться в диапазоне PC ± 2K слов. Адрес возврата (команда, следующая после команды `rjmp`), помещается в стек. Благодаря круговой адресации, команда `rjmp` может использоваться без ограничений всеми представителями базовой серии семейства AVR

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:

```
.org $01A
Label1:
00001a 0000 nop ; Продолжение программы
.org $FF0
000ff0 c029 rjmp Label1 ; Переход к Label1
```

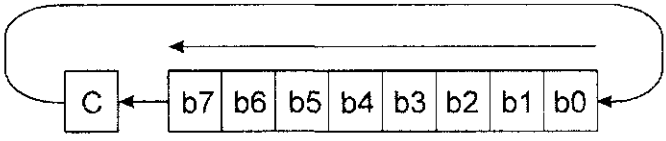
ROL

Название: Rotate Left trough Carry — циклический сдвиг влево через флаг переноса

Синтаксис: `rol Rd`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция:



Операнды: $0 \le d \le 31$

16-битный код операции: `0001 11dd dddd dddd ( = adc Rd, Rd)`

Влияние на флаги: H, S, V, N, Z, C

Описание: Все разряды регистра Rd сдвигаются на одну позицию влево; флаг переноса копируется в разряд 0, а разряд 7 — во флаг переноса

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
1fdd rol r29 ; Циклический сдвиг влево
             ; через флаг переноса регистра r29
```

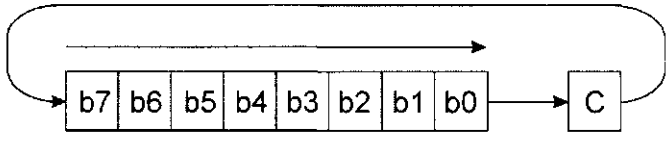
ROR

Название: Rotate Right trough Carry — циклический сдвиг вправо через флаг переноса

Синтаксис: `ror Rd`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция:



Операнды: $0 \le d \le 31$

16-битный код операции: `1001 010d dddd 0111`

Влияние на флаги: S, V, N, Z, C

Описание: Все разряды регистра Rd сдвигаются на одну позицию вправо; флаг переноса копируется в разряд 7, а разряд 0 — во флаг переноса

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
9437 ror r3 ; Циклический сдвиг вправо
            ; через флаг переноса регистра r3
```

SBC

Название: Subtract with Carry — вычитание с учетом переноса

Синтаксис: `sbc Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd – Rr – C

Операнды: $0 \le d \le 31; 0 \le r \le 31$

16-битный код операции: `0000 10rd dddd rrrr`

Влияние на флаги: H, S, V, N, Z, C

Описание: Содержимое регистра Rr и флаг переноса вычитаются из содержимого регистра Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
;Вычитание байтов r1:r0 из r3:r2
1820 sub r2,r0 ; Вычитаем младший байт
0831 sbc r3,r1 ; Вычитаем старший байт
               ; с учетом переноса
```

SBCI

Название: Subtract Immediate with Carry — вычитание непосредственного значения с учетом переноса

Синтаксис: `sbci Rd, K8`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd – K8 – C

Операнды: $16 \le d \le 31; 0 \le K \le 255$

16-битный код операции: `0100 KKKK dddd KKKK`

Влияние на флаги: H, S, V, N, Z, C

Описание: 8-битная константа K8 и флаг переноса вычитаются из содержимого регистра Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
; Вычитание константы $4F23 из r17:r16
5203 subi r16,low($4F23)  ; Вычитаем
                          ; младший байт
441f sbci r17,high($4F23) ; Вычитаем
                    ; старший байт и флаг C
```

SBI

Название: Set Bit in I/O-Register — установка разряда в регистре ввода/вывода

Синтаксис: `sbi P, b`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: I/O(P, b) ← 1

Операнды: $0 \le P \le 31$; $0 \le b \le 7$

16-битный код операции: `1001 1010 pppp pbbb`

Влияние на флаги: Нет

Описание: Запись лог. 1 в разряд b регистра ввода/вывода P. Эта команда может быть использована только для регистров ввода/вывода 0–31 ($00–$1F)

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:
```
9ab8 sbi $17,0 ; Установка разряда
               ; направления передачи данных DDB0
```

SBIC

Название: Skip if Bit in I/O-Register is Cleared — пропуск следующей команды, если очищен разряд в регистре ввода/вывода

Синтаксис: `sbic P, b`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если I/O(b) = 0, то PC ← PC + 2 (3), иначе PC ← PC + 1

Операнды: $0 \le P \le 31$; $0 \le b \le 7$

16-битный код операции: `1001 1001 pppp pbbb`

Влияние на флаги: Нет

Описание: Если разряд b регистра ввода/вывода P содержит лог. 0, то следующая команда пропускается

Слов (байт): 1 (2)

Тактовых циклов: 1 — если разряд b содержит лог. 1;
2 — если разряд b содержит лог. 0, и далее следует команда длиной в одно слово;
3 — если разряд b содержит лог. 0, и далее следует команда длиной в два слова (например, команда `lds`)

Пример использования:
```
EEWait:
99e1 sbic $1C,1 ; Если EEWE=0, то
                ; следующая команда пропускается
cffe rjmp EEWait ; Активен процесс
                 ; записи в память EEPROM
0000 nop ; Продолжение программы
```

SBIS

Название: Skip if Bit in I/O-Register is Set — пропуск следующей команды, если установлен разряд в регистре ввода/вывода
Синтаксис: `sbis P, b`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Если I/O(b) = 1, то PC ← PC + 2 (3), иначе PC ← PC + 1
Операнды: $0 \le P \le 31$; $0 \le b \le 7$
16-битный код операции: `1001 1011 pppp pbbb`
Влияние на флаги: Нет
Описание: Если разряд b регистра ввода/вывода P содержит лог. 1, то следующая команда пропускается
Слов (байт): 1 (2)
Тактовых циклов: 1 — если разряд b содержит лог. 0;
2 — если разряд b содержит лог. 1, и далее следует команда длиной в одно слово;
3 — если разряд b содержит лог. 1, и далее следует команда длиной в два слова (например, команда `sts`)
Пример использования:

```
WaitSet:
9b87 sbis $10,7 ; Если установлен разряд
                ; 7 регистра порта D, то
        ; следующая команда пропускается
cffe rjmp EEWait ; Ожидаем до тех пор,
       ; пока не будет установлен разряд 7
0000 nop ; Продолжение программы
```

SBIW

Название: Subtract Immediate from Word — вычитание непосредственного значения из слова
Синтаксис: `sbiw Rdl, K6`
Реализована в моделях: AT90S2313, AT90S4414, AT90S8515
Операция: Rdh:Rdl ← Rdh:Rdl – K6
Операнды: dl ∈ {24, 26, 28, 30}; $0 \le K6 \le 63$
16-битный код операции: `1001 0111 KKdd KKKK`
Влияние на флаги: S, V, N, Z, C
Описание: 6-битная гонстанта K6 вычитается из содержимого регистровой пары Rdh:Rdl. Эта команда применяется к четырем старшим регистровым парам из регистрового файла, и обычно используется при работе с указателями
Слов (байт): 1 (2)
Тактовых циклов: 2
Пример использования:

```
9701 sbiw r24,1 ; Уменьшаем содержимое
                ; r25:r24 на 1
97ff sbiw r30,63 ; Вычитаем 63 из
                 ; указателя Z (r31:r30)
```

SBR

Название: Set Bit(s) in Register — установка разряда(ов) в регистре
Синтаксис: `sbr Rd, K8`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Rd ← Rd ∨ K8
Операнды: 16 ≤ d ≤ 31; 0 ≤ K8 ≤ 255
16-битный код операции: `0110 KKKK dddd KKKK`
Влияние на флаги: S, V = 0, N, Z
Описание: Установка в регистре Rd разрядов, заданных с помощью операции "ИЛИ" с константой K8. Эта команда применима только к регистрам r16–r31
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
6f20 sbr r18,$F0 ; Запись лог. 1 во все
                 ; разряды старшего полубайта
6003 sbr r16,3 ; Запись лог. 1 в разряды 0
               ; и 1 регистра r16
```

SBRC

Название: Skip if Bit in Register is Cleared — пропуск следующей команды, если очищен разряд в регистре
Синтаксис: `sbrc Rr, b`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Если Rr(b) = 0, то PC ← PC + 2 (3), иначе PC ← PC + 1
Операнды: 0 ≤ r ≤ 31; 0 ≤ b ≤ 7
16-битный код операции: `1111 110r rrrr Xbbb`
Влияние на флаги: Нет
Описание: Если разряд b регистра Rr содержит лог. 0, то следующая команда пропускается
Слов (байт): 1 (2)
Тактовых циклов: 1 — если разряд b содержит лог. 1;
2 — если разряд b содержит лог. 0, и далее следует команда длиной в одно слово;
3 — если разряд b содержит лог. 0, и далее следует команда длиной в два слова (например, команда `lds`)
Пример использования:

```
1801 sub r0,r1 ; Вычитаем r1 из r0
fc07 sbrc r0,7 ; Если разряд 7 регистра
      ; r0 содержит лог. 0, то пропускаем,
      ; следующую команду
1801 sub r0, r1 ; Выполняется только в том
  ; случае, если разряд 7 содержит лог. 1
0000 nop ; Продолжение программы
```

SBRS

Название: Skip if Bit in Register is Set — пропуск следующей команды, если установлен разряд в регистре

Синтаксис: `sbrs Rr, b`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Если Rr(b) = 1, то PC ← PC + 2 (3), иначе PC ← PC + 1

Операнды: $0 \le r \le 31$; $0 \le b \le 7$

16-битный код операции: `1111 111r rrrr Xbbb`

Влияние на флаги: Нет

Описание: Если разряд b регистра Rr содержит лог. 1, то следующая команда пропускается

Слов (байт): 1 (2)

Тактовых циклов: 1 — если разряд b содержит лог. 0;
2 — если разряд b содержит лог. 1, и далее следует команда длиной в одно слово;
3 — если разряд b содержит лог. 1, и далее следует команда длиной в два слова (например, команда `sts`)

Пример использования:

```
1801 sub r0,r1 ; Вычитаем r1 из r0
fe07 sbrs r0,7 ; Если разряд 7 регистра
      ; r0 содержит лог. 1, то пропускаем,
      ; следующую команду
1801 sub r0, r1 ; Выполняется только в том
  ; случае, если разряд 7 содержит лог. 0
0000 nop ; Продолжение программы
```

SEC

Название: Set Carry Flag — установка флага переноса

Синтаксис: `sec`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: C ← 1

Операнды: Нет

16-битный код операции: `1001 0100 0000 1000`

Влияние на флаги: C = 1

Описание: Установка флага переноса в регистре SREG

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
9408 sec ; Устанавливаем флаг C
1c01 adc r0,r1 ; r0 = r0 + r1 + C
```

SEH

Название: Set Half Carry Flag — установка флага половинного переноса
Синтаксис: `seh`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: H ← 1
Операнды: Нет
16-битный код операции: `1001 0100 0101 1000`
Влияние на флаги: H = 1
Описание: Установка флага половинного переноса в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
9458 seh ; Устанавливаем флаг H
```

SEI

Название: Set Global Interrupt Flag — установка флага общего разрешения прерываний
Синтаксис: `sei`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: I ← 1
Операнды: Нет
16-битный код операции: `1001 0100 0111 1000`
Влияние на флаги: I = 1
Описание: Установка флага общего разрешения прерываний в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
9478 sei ; Устанавливаем флаг I
```

SEN

Название: Set Negative Flag — установка флага отрицательного результата
Синтаксис: `sen`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: N ← 1
Операнды: Нет
16-битный код операции: `1001 0100 0010 1000`
Влияние на флаги: N = 1
Описание: Установка флага отрицательного результата в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
9428 sen ; Устанавливаем флаг N
```

SER

Название: Set all Bits in Register — установка всех разрядов в регистре
Синтаксис: `ser Rd`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Rd ← $FF
Операнды: $16 \le d \le 31$
16-битный код операции: `1110 1111 dddd 1111`
Влияние на флаги: Нет
Описание: Установка всех разрядов регистра посредством записи непосредственного значения $FF
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
ef0f ser r16
efff ser r31
```

SES

Название: Set Signed Flag — установка флага знака
Синтаксис: `ses`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: S ← 1
Операнды: Нет
16-битный код операции: `1001 0100 0100 1000`
Влияние на флаги: S = 1
Описание: Установка флага знака в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
0c23 add r2,r3 ; Прибавляем r3 к r2
9448 ses       ; Устанавливаем флаг S
```

SET

Название: Set Transfer Flag — установка флага копирования
Синтаксис: `set`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: T ← 1
Операнды: Нет
16-битный код операции: `1001 0100 0110 1000`
Влияние на флаги: T = 1
Описание: Установка флага копирования в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
9468 set ; Устанавливаем флаг T
```

SEV

Название: Set Overflow Flag — установка флага переполнения
Синтаксис: `sev`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: V ← 1
Операнды: Нет
16-битный код операции: `1001 0100 0011 1000`
Влияние на флаги: V = 1
Описание: Установка флага переполнения в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
9438 sev ; Устанавливаем флаг V
```

SEZ

Название: Set Zero Flag — установка нулевого флага
Синтаксис: `sez`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Z ← 1
Операнды: Нет
16-битный код операции: `1001 0100 0001 1000`
Влияние на флаги: Z = 1
Описание: Установка нулевого флага в регистре SREG
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:
```
9418 sez ; Устанавливаем флаг Z
```

SLEEP

Синтаксис: `sleep`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Переводит процессор в "спящий" режим
Операнды: Нет
16-битный код операции: `1001 0101 100X 1000`
Влияние на флаги: Нет
Описание: Процессор переходит в "спящий" режим, определенный с помощью регистра MCUCR
Слов (байт): 1 (2)
Тактовых циклов: 1 (в некоторых случаях компания Atmel определяет для команды `sleep` три тактовых цикла)
Пример использования:
```
b705 in r16,$35 ; Извлекаем содержимое
                ; регистра MCUCR
6300 sbr r16,0b00110000 ; Устанавливаем
                        ; разряды 4(SM) и 5(SE)
bf05 out $35,r16 ; Записываем в MCUCR
                 ; обновленное содержимое
9588 sleep ; Переводим ЦП в "спящий" режим
0000 nop ; Продолжение программы
```

ST (STD)

Косвенное сохранение содержимого регистра в SRAM с помощью указателя X

Синтаксис: а) `st X, Rr` (указатель X остается неизменным);
б) `st X+, Rr` (после передачи указатель X инкрементируется);
в) `st -X, Rr` (перед передачей указатель X декрементируется)

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: а) (X) ← Rr;
б) (X) ← Rr / X ← X + 1;
в) X ← X – 1 / (X) ← Rd

Операнды: $0 \le r \le 31$

16-битный код операции:
```
а) 1001 001r rrrr 1100
б) 1001 001r rrrr 1101
в) 1001 001r rrrr 1110
```

Влияние на флаги: Нет

Описание: Содержимое регистра Rr сохраняется в ячейку SRAM, адресуемую с помощью указателя X (r27:r26). В случае (а) после выполнения операции содержимое X остается неизменным, в случае (б) оно увеличивается на 1, а в случае (в) — уменьшается на 1 еще до выполнения передачи. При наличии внешней памяти SRAM доступно все адресное пространство 64K

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:
```
27bb clr r27 ; Очищаем старший байт в X
e2a0 ldi r26,$20 ; Младший байт в X = 32
920d st X+,r0 ; Сохраняем r0 в ячейке
              ; адресу $20 и инкрементируем X
921c st X,r1 ; Сохраняем r1 в ячейке по
             ; адресу $21
e2a3 ldi r26,$23 ; Младший байт в X = 35
922c st X,r2 ; Сохраняем r2 в ячейке по
             ; адресу $23
923e st -X,r3 ; Декрементируем X и
     ; сохраняем r3 в ячейке по адресу $22
```

Косвенное сохранение содержимого регистра в SRAM с помощью указателя Y

Синтаксис: а) `st Y, Rr` (указатель Y остается неизменным);
б) `st Y+, Rr` (после передачи указатель Y инкрементируется);
в) `st -Y, Rr` (перед передачей указатель Y декрементируется);
г) `std Y+q, Rr` (указатель Y остается неизменным; q — смещение);

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: а) (Y) ← Rr;
б) (Y) ← Rr / Y ← Y + 1;
в) Y ← Y – 1 / (Y) ← Rr;
г) (Y+q) ← Rr

Операнды: $0 \le r \le 31$; $0 \le q \le 63$

16-битный код операции:
а) `1000 001r rrrr 1000`
б) `1001 001r rrrr 1001`
в) `1001 001r rrrr 1010`
г) `10q0 qq1r rrrr 1qqq`

Влияние на флаги: Нет

Описание: Содержимое регистра Rr сохраняется в ячейку SRAM, адресуемую с помощью указателя Y (r29:r28). В случае (а) после выполнения операции содержимое Y остается неизменным, в случае (б) оно увеличивается на 1, а в случае (в) — уменьшается на 1 еще до выполнения передачи. В случае (г) содерджимое Y не изменяется, но во время выполнения операции складывается с q. При наличии внешней памяти SRAM доступно все адресное пространство 64K

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:

```
27dd clr r29 ; Очищаем старший байт в Y
e2c0 ldi r28,$20 ; Младший байт в Y = 32
9209 st Y+,r0 ; Сохраняем r0 в ячейке по
             ; адресу $20 и инкрементируем Y
8218 st Y,r1 ; Сохраняем r1 в ячейке по
             ; адресу $21
e2c3 ldi r28,$23 ; Младший байт в Y = 35
8228 st Y,r2 ; Сохраняем r2 в ячейке по
             ; адресу $23
923a st -Y,r3 ; Декрементируем Y и
     ; сохраняем r3 в ячейке по адресу $22
```

Косвенное сохранение содержимого регистра в SRAM с помощью указателя Z

Синтаксис: а) `st Z, Rr` (указатель Z остается неизменным);
б) `st Z+, Rr` (после передачи указатель Z инкрементируется);
в) `st -Z, Rr` (перед передачей указатель Z декрементируется);
г) `std Z+q, Rr` (указатель Z остается неизменным; q — смещение);

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: а) (Z) ← Rr;
б) (Z) ← Rr / Z ← Z + 1;
в) Z ← Z – 1 / (Z) ← Rr;
г) (Z+q) ← Rr

Операнды: $0 \le r \le 31$; $0 \le q \le 63$

16-битный код операции:
а) `1000 001r rrrr 0000`
б) `1001 001r rrrr 0001`
в) `1001 001r rrrr 0010`
г) `10q0 qq1r rrrr 0qqq`

Влияние на флаги: Нет

Описание: Содержимое регистра Rr сохраняется в ячейку SRAM, адресуемую с помощью указателя Z (r31:r30). В случае (а) после выполнения операции содержимое Z остается неизменным, в случае (б) оно увеличивается на 1, а в случае (в) — уменьшается на 1 еще до выполнения передачи. В случае (г) содерджимое Z не изменяется, но во время выполнения операции складывается с q. При наличии внешней памяти SRAM доступно все адресное пространство 64K

Слов (байт): 1 (2)

Тактовых циклов: 2

Пример использования:

```
27ff clr r31 ; Очищаем старший байт в Z
e2e0 ldi r30,$20 ; Младший байт в Z = 32
9201 st Z+, r0 ; Сохраняем r0 в ячейке по
          ; адресу $20 и инкрементируем Z
8210 st Z, r1 ; Сохраняем r1 в ячейке по
            ; адресу $21
e2e3 ldi r30,$23 ; Младший байт в Z = 35
8220 st Z, r2 ; Сохраняем r2 в ячейке по
            ; адресу $23
9232 st -Z, r3 ; Декрементируем Z и
     ; сохраняем r3 в ячейке по адресу $22
```

STS

Название: Store Direct to SRAM — прямая запись в SRAM

Синтаксис: `sts k16, Rr`

Реализована в моделях: AT90S2313, AT90S4414, AT90S8515

Операция: (k16) ← Rr

Операнды: $0 \le r \le 31$; $0 \le k16 \le 65535$

32-битный код операции: `1001 001d dddd 0000 kkkk kkkk kkkk kkkk`

Влияние на флаги: Нет

Описание: Загрузка в ячейку SRAM по абсолютному 16-битному адресу k16 содержимого регистра Rr. При наличии внешней памяти SRAM доступно все адресное пространство 64K

Слов (байт): 2 (4)

Тактовых циклов: 3

Пример использования:

```
9020 ff00 lds r2,$FF00 ; Загрузка d r2
                       ; байта по адресу $FF00
0c21 add r2,r1 ; Прибавляем r1 к r2
9220 ff00 sts $FF00,r2 ; Результат -
                     ; обратно по адресу $FF00
```

SUB

Название: Subtract without Carry — вычитание без учета переноса

Синтаксис: `sub Rd, Rr`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd – Rr

Операнды: $0 \le d \le 31$; $0 \le r \le 31$

16-битный код операции: `0001 10rd dddd rrrr`

Влияние на флаги: H, S, V, N, Z, C

Описание: Содержимое регистра Rr вычитается из содержимого регистра Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
;Вычитание r1:r0 из r3:r2
1820 sub r2,r0 ; Вычитаем младший байт
0831 sub r3,r1 ; Вычитаем старший байт и
               ; флаг переноса
```

SUBI

Название: Subtract Immediate without Carry — вычитание непосредственного значения без учета переноса

Синтаксис: `subi Rd, K8`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd ← Rd – K8

Операнды: 16 ≤ d ≤ 31; 0 ≤ K8 ≤ 255

16-битный код операции: `0101 KKKK dddd KKKK`

Влияние на флаги: H, S, V, N, Z, C

Описание: 8-битная константа K8 вычитается из содержимого регистра Rd

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
;Вычитание константы $4F23 из r17:r16
5203 subi r16,low($4F23)  ; Вычитаем
                          ; младший байт
441f sbci r17,high($4F23) ; Вычитаем
                          ; старший байт
```

SWAP

Название: Перестановка полубайтов

Синтаксис: `swap Rd`

Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515

Операция: Rd(7–4) ← Rd(3–0), Rd(3–0) ← Rd(7–4)

Операнды: 0 ≤ d ≤ 31

16-битный код операции: `1001 010d dddd 0010`

Влияние на флаги: Нет

Описание: Старший и младший полубайты регистра Rd меняются местами

Слов (байт): 1 (2)

Тактовых циклов: 1

Пример использования:

```
9412 swap r1 ; Меняем местами старший и
             ; младший полубайты регистра r1
9413 inc r1 ; Увеличиваем на 1 старший
            ; полубайт
9412 swap r1 ; Опять меняем местами
             ; полубайты регистра r1
```

TST

Название: Test for Zero or Minus — проверка на нуль или минус
Синтаксис: `tst Rd`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Rd ← Rd ∧ Rd
Операнды: 0 ≤ d ≤ 31
16-битный код операции: `0010 00dd dddd dddd`
Влияние на флаги: S, V = 0, N, Z
Описание: Проверка, является ли содержимое регистра Rd нулевым или отрицательным, с помощью применения логической операции "И". Само содержимое регистра Rd остается неизменным
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
2000 tst r0     ; Проверяем регистр r0
f169 breq Null ; Переход, если = 0
.org 47
Null:
0000 nop ; Продолжение программы
```

WDR

Название: Watchdog Reset — сброс сторожевого таймера
Синтаксис: `wdr`
Реализована в моделях: AT90S1200, AT90S2313, AT90S4414, AT90S8515
Операция: Перевод сторожевого таймера в исходное состояние
Операнды: Нет
16-битный код операции: `1001 0101 101X 1000`
Влияние на флаги: Нет
Описание: Эта команда устанавливает в исходное состояние сторожевой таймер (см. главу 5)
Слов (байт): 1 (2)
Тактовых циклов: 1
Пример использования:

```
95a8 wdr ; Сброс сторожевого таймера
```

13 Ассемблер

AVR-ассемблер от компании Atmel транслирует исходный код пользовательской программы в объектный и исполняемый коды. Последний можно протестировать в оболочке AVR-Studio и после этого загрузить (например, с помощью инструментального набора AVR STK200) во флэш-память или в память EEPROM микроконтроллера AVR.

В случае с AVR-ассемблером речь идет о макроассемблере, где часто повторяющиеся части программы, которые не должны выполняться в виде подпрограмм, могут определяться как макросы. Подробнее об этом речь пойдет ниже при рассмотрении директив `.MACRO` и `.ENDMACRO`.

И AVR-ассемблер, и AVR-Studio включены в состав прилагаемого к книге компакт-диска. Сюда входит как Windows-версия ассемблера `WAVRASM.EXE`, поддерживающая операционные системы Windows 3.11, Windows 95 и Windows NT; так и MS-DOS-версия для работы в режиме командной строки `AVRASM.EXE`.

Микроконтроллеры AVR, как и любые другие микропроцессоры, используют в работе, главным образом, собственный машинный язык. Поскольку эти последовательности нулей и единиц (см., к примеру, параметр "16-битный код операции" из описания команд в предыдущей главе) сложны для восприятия, существует язык ассемблера, который, благодаря своей мнемонической форме, гораздо проще в применении. При этом каждой инструкции ассемблера соответствует отдельная машинная команда.

Пользователь создает свою программу (исходный код — файл, который, по умолчанию, имеет расширение `*.asm`) с помощью интегрированного или внешнего редактора, а всю работу по трансляции на машинный язык (объектный код — файл с расширением `*.obj`) ассемблер берет на себя. Во время трансляции также создается листинг программы (файл с расширением `*.lst`), по которому пользователь может определить адреса и машинный код, назначенные ассемблером своим командам.

В загрузочном файле с расширением `*.hex` находятся машинные команды для записи в память программ в виде ASCII-кода в одном из следующих форматов: Generic (Atmel), S-record (Motorola) или Intellec 8/MDS (Intel). Необязательный EEPROM-файл с расширением `*.eep` содержит данные (при их наличии), которые должны записываться в память EEPROM. Желаемый формат выбирается в меню ассемблера **Options** (Параметры). Большинство программаторов могут обрабатывать эти форматы напрямую.

Ниже представлены примеры файлов, созданных AVR-ассемблером.

Исходный код: Beispiel.asm

```
cseg            ; Сегмент кода
.org 512        ; Начальный адрес программы
clr r31         ; Обнулить старший байт указателя Z
ldi r30,$20     ; Младший байт указателя Z = 32
ld r0,Z+        ; Копировать байт по адресу $20 в r0, увеличить Z на 1
ld r1,Z         ; Копировать байт по адресу $21 в r1

.org 1024       ; Начальный адрес для констант
Konst: .db "ABC", $0A, $0D, 0, -126
.eseg           ; Сегмент EEPROM
.org 24         ; Начальный адрес для констант в EEPROM
eeKonst: .db $AA, $55, 0, 255, " "
```

Файл листинга: Beispiel.lst:

```
AVRASM ver. 1.21 BEISPIEL.ASM Mon Jun 28 14:31:15 1999

                cseg                    ; Сегмент кода
                org 512                 ; Начальный адрес программы
000200 27ff             clr r31         ; Обнуляем старший байт указателя Z
000201 e2e0             ldi r30,$20     ; Младший байт указателя Z = 32
000202 9001             ld r0,Z+        ; Загружаем байт по адресу $20 в r0, Z=Z+1
000203 8010             ld r1,Z         ; Загружаем байт по адресу $21 в r1

                .org 1024               ; Начальный адрес для констант
000400 Konst: .db "ABC", $0A, $0D, 0, -126
000400 4241
000401 0a43
000402 000d
000403 0082

                .eseg                   ; Сегмент EEPROM
                .org 24                 ; Начальный адрес для констант в EEPROM
000018 eeKonst: .db $AA, $55, 0, 255, " "
000018 aa
000019 55
00001a 00
00001b ff
00001c 20

Assembly complete with no errors.
```

Рассмотрим загрузочный файл в формате Generic.

Файл Beispiel.hex

```
000200:27ff
000201:e2e0
000202:9001
000203:8010
000400:4241
000401:0a43
000402:000d
000403:0082
```

EEPROM-файл `Beispiel.eep`

```
0018:aa
0019:55
001a:00
001b:ff
001c:20
```

Синтаксис формата Atmel Generic:

- шестнадцатиричные данные представлены в виде ASCII-символов и, таким образом, для представления одного байта требуется два ASCII-символа;
- каждая строка для записи в память программ состоит из трех байтов: адреса, оканчивающегося символом ":", и следующего за ним слова данных, предназначенного для записи по указанному адресу;
- строка для записи в память EEPROM состоит только из двух байтов: адреса, оканчивающегося символом ":", и следующего за ним байта данных, предназначенного для записи по указанному адресу.

Рассмотрим загрузочный файл в формате Intel Intellec 8/MDS.

Файл `Beispiel.hex`

```
:020000020000FC
:08040000FF27E0E201901080EB
:0808000041424 30A0D00820091
:00000001FF
```

EEPROM-файл `Beispiel.eep`

```
:05001800AA5500FF20C5
:00000001FF
```

Синтаксис формата Intel Intellec 8/MDS:

- шестнадцатеричные данные представлены в виде ASCII-символов и, таким образом, для представления одного байта требуется два ASCII-символа;
- каждая строка начинается с ":" и следующего за ним байта, определяющего количество байтов для загрузки; затем следуют два байта: адрес, по которому должен быть загружен первый байт данных, и байт, задающий тип строки.

Существует три типа строк в формате Intel Intellec 8/MDS:

- 00 — данные — после указателя типа следует столько байтов данных, сколько было указано в первом байте строки, а также контрольная сумма;
- 01 — оконечная запись — указывает на окончание файла данных (после указателя типа следует только контрольная сумма);
- 02 — адрес*а* сегментов для расширения адресного пространства — после указателя типа следует смещение адреса, которое вместе с двумя байтами из строки дает абсолютный адрес. Для вычисления абсолютного адреса смещение адреса умножается на 16 (сдвиг влево на четыре разряда) и после этого прибавляется к адресу из строки.

Пример: смещение адреса = 1230_h; адрес в наборе данных = 0045_h.
Абсолютный адрес = $12300_h + 0045_h = 12345_h$.

Строка типа "*адреса сегментов*" может встречаться в любом месте файла и задавать смещение для последующих строк. После смещения адреса следует завершающая контрольная сумма.

Все вышесказанное иллюстрирует рис. 13.1. Каждая строка завершается контрольной суммой. Эта контрольная сумма представляет собой дополнение до двух суммы всех байтов строки за исключением вводного двоеточия, то есть, — количества байтов данных, *а*дреса, типа и всех байтов данных (если они присутствуют).

Рис. 13.1. Структура строк в формате Intel Intellec 8/MDS

После контрольной суммы могут следовать непечатаемые ASCII-символы возврата каретки (0Dh) или конца строки (0Ah), а также нулевой символ (00h).

Теперь рассмотрим загрузочный файл в формате Motorola S-record.

Файл `Beispiel.hex`

```
S00F000042454953504945C2E48
455890
S10B0400FF27E0E201901080E7
S10B08004142430A0D0082008D
S9030000FC
```

EEPROM-файл `Beispiel.eep`

```
S00F000042454953504945C2E45
45509B
S1080018AA5500FF20C1
S9030000FC
```

Синтаксис формата Motorola S-record:

- шестнадцатиричные данные представлены в виде ASCII-символов и, таким образом, для представления одного байта требуется два ASCII-символа;
- файл данных в формате S-record может (но не обязазательно) начинаться с "вводной" строки, содержащей стартовый код "S0" или "S5", за которым следует байт, информирующий о количестве следующих далее байтов,

включая контрольную сумму (AVR-ассемблер вставляет также в этот набор имя файла `*.hex`);

- каждая строка начинается с одного из стартовых кодов: "S1", "S2" или "S3" (код "S1" задает два, "S2" — три, а "S3" — четыре байта адреса);
- после стартового кода следует байт, который определяет количество последующих байтов в этой строке — число, соответствующее общей длине адреса, данных и контрольной суммы;
- далее следует адрес первого байта данных (длина адреса задается стартовым кодом);
- строка завершается контрольной суммой, которая представляет собой дополнение до единицы количества байтов, *а*дреса и всех байтов данных (если они присутствуют);
- окончание строки "S3" обозначается с помощью записи "S7", формат которой идентичен записи "Конец файла" (см. ниже) за исключением того, что поле адреса может содержать начальный адрес выполнения программы после загрузки файла (этот начальный адрес программатором, как правило, игнорируется).
- запись "Конец файла" указывает на окончание файла данных. Она состоит из кода "S9" или "S8", байта количества байтов , *а*дреса и контрольной суммы.

После контрольной суммы могут следовать непечатаемые ASCII-символы возврата каретки (0Dh) или конца строки (0Ah), а также нулевой символ (00h).

Все вышесказанное иллюстрирует рис. 13.2.

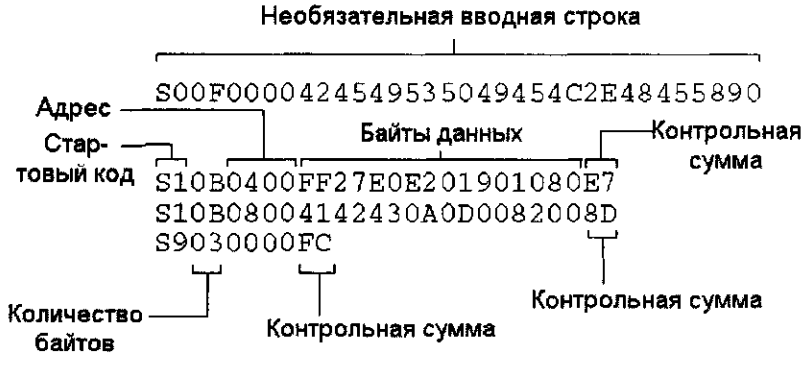


Рис. 13.2. Структура строк в формате Motorola S-record

AVR-ассемблер генерирует код с абсолютной адресацией, и потому отпадает необюходимость в последующей компоновке и/или локализации.

Установка AVR-ассемблера

На прилагаемом к книге компакт-диске, в папке `AVRTOOLS` находится архивный самораспаковывающийся файл `ASMPACK.EXE`, содержащий пакет установки

полнофункционального ассемблера. Наиболее подходящий, исходя из обстоятельств, вариант этого пакета можно также бесплатно загрузить с Web-сайта компании Atmel: `www.atmel.com`.

Файл `ASMPACK.EXE` следует скопировать в какую-либо папку на жестком диске и затем запустить на выполнение, после чего все файлы из архива будут автоматически распакованы и сохранены в той же папке.

Теперь можно запустить на выполнение файл `SETUP.EXE` (перед этим должны быть закрыты все другие приложения!) и просто выполнять действия, предлагаемые программой установки.

В результате, кроме самого ассемблера, в подкаталоги `APPNOTES` и `DOC` будут также установлены файлы примеров и описаний (на английском языке) соответственно.

Синтаксис ассемблера

Программа пользователя (исходный файл) состоит, главным образом, из командных строк длиной максимум в 120 знаков, а также из следующих элементов.

- **Метки** (точки для перехода) — состоят из алфавитно-цифровых знаков, должны начинаться с буквы и заканчиваться двоеточием. Служат в качестве точки назначения при ветвлениях в программе и вызовах подпрограмм. Каждая командная строка может (но не обязательно) начинаться с метки.

  Примеры: `Метка1:`, `Test:`

- **Мнемоники команд** (символические представления машинных команд). Представляют собой последовательности символов, определенных компанией Atmel, которые обозначают в каждом конкретном случае некоторую команду. Зачастую сопровождаются операндами, к которым должна быть применена данная команда.

  Примеры: `andi` — обьединение с константой по логическому "И");

  `clc` — сброс флага переноса;

  `rjmp` — программный переход относительно текущего адреса.

- **Директивы** (управляющие указания для ассемблера). Представляют собой последовательности символов, определенных компанией *Atmel,* которые всегда начинаются с точки и не транслируются в машинные команды, а лишь используются в процессе ассемблирования.

  Примеры: `.db` — определяет байтовую константу в памяти программ;

  `.def` — задает символическое имя для регистра;

  `.org` — устанавливает счетчик команд в абсолютный адрес.

- **Комментарий**. Всегда начинается с символа ";" (точка с запятой) и добавляется в программу для лучшего ее понимания. Все знаки от ";" до конца строки ассемблером игнорируются.

  Пример: `; Это комментарий`

Командная строка может быть представлена в ассемблере в одной из четырех форм (то, что указано, в квадратных скобках, — опционально):

- [метка:] директива [операнды] [комментарий];
- [метка:] команда [операнды];
- [метка:] комментарий;
- пустая строка.

Примеры корректного исходного кода:

```
Marke1: .EQU var1 = 100 ; Директива - присваивает переменной var1
                        ; значение 100
        .EQU var2 = 200 ; Директива - присваивает переменной var2
                        ; значение  200
Test3A: rjmp Test3A     ; Команда, бесконечный цикл
Comment:                ; Строка комментария
```

Работа с *AVR*-ассемблером

На рис. 13.3. показана панель инструментов AVR-ассемблера для Windows (WAVRASM).

Рис. 13.3. Панель инструментов AVR-ассемблера

Ассемблер с интегрированным редактором ориентирован на привычную Windows-среду, и потому мы кратко рассмотрим только некоторые особенности пользовательского интерфейса.

Кнопки панели инструментов

Назначение кнопок панели инструментов AVR-ассемблера представлено в табл. 13.1.

Таблица 13.1. Назначение кнопок панели инструментов AVR-ассемблера

Кнопка	Назначение
	Создает новый исходный файл. Если исходный файл должен обрабатываться в интегрированном редакторе, то его размер не должен превышать 28 Кбайт. Исходные файлы большего размера можно создать с помощью какого-либо внешнего редактора (при этом допустимо применять только такой текстовый редактор, который не вставляет в текст управляющие символы), а затем — ассемблировать
	Открывает уже существующий исходный файл. Для каждого открываемого исходного файла создается новое окно редактора
	Обновляет уже открытый файл. Если, например, исходный файл и файл листинга открыты, и выполняется новое ассемблирование, то с помощью этой кнопки можно обновить файл листинга
	Сохраняется содержимое окна, активного в данный момент. Эта кнопка доступна только тогда, когда данные были изменены

Таблица 13.1. Окончание

Кнопка	Назначение
	Выделенный текст вырезается и помещается в буфер обмена
	Выделенный текст копируется в буфер обмена
	Текст, находящийся в буфере обмена, вставляется в текущей позиции курсора
	Отменяет последнее выполненное действие
	Функция поиска некоторого фрагмента текста
	Продолжение поиска ранее определенного фрагмента текста
	Печать содержимого активного в данный момент окна

Меню AVR-ассемблера WAVRASM

Меню **File** и **Edit** по своему назначению аналогичны одноименным меню в любом другом Windows-приложении и потому в дополнительном разъяснении не нуждаются. Меню **Search** (Поиск) содержит команды поиска произвольных строк текста в файле.

Команда **Assemble** (Ассемблировать) запускает ассемблирование исходного файла, отображаемого в активном в данный момент окне. В том случае, если с момента последнего ассемблирования исходный файл был изменен, перед трансляцией он сохраняется при условии, что в диалоговом окне, вызываемом по команде меню **Options** (Параметры), был установлен флажок **Save before assemble** (рис. 13.4).

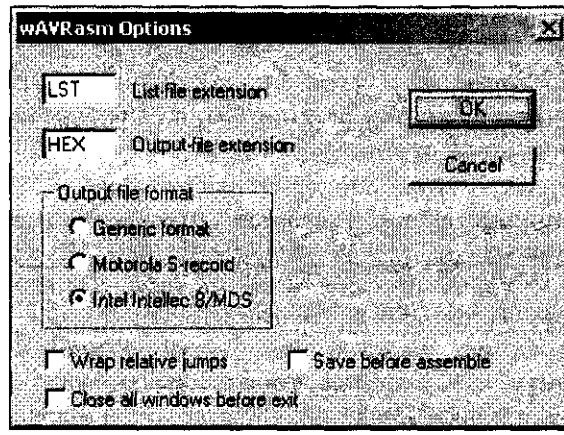


Рис. 13.4. Диалоговое окно параметров ассемблера WAVRASM

В этом окне к полям **Расширение файла листинга** и **Расширение загрузочного файла** можно указать расширения, отличные от выбранных по умолчанию `.lst` и `.hex`.

Желаемый формат файла `.hex` выбирают с помощью переключателей из набора **Output file format** (Формат загрузочного файла). Флажок **Wrap relative jumps** активизирует круговую адресацию для микроконтроллеров AVR с памятью программ объемом 4 Кслов. Если установлен флажок **Close all windows before exit**, то перед завершением работы ассемблера будут автоматически закрыты все окна. При установленном флажке **Save before assemble** исходный файл сохраняется перед каждым ассемблированием автоматически, в противном случае пользователю будет задан вопрос о том, хочет ли он сохранить исходный файл ввиду изменений с момента последнего ассемблирования.

Меню Window

Меню **Window** показано на рис. 13.5.

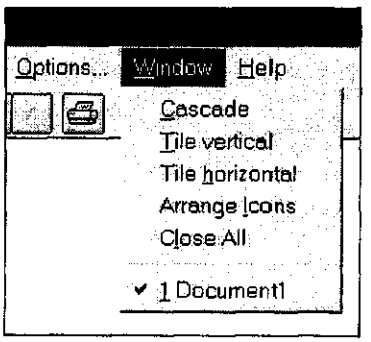


Рис. 13.5. Меню **Window**

Команда меню **Window → Cascade** размещает все окна таким образом, что виден заголовок каждого окна, однако верхние окна закрывают нижние. По команде **Window → Tile Vertical** окна размещаются на экране по вертикали одно над другим, а по команде **Window → Tile Horizontal** — по горизонтали. Команда меню **Window → Arrange Icons** упорядочивает свернутые окна, а команда **Window → Close All** — закрывает все открытые окна.

Поиск ошибок

На рис. 13.6 показано окно AVR-ассемблера, в котором открыто окно исходного файла `Beispiel.asm` и листинга `Beispiel.lst`. Исходный файл был ассемблирован по команде меню **Assemble**, и в окне сообщений **Message** появилось сообщение об ошибке. Если щелчкнуть мышью на этом сообщении, то строка в исходном файле, содержащая ошибку, будет выделена красным цветом. По двойному щелчку мышью на сообщении об ошибке курсор перемещается в строку исходного файла, содержащую ошибку.

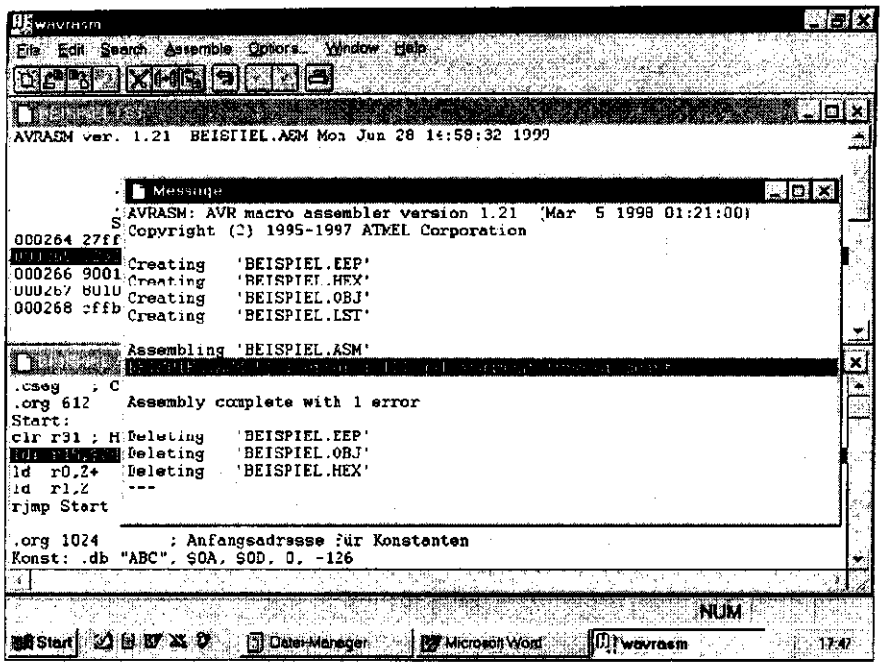


Рис. 13.6. Окно AVR-ассемблера

Сообщение об ошибке вида "BEISPIEL.ASM(5) : error : Illegal argument type or count" означает, что в строке 5 исходного файла в качестве операнда был выбран регистр, что в данном случае недопустимо.

Во время установки ассемблера в подкаталог `APPNOTES` копируется файл `TUTOR1.ASM`, который в целях обучения специально содержит ошибки. Эти примеры призваны помочь новичку быстро освоить методы работы в среде AVR-ассемблера.

Директивы ассемблера

Директивы ассемблера не порождают исполняемых кодов операций, а лишь являются управляющими указаниями для самого ассемблера. Они инициализируют участки памяти, определяют константы в памяти, устанавливают счетчик команд на определенный адрес и т.д. Существующие директивы перечислены в табл. 13.2.

Таблица 13.2. Директивы AVR-ассемблера

Директивы	*Описание*
`.BYTE`	Отводит для переменной место в памяти
`.CSEG`	Сегмент кода
`.DB`	Определяет однобайтовую(ые) константу(ы)
`.DEF`	Назначает символическое имя регистру
`.DEVICE`	Сообщает ассемблеру о типе микроконтроллера
`.DSEG`	Сегмент данных

Таблица 13.2. Окончание

Директивы	Описание
.DW	Определяет двухбайтную(ые) константу(ы)
.ENDMACRO	Конец макроса
.EQU	Сопоставляет символическое имя с некоторым выражением
.ESEG	Сегмент памяти EEPROM
.EXIT	Завершает ассемблирование файла
.INCLUDE	Вставляет исходный код из другого файла
.LIST	Создает файл листинга
.LISTMAC	Отображение кода макроса в листинге
.MACRO	Начало макроса
.NOLIST	Отменяет создание файла листинга
.ORG	Определяет абсолютный адрес
.SET	Назначает символическое имя некоторому выражению

Директива .BYTE

Отводит переменной место в памяти. Формат:

```
[Метка:]   .BYTE  exp
```

где exp — простое арифметическое или логическое выражение.

Директива .BYTE может применяться только в сегменте данных (DSEG). Переменная привязана к адресу первого зарезервированной ячейки памяти. Именем переменной является метка. Количество зарезервированных байтов памяти определяется по значению выражения exp. Содержимого этого адреса памяти невозможно ни изменить, ни сбросить в 0. Следующая свободная ячейка памяти находится по адресу "Метка+Выражение" (в следующем примере — по адресу var1+1, то есть, — table+16).

Пример использования:

```
.equ tab_size = 16
.DSEG
   var1:  .BYTE 1              ; Резервируем 1 байт для var1
   table: .BYTE tab_size       ; Резервируем таблицу из 16 байтов
.CSEG
   ldi r30,low(table)          ; Младший байт указателя Z
   ldi r31,high(table)         ; Старший байт указателя Z
   ldd r1,Z+3                  ; Загрузить 4-й байт таблицы в r1
```

Директива .CSEG

Определяет сегмент кода. Формат:

```
.CSEG
```

Директива .CSEG определяет начало сегмента кода. Исходный файл может состоять из нескольких сегментов кода, которые затем во время ассемблирования объединяются в один. Если ни один сегмент кода не указан, то по умолчанию им является сегмент CSEG. Сегмент кода имеет собственный счетчик адресов, указы-

вающий на 16-разрядную ячейку памяти (слово). С помощью директивы .ORG этот счетчик может быть установлен на определенный адрес памяти программ.

Пример использования:

```
.CSEG
KonstB: .db $1A, $2B    ; Определяем константу в 2 байта
KonstW: .dw $1A2B       ; Определяем константу в 1 слово
Progr: mov r0,r1        ; Исполняемая команда
```

Директива .DB

Определяет однобайтную константу(-ы) в памяти программ или в памяти EEPROM. Формат:

```
[Метка:] .DB exp-list
```

где в качестве exp-list может выступать:

- ряд разделенных запятыми выражений, которые могут содержать арифметические и логические операции, и должны находиться в диапазоне от -128 до 255 (отрицательные выражения записываются в памяти в виде дополнения до двух);
- последовательность ASCII-символов, заключенных в кавычки.

Директива .DB может применяться в сегменте программ (CSEG) или в сегменте памяти EEPROM (ESEG). Если байты должны сохраняться в CSEG, а exp-list содержит более одного выражения, то первое выражение всегда сохраняется в младшем байте слова, второе — в старшем байте. Если количество выражений нечетное, то последнее выражение запоминается в младшем байте слова, а старший байт заполняется нулем.

Пример использования:

```
.CSEG
000000 Konst1: .db $1A, $2B, 12/3, "ABCD"
000000 2b1a
000001 4104
000002 4342
000003 0044
000004 Konst2: .db 4+$0B, $F0 & $73, 0b01100011>>1
000004 700f
000005 0031
.ESEG
000000 Konst3: .db -3
000000 fd
```

Директива .DEF

Назначает регистру символическое имя. Формат:

```
.DEF  Обозначение = Регистр
```

Директива .DEF назначает регистру символическое имя, по которому остальная часть программы может обращаться к этому регистру. Регистр может иметь несколько символических имен, а обозначение может быть позже в программе переопределено.

Пример использования:

```
.def temp = r16          ; К r16 можно обращаться
.def work = r16          ; по двум именам
.def ioreg = r0
.CSEG
000000 ef00 ldi temp,$f0
000001 b60f in ioreg,$3f
.def temp = r17          ; Переопределение. С этого момента
                         ; по имени temp - обращение к r17
000002 e11a ldi temp,$1A
```

Директива .DEVICE

Сообщает ассемблеру о типе контроллера. Формат:

```
.DEVICE   AT90S1200 | AT90S2313 | AT90S4414 | AT90S8515
```

Директива .DEVICE сообщает ассемблеру о том, на каком микроконтроллере AVR должна выполняться созданная программа. Если в исходном файле появляется команда, которая не поддерживается данным контроллером, или происходит выход за границы памяти, то ассемблер выдает в своем окне сообщений и в листинге программы соответствующее предупреждение. Если директива .DEVICE отсутствует, то ассемблер допускает, что разрешены все команды, а границы памяти отсутствуют.

Пример использования:

```
.device AT90S1200
.CSEG
.org 511
0001ff 2c10 mov r1,r0
000200 2c32 mov r3,r2
BEISP2.ASM(6): warning: 'LPM' not supported on this device
000201 95c8 lpm
```

Дополнительное предупреждение в окне сообщений:

```
Total : 514 words
Warning : 512 words are maximum for this device
```

Директива .DSEG

Определяет сегмент данных. Формат:

```
.DSEG
```

Директива .DSEG определяет начало сегмента данных. Исходный файл может состоять из нескольких сегментов данных, которые затем в процессе ассемблирования объединяются, размещаясь друг за другом. Сегмент данных обычно состоит исключительно из директив .BYTE и меток. Кроме того, он использует собственный счетчик адреса, указывающий на 8-разрядные ячейки памяти (байты). С помощью директивы .ORG этот счетчик может устанавливаться на определенный адрес в памяти SRAM.

Пример использования — см. пример использования директивы .BYTE.

Директива .DW

Определяет константу(ы) размером в слово в сегменте программ или в сегменте EEPROM. Формат:

```
[Метка:] .DW exp-list
```

где в качестве exp-list может выступать ряд разделенных запятыми выражений, которые могут содержать арифметические и логические операции и должны лежать в диапазоне от -32768 до 65535. Отрицательные выражения записываются в памяти в виде дополнения до двух.

Директива .DW может применяться в сегменте программ (CSEG) или в сегменте памяти EEPROM (ESEG). Если в ячейки ESEG (8-разрядные) записываются слова, то вначале записывается младший байт, а старший байт размещается по адресу, следующему за адресом младшего байта.

Пример использования:

```
.CSEG
000000 Konst1: .dw $1A2B, 196512/3
000000 1a2b
000001 ffe0
000002 Konst2: .dw 1024+$0B, $FF00&$7654, $A987>>4
000002 040b
000003 7600
000004 0a98
.ESEG
000000 Konst3: .dw -3, 100, $100
000000 fd ff
000002 64 00
000004 00 01
```

Директива .ENDMACRO

Обозначение окончания макроса. Формат:

```
.ENDMACRO
```

Директива .ENDMACRO указывает на конец макроопределения. Более подробно определение макросов рассматривается ниже в описании директивы .MACRO.

Пример использования — см. пример к директиве .MACRO.

Директива .EQU

Назначает выражению символическое имя. Формат:

```
.EQU  Обозначение = exp
```

где exp — простое арифметическое или логическое выражение.

Директива . EQU присваивает обозначению некоторое значение. Остальная часть программы может оперировать с обозначением как с выражением. В данном случае речь идет об определении константы, и потому обозначению нельзя позже в программе присвоить новое значение.

Пример использования:

```
.equ Offset = $0100+100
.equ Div = 256/8
.equ Next = Div<<2
.CSEG
000000 e200 ldi r16,Div
000001 e6e4 ldi r30,low(Offset)
000002 e0f1 ldi r31,high(Offset)
000003 e810 ldi r17,Next
.ESEG
Сегмент EEPROM
Format:
```

Директива .ESEG

Определяет начало сегмента EEPROM. Формат:

```
.ESEG
```

Исходный файл может состоять из нескольких сегментов EEPROM, которые затом в процессе ассемблирования становятся единым сегментом, состоящим из расположенных друг за другом отдельных сегментов EEPROM. Сегмент EEPROM использует собственный счетчик адреса, который указывает на 8-разрядные ячейки памяти (байты). С помощью директивы .ORG этот счетчик может устанавливаться на определенный адрес в памяти EEPROM.

Директива .EXIT

Завершает ассемблирование файла. Формат:

```
.EXIT
```

Директива .EXIT указывает ассемблеру завершить в этом месте процесс ассемблирования, не достигнув конца файла (EOF) как это происходит в обычном режиме. Если ассемблер встречает директиву .EXIT в файле, вставленном с помощью директивы .INCLUDE, то на этом прерывается ассемблирование вставленного файла и продолжается со следующего шага в главном файле.

Директива .INCLUDE

Вставляет в исходный файл другой файл. Формат:

```
.INCLUDE "Имя файла"
```

Директива вставляет в этом месте программы содержимое другого исходного файла, а затем указывает ассемблеру транслировать вставляемый файл до его окончания или до директивы .EXIT и добавить полученный объектный код к исходной программе. Вставляемый файл, в свою очередь, также может включать собственные вставляемые файлы.

Пример использования:

В некоторой папке хранится исходный файл iodefs.asm следующего содержания:

```
.equ sreg = $3f
.equ sphigh = $3e
.equ splow = $3d
```

Для того чтобы можно было использовать определения, этот файл вставляется в один из файлов в той же папке:

```
 .include "iodefs.asm"
.equ sreg = $3f         ; Эти три строки
.equ sphigh = $3e       ; вставляются
.equ splow = $3d        ; ассемблером
.CSEG
000000 b60d in r0,splow ; splow  теперь известно
```

Директива .LIST

Активизирует создание файла листинга. Формат:

```
.LIST
```

Наряду с объектным и загрузочным файлом, ассемблер также создает файл листинга, содержащий команды из исходного файла вместе с соответствующими им кодами операций и адресами. Директива `.LIST` указывает ассемблеру начать с этого места создание листинга программы. Вместе с директивой `.NOLIST` директива `.LIST` может применяться для отключения отображения определенных фрагментов файла.

Пример использования — см. пример к директиве `.NOLIST`.

Директива .LISTMAC

Отображение кода макроса в листинге. Формат:

```
.LISTMAC
```

Директива `.LISTMAC` указывает ассемблеру при вызове макроса вставлять все его команды в этом месте листинга программы. По умолчанию, указываются только имя и параметры макроса.

Первый пример (без `.LISTMAC`):

```
.macro subi16           ; Начало макроопределения
       subi @1,low(@0)  ; Вычесть константу-слово
       sbci @2,high(@0); из регистровой пары
.endmacro               ; Конец макроопределения
                        ; .CSEG
000000 + subi16 $ABCD,r16,r17 ; В листинге - только вызов макроса
000002 0000 nop
```

Пример использования директивы `.LISTMAC`:

```
.LISTMAC
.CSEG
000000 + subi16 $1234,r18,r19 ; Весь макрос
000000 5324 subi r18,low(0x1234)
000001 4132 sbci r19,high(0x1234)
.endmacro                       ; Конец макроопределения
000002 0000 nop                 ; появляется в листинге
```

Директива .MACRO

Обозначение начала макроса. Формат:

```
. MACRO Имя_макроса
```

Директива .MACRO отмечает начало макроса — набора команд, выполняемого при вызове макроса, подобно подпрограмме, но в отличие от подпрограммы, которая записывается в память только один раз, команды макроса размещаются в памяти столько раз, сколько вызывается макрос. Макросы обычно применяют во фрагментах, критичных по времени, где три тактовых импульса для выполнения команды rcall и четыре для выполнения команды ret — непозволительная роскошь, а также в случаях, когда должны определяться максимально эффективные собственные команды пользователя.

Всегда, когда в программе встречается имя макроса, макрос как бы "разворачивается" (то есть, его команды вставляются в программу). Макрос может принимать до десяти параметров. Эти параметры в определении макроса пронумерованы от @0 до @9 в порядке их следования. При вызове макроса параметры передаются в виде отдельного списка и отделяются друг от друга запятыми.

Определение макроса заканчивается директивой .ENDMACRO. Если команды макроса должны отображаться в каждом месте его вызова в листинге, то в исходном файле должна быть указана директива .LISTMAC. В противном случае в листинге будет отображаться только имя макроса с параметрами. Вхождение макроса помечается в листинге плюсом перед его именем.

Пример использования — см. пример к директиве .LISTMAC.

Директива .NOLIST

Отключает создание листинга. Формат:

```
.NOLIST
```

Наряду с объектным и загрузочным файлом, ассемблер также создает файл листинга, содержащий команды из исходного файла вместе с соответствующими им кодами операций и адресами. Изначально считается активной директива .LIST, и директива .NOLIST указывает ассемблеру прекратить создание листинга. Вместе с директивой .LIST директива .NOLIST может применяться для отмены отображения определенных фрагментов файла.

Пример использования:

```
.nolist                 ; Создание листинга отключено
.include "iodefs.asm"   ; Вставленные файлы
.include "const.def"    ; не отображаются в файле листинга
.list                   ; Создание листинга включено
```

Директива .ORG

Определяет абсолютный адрес. Формат:

```
[Метка:] .ORG  exp
```

где exp — простое арифметическое или логическое выражение.

Директива .ORG устанавливает счетчик адреса текущего сегмента на абсолютный адрес exp. В сегменте данных, обозначенном директивой .DSEG, устанавливается счетчик адреса в SRAM; в сегменте кода, обозначенном директивой .CSEG, — счетчик адреса во флэш-памяти команд; а в сегменте EEPROM, обозначенном директивой .ESEG, — счетчик адреса в памяти EEPROM. Если директива .ORG сопровождается меткой, то этой метке присваивается значение exp.

Если директива .ORG не указана, то счетчик адреса команд и счетчик адреса в памяти EEPROM изначально содержат 0, в то время как счетчик адреса в памяти SRAM начинает со значения 96 ($60), поскольку в начальной области SRAM размещены 32 рабочих регистра и 64 регистра ввода/вывода.

Счетчики адреса в EEPROM и SRAM адресуют 8-разрядные ячейки памяти (байты), в то время как счетчик адреса команд адресует 16-разрядные ячейки памяти (слова).

Пример использования:

```
.dseg                    ; Начало сегмента данных
000060 .byte 1           ; Адрес по умолчанию
.org 11*9+5
000068 var1: .byte 1     ; Один байт по адресу 104($68) в памяти SRAM
.eseg                    ; Начало сегмента EEPROM
.org 72/8-3
000006 var2: .dw $FEFF ; Слово по адресам 6 и 7 в памяти EEPROM
000006 ff fe
.cseg                    ; Начало сегмента программ
.org $100
000100 2c01 Lab: mov r0,r1 ; Команда по адресу  256
```

Директива .SET

Назначает выражению символическое имя. Формат:

```
.SET Обозначение = exp
```

где exp — простое арифметическое или логическое выражение.

Директива .SET присваивает обозначению некоторое значение. В дальнейшем в программе к этому обозначению можно обращаться как к значению. Ниже в программе обозначению можно сопоставить новое значение.

Пример использования:

```
.set Div = 256/8         ; Определение "Div" ($20)
.set Next = Div<<2       ; Определение "Next" ($80)
.CSEG
000000 e200 ldi r16,Div
000001 e810 ldi r17,Next
.set Div = $26           ; Новое определение для "Div"
000002 e206 ldi r16,Div
000003 e810 ldi r17,Next ; "Next" остается неизмененным
```

Выражения

AVR-ассемблер может обрабатывать выражения в исходном файле, которые могут состоять из операндов, функций и операторов. Все выражения занимают в памяти по 4 байта (32 бита).

Операнды

В исходном файле могут применяться следующие операнды:

- определенные пользователем метки — им ассемблер назначает то значение, которое содержит счетчик команд в месте их появления в исходном файле;
- переменные, определенные пользователем с помощью директивы `.SET`;
- переменные, определенные пользователем с помощью директивы `.EQU`;
- целочисленные константы — здесь допустимы следующие форматы:
  - десятичные (по умолчанию) — `10`, `255`;
  - шестнадцатиричные — `0x0A`, `0xFF` или `$0A`, `$FF`;
  - двоичные — `0b00001010`, `0b11111111`;
- символы ASCII — `'A'`, `'a'`;
- последовательности символов ASCII (строки) — "Это строка";
- содержимое счетчика команд в текущий момент времени.

Функции

Предопределены следующие функции:

- `LOW(exp)` — возвращает младший байт выражения `exp`.
  Пример: `LOW($9ABC)` возвращает `$BC`.
- `HIGH(exp)` — возвращает старший байт выражения `exp`.
  Пример: `HIGH($9ABC)` возвращает `$9A`.
- `BYTE2(exp)` — имеет то же назначение, что и `HIGH(exp)`.
- `BYTE3(exp)` — возвращает третий байт выражения `exp`.
  Пример: `BYTE3($12345678)` возвращает `$34`.
- `BYTE4(exp)` — возвращает четвертый байт выражения `exp`.
  Пример: `BYTE4($12345678)` возвращает `$12`.
- `LWRD(exp)` — возвращает разряды 0...15 выражения `exp`.
  Пример: `LWRD($12345678)` возвращает `$5678`.
- `HWRD(exp)` — возвращает разряды 16...31 выражения `exp`.
  Пример: `HWRD($12345678)` возвращает `$1234`.
- `PAGE(exp)` — возвращает разряды 16...21 выражения `exp`.
  Пример: `PAGE($56789)` возвращает `$5`.
- `EXP2(exp)` — возвращает значение 2^{exp}.

Пример: `EXP2(4)` возвращает `16`.

- `LOG2(exp)` — возвращает целую часть $\log_2(exp)$.

  Пример: `LOG2(5)` возвращает `2`.

Операторы

AVR-ассемблер поддерживает следующие операторы (приоритетность операторов уменьшается по мере их следования в списке; выражения, заключенные в скобки, всегда вычисляются в первую очередь).

- Логическое "НЕ":
  - символ: `!`;
  - унарный оператор; если значение выражения равно нулю, то возвращается 1, в остальных случаях — 0;
  - приоритет: 14
  - пример: `ldi r16,!0xf0 ; Загружается 0x00 в r16.`
- Поразрядное "НЕ":
  - символ: ~;
  - унарный оператор; все разряды выражения возвращаются инвертированными (дополнение до единицы);
  - приоритет: 14
  - пример: `ldi r16,~0xf0 ; Загружается 0x0f в r16.`
- Унарный минус:
  - символ: -;
  - унарный оператор; возвращается арифметическое отрицание выражения (дополнение до двух);
  - приоритет: 14;
  - пример: `ldi r16,-2 ; Загружается 0xfe (-2) в r16.`
- Умножение:
  - символ: *;
  - бинарный оператор; возвращает результат умножения двух выражений;
  - приоритет: 13;
  - пример: `ldi r16,LabelA*2 ; Загружается LabelA * 2 в r16.`
- Деление:
  - символ: /;
  - бинарный оператор; возвращает целое от деления выражения, указанного слева, на выражение, указанного справа;
  - приоритет: 13;
  - пример: `ldi r16,LabelA/2 ; Загружается LabelA / 2 в r16.`
- Сложение:
  - символ: +;
  - бинарный оператор; возвращает сумму двух выражений;
  - приоритет: 12;
  - пример: `ldi r16,k1+k2 ; Загружается k1 + k2 в r16.`

- Вычитание:
  - символ: -;
  - бинарный оператор; возвращает разницу двух выражений;
  - приоритет: 12;
  - пример: `ldi r16,k1-k2 ; Загружается k1 - k2 в r16.`
- Сдвиг влево:
  - символ: <<;
  - бинарный оператор; выражение, указанное слева, сдвигается влево на количество разрядов, указанное справа.
  - приоритет: 11;
  - пример:
```
ldi r16,1<<5 ; В r16 загружается $20
             ; (0b00000001 сдвинутое влево на 5 позиций).
```
- Сдвиг вправо:
  - символ: >>;
  - бинарный оператор; выражение, указанное слева, сдвигается вправо на количество разрядов, указанных справа;
  - приоритет: 11;
  - пример:
```
ldi r16,8>>1 ; В r16 загружается $04
             ; (0b00001000 сдвинутое вправо на 1 позицию).
```
- Меньше:
  - символ: <;
  - бинарный оператор; если выражение со знаком, указанное слева, меньше выражения со знаком, указанного справа, то возвращается 1, в противном случае возвращается 0;
  - приоритет: 10;
  - пример:
```
ori r16,k1<k2 ; Если k1 < k2, то
              ; в r16 установится разряд 0.
```
- Больше или равно:
  - символ: <=;
  - бинарный оператор; если выражение со знаком, указанное слева, меньше или равно выражению со знаком, указанному справа, то возвращается 1,
  - в противном случае возвращается 0;
  - приоритет: 10;
  - пример:
```
ori r16,k1<=k2 ; Если k1 <= k2, то в r16
               ; устанавливается разряд 0.
```
- Больше:
  - символ: >;
  - бинарный оператор; если выражение со знаком, указанное слева, больше выражения со знаком, указанного справа, то возвращается 1, в противном случае возвращается 0;
  - приоритет: 10;
  - пример:
```
ori r16,k1>k2 ; Если k1 > k2, то в r16
              ; устанавливается разряд 0.
```

- Больше или равно:
  - символ: >=;
  - бинарный оператор; если выражение со знаком, указанное слева, больше или равно выражению со знаком, указанному справа, то возвращается 1,
  - в противном случае возвращается 0;
  - приоритет: 10;
  - пример:
```
ori r16,k1>=k2 ; Если k1 >= k2, то в r16
               ; устанавливается разряд 0.
```
- Равно:
  - символ: ==;
  - бинарный оператор; если выражение со знаком, указанное слева, равно выражению со знаком, указанному справа, то возвращается 1, в противном случае возвращается 0;
  - приоритет: 9;
  - пример:
```
ori r16,k1==k2 ; Если k1 == k2, то в r16
               ; устанавливается разряд 0.
```
- Не равно:
  - символ: !=;
  - бинарный оператор; если выражение со знаком, указанное слева, не равно выражению со знаком, указанному справа, то возвращается 1, в противном случае возвращается 0.
  - приоритет: 9;
  - пример:
```
.set flag=(k1!=k2) ; Если  k1 не равно k2, то
                   ; flag "установлен"
```
- Поразрядное "И":
  - символ: &;
  - бинарный оператор; возвращает результат логической операции "И" над каждой парой соответствующих разрядов выражений, указанных слева и справа;
  - приоритет: 8;
  - пример:
```
ldi r16,$d2&$91 ; Загружается $90 в r16.
```
- Поразрядное "Исключающее ИЛИ":
  - символ: ^
  - бинарный оператор; возвращает результат логической операции "Исключающее ИЛИ" над каждой парой соответствующих разрядов выражений, указанных слева и справа;
  - приоритет: 7;
  - пример:
```
ldi r16,$d2^$91 ; Загружается $43 в  r16.
```
- Порязрядное "ИЛИ":
  - символ: |;
  - бинарный оператор; возвращает результат логической операции "ИЛИ" над каждой парой соответствующих разрядов, указанных слева и справа;
  - приоритет: 6;
  - пример:
```
ldi r16,$d2|$91 ; Загружается $d3 в r16.
```

- Логическое "И":
  - символ: `&&`;
  - бинарный оператор; если оба выражения не равны нулю, то возвращается 1, в противном случае — 0;
  - приоритет: 5;
  - пример: `ldi r16,$d2&&$04 ; Загружается $01 в r16.`
- Логическое ИЛИ:
  - символ: `||`;
  - бинарный оператор; если хотя бы одно из двух выражений не равно нулю, то возвращается 1, в противном случае — 0;
  - приоритет: 4;
  - пример: `ldi r16,$d2||$00 ; Загружается $01 в r16.`

Работа с интегрированным редактором AVR-ассемблера базируется на общих подходах, применяемых в Windows-приложениях, поэтому материал данной главы можно считать исчерпанным.

14 Отладка программ в среде AVR-Studio

Программная среда AVR-Studio от компании Atmel — это очень полезный инструмент для ввода в эксплуатацию собственноручно написанных программ для микроконтроллеров семейства AVR. Пользователь на свое усмотрение вводит программу в действие или же тестирует ее и исправляет возможные ошибки. Для этого он может целенаправленно управлять ходом выполнения программы и приостанавливать ее в любых местах с целью контроля (например, проверять изменения, внесенные в содержимое регистров последними командами, или же просматривать ячейки памяти и содержимое регистров периферийных устройств, наподобие приемопередатчика UART, интерфейса SPI, таймера и т.д).

AVR-Studio работает или как программная оболочка для внутрисистемных эмуляторов Atmel ICEPRO или ICE200, реализованных компанией Atmel для семейства AVR, или как симулятор системы команд AVR, в котором поведение соответствующего микроконтроллера имитируется с помощью программного обеспечения. Если при открытии проекта обнаружится внутрисистемный эмулятор, подключенный к последовательному порту компьютера, то он будет автоматически выбран как целевая система. В противном случае, ход выполнения программы моделируется чисто программно.

Поскольку описание эмуляторов, которые сами по себе не являются составной частью AVR-Studio, а должны приобретаться у компании Atmel, выходит за тематические рамки этой книги, то в дальнейшем ограничимся только рассмотрением интегрированного симулятора. Среда AVR-Studio версии 2.0 находится на прилагаемом к книге компакт-диске в папке `\Atmel\Tools` (файл `AStudio2.exe`). Она работает под управлением операционной системы Windows 95/98 или Windows NT. Версии для MS-DOS или Windows 3.11 отсутствуют.

Установка AVR-Studio

На прилагаемом к книге компакт-диске в папке `\Atmel\Tools` находится самораспаковывающийся архив `ASTUDIO.EXE`, содержащий полную версию AVR-Studio. Поскольку среда AVR-Studio постоянно совершенствуется, то самую свежую ее версию можно также бесплатно загрузить с домашней страницы компании Atmel (`www.atmel.com`).

Файл `ASTUDIO.EXE` следует скопировать на жесткий диск и запустить на выполнение. Все файлы архива автоматически распакуются в текущую папку. В ней следует найти и запустить на выполнение файл `SETUP.EXE` (перед этим все приложения необходимо закрыть!). Далее от пользователя требуется просто придерживаться подсказок программы установки, выполняя предлагаемые действия.

По окончании установки будет создана группа программ с пиктограммой **Studio**, двойным щелчком на которой и запускается среда AVR-Studio.

Обзор

Наиболее важное окно Studio — это расположенное в верхнем левом углу (рис. 14.1) окно с исходным кодом программы. Это окно появляется автоматически при открытии объектного файла (объектный файл, как уже было сказано в главе 13, — это один из выходных файлов AVR-ассемблера).

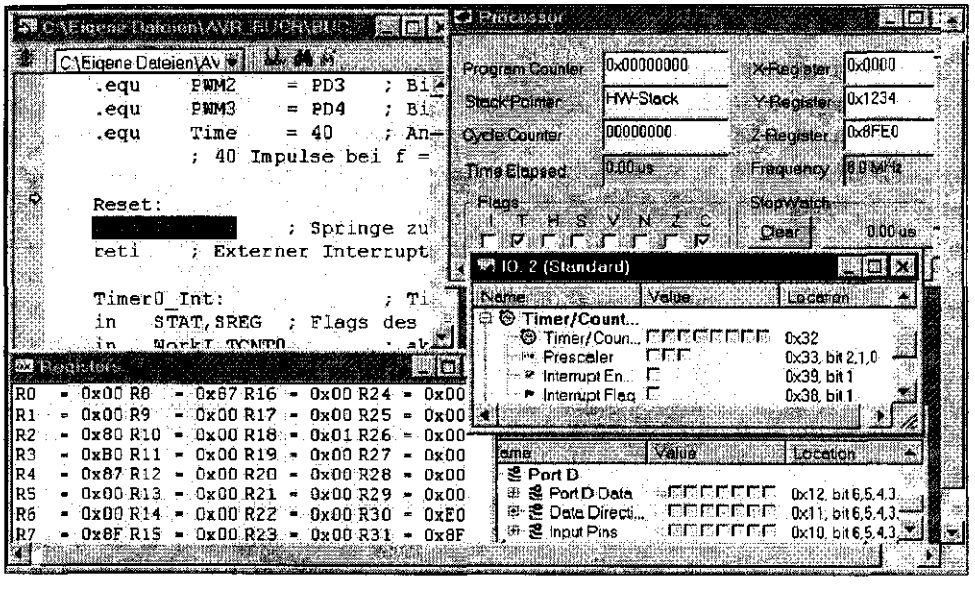


Рис. 14.1. Типичный пример экрана AVR-Studios при моделировании хода программы

Окно исходного кода отображает команды тестируемой программы, при этом курсор установлен на той команде, которая подлежит выполнению. Если вместо объектного файла загружается выходной файл `.hex` в формате Intel, то Studio работает на уровне дизассемблера. Программные метки и определения идентификаторов в этом файле отсутствуют.

Для того чтобы смоделировать ход программы, исходный файл должен прежде быть ассемблирован AVR-ассемблером (см. главу 13) или равноценным ассемблером других изготовителей. Если программа составлялась на языке высокого уровня C, то объектный файл, соответствующий стандарту Atmel, можно получить с помощью компилятора от компании IAR Systems

Как можно сделать вывод, глядя на рис. 14.1, пользователь для наибольшего удобства отслеживания хода выполнения программы, кроме исходного файла, может открыть еще целый ряд других окон. В его распоряжение предоставлены следующие окна (более подробно они рассматриваются в следующем разделе):

- **Registers** — отображает текущее содержимое всех 32 рабочих регистров r0...r31;
- **Processor** — помимо других данных, показывает содержимое счетчика команд, указателя стека и флагов;
- **Memory** — отображает содержимое памяти в областях ввода/вывода, флэш, EEPROM и SRAM;

- **IO** — содержит информацию о таких элементах аппаратной части микроконтроллера как регистры таймера, портов ввода/вывода, UART, сторожевого таймера, EEPROM и др.;
- **Watch** — содержит текущие значения и адреса определенных идентификаторов (например, переменных в программе на С);
- **Trace** — отображает последние 32 К х 96 бит выполнения программы;
- **Message** — выдает пользователю сообщения от Studio.

Для выполнения своей программы пользователь может избрать один из нескольких вариантов. К примеру, он может воспользоваться командой **Go** до достижения точки останова, или же работать с программой в пошаговом режиме (**Single Step**). В последнем случае он может пропускать подпрограммы или так же выполнять их пошагово. Существует также вариант выполнения программы до команды, в которой в данный момент установлен курсор (команда **Run To Cursor**). Кроме того, в распоряжении пользователя есть неограниченное количество точек останова (breakpoints), позволяющих временно остановить программу в требуемом месте, чтобы просмотреть содержимое регистров и состояние аппаратных ресурсов. Эти точки можно по необходимости активировать и деактивировать, а также сохранять после завершения работы с программой до следующего сеанса. Более подробно все возможности выполнения программы описаны ниже в разделе "Меню AVR-Studio".

При первой загрузке новой программы в Studio автоматически открывается только окно исходного кода. Набор остальных окон пользователь должен определить и разместить на экране на свое усмотрение. Среда Studio запоминает выбранное расположение окон, и автоматически восстанавливает его при всех последующих тестах данной программы.

В распоряжении пользователя есть также справочная система, с помощью которой, в случае возникновения какой-либо проблемы, он может получить важные указания. Вызывается она привычным способом — с помощью меню **Help**, расположенного у правого края строки меню Studio.

Если задержать указатель мыши над одной из кнопок панели инструментов, то появится краткое описание этой кнопки в форме всплывающей подсказки.

Окна AVR-Studio

Окно исходного кода

Окно исходного кода — самое важное окно Studio. Оно автоматически открывается, когда загружается объектный файл, и с его закрытием текущий сеанс завершается. Это окно содержит код, подлежащий выполнению в данный момент (рис. 14.2).

Studio помечает подлежащую выполнению команду черным штрихом. Если пользователь переместит программный курсор, то он все равно сможет обнаружить текущую команду, так как Studio окрашивает ее в таком случае в красный цвет.

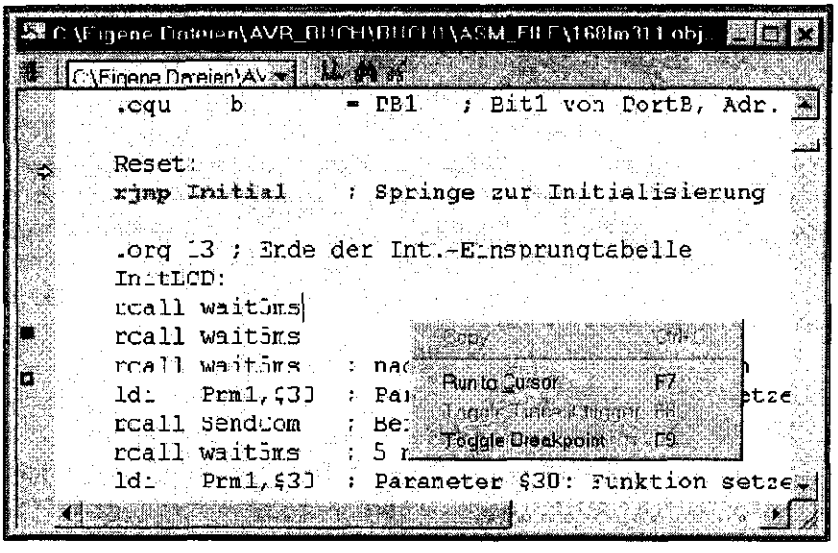


Рис. 14.2. Пример окна исходного кода AVR-Studio

На рис. 14.2 стрелка в левой колонке соответствует текущему состоянию счетчика команд, черный квадратик в той же колонке обозначает точку останова, а пустой квадратик под ним идентифицирует неактивную точку останова.

Если указатель разместить в пределах окна исходного кода и затем нажать правую кнопку мыши, то в позиции указателя появится маленькое меню (см. рис. 14.2), которое дает возможность скопировать текст в буфер обмена (команда **Copy**), выполнить программу до позиции курсора (команда **Run To Cursor**), переключить режим трассировки (команда **Toggle Trace & Trigger**) или точки останова (команда **Toggle Breakpoints**).

Объектный файл может состоять из нескольких модулей, из которых отображается только один. Если пользователь хочет сменить модуль (например, для расстановки точек останова или наблюдения за ними), то он может это сделать с помощью раскрывающегося списка, расположенного в верхнем левом углу окна.

Три кнопки, расположенные справа от поля выбора модуля, служат для переключения между уровнями "Исходный код"/"Дизассемблер", а также для вызова функций поиска и повторного поиска.

Окно ***Registers***

Окно **Registers** (рис. 14.3) показывает содержимое 32 рабочих регистров микроконтроллера, которое обновляется после выполнения каждой команды.

Значения, изменившиеся в результате последней отладочной операции (например, при пошаговом проходе или в результате достижения точки останова) отображаются красным шрифтом.

Для того чтобы изменить содержимое регистра, следует остановить выполнение программы и сделать два щелчка левой кнопкой мыши, с короткой паузой между ними, по выбранному регистру. Теперь можно задать новое содержимое в шестнадцатеричной форме. Внесенное изменение подтверждается нажатием клавиши <Enter> или отменяется нажатием клавиши <Esc>.

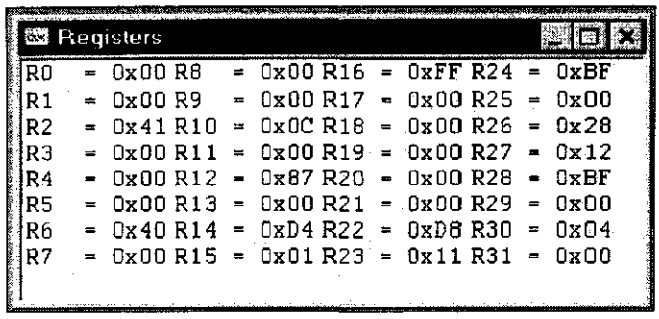


Рис. 14.3. Окно **Registers** AVR-Studio

Окно ***Processor***

Окно **Processor** (рис. 14.4) показывает содержимое счетчика команд (поле **Program Counter**), указателя стека (поле **Stack Pointer**), счетчика тактов (поле **Cycle Counter**), а также трех указателей X, Y и Z и флагов из регистра состояния SREG. Кроме того, оно предоставляет информацию о времени, затраченном программой, начиная от ее старта (поле **Time Elapsed**), а также о тактовой частоте микроконтроллера (поле **Frequency**).

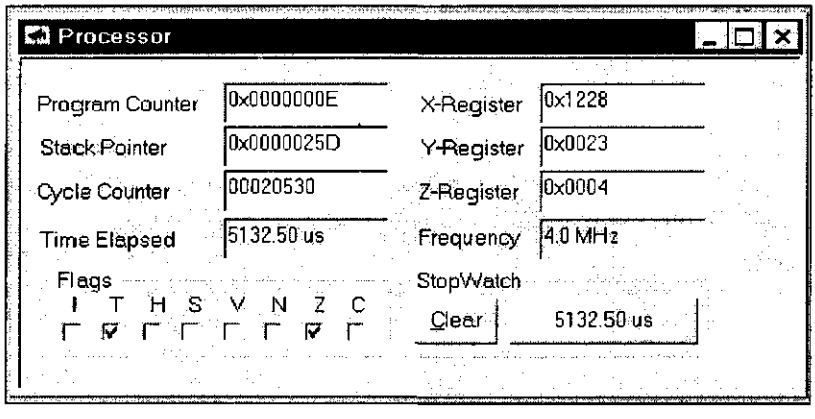


Рис. 14.4. Окно **Processor** AVR-Studio

В качестве дополнительного свойства, окно **Processor** содержит поле **Stop Watch**, отображающее время, прошедшее с момента запуска (или предыдущего сброса часов) до останова программы. С помощью кнопки **Clear** при остановленной программе часы можно сбросить в нуль. Это свойство прекрасно подходит для измерения времени выполнения определенных отрезков программы. Для переключения единиц измерения времени между мкс и мс следует щелкать мышью в поле показаний при условии, что нет переполнения в области мкс.

Значения полей, имеющих в окне **Processor** белый цвет фона, можно изменять после остановки программы. Содержимое окна **Processor** обновляется после каждой выполненной команды.

Счетчик команд содержит в шестнадцатеричной форме адрес команды, которая подлежит выполнению следующей. Изменив его содержимое вручную, можно

за один шаг продолжить выполнение программы с нового произвольного адреса. Содержимое указателя стека — это копия содержимого двух ячеек памяти SPH:SPL по адресу $3E и $3D в области ввода/вывода AVR. Если целевой процессор имеет только аппаратный стек (как, например, в случае с AT90S1200), то это отображается в поле **Stack Pointer** (см. рис. 14.1).

Счетчик тактов предоставляет информацию о числе тактовых импульсов с момента последнего сброса выполнения программы. Здесь идет речь о десятичном значении. Флаг регистра SREG, отмеченный "галочкой", содержит лог. 1, а сброшенному флажку соответствует лог. 0.

Окна ***Memory***

Окно **Memory** позволяет пользователю при необходимости контролировать или изменять некоторую область памяти AVR, например: память программ, память EEPROM, память данных и память ввода/вывода. Одновременно могут быть открыты сразу несколько окон **Memory** (рис. 14.5).

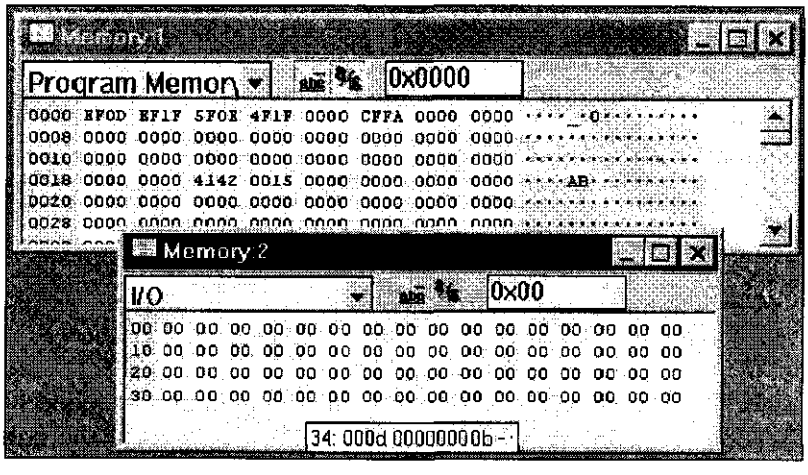


Рис. 14.5. Два окна **Memory** AVR-Studio

Тип области памяти выбирают с помощью раскрывающегося списка, расположенного в верхнем левом углу окна **Memory**. В примере на рис. 14.5 для верхнего окна (**Memory:1**) была выбрана память программ, а для нижнего (**Memory:2**) — область ввода/вывода.

С помощью кнопки, расположенной справа от раскрывающегося списка, пользователь может подключить дополнительную колонку с ASCII-значениями отображенной области памяти (как в показанном выше примере памяти программ). Еще одна кнопка, расположенная справа от кнопки **ASCII**, служит для переключений между представлениями значений в памяти по 8 бит и по 16 бит, а следующее поле показывает адрес ячейки памяти, на которой в данный момент установлен курсор.

По умолчанию, для представления данных в окне **Memory** используется шестнадцатеричная система счисления. Если указатель мыши разместить над какой-либо ячейкой памяти, то в виде всплывающей подсказки отобразится содержимое

этой ячейки в других системах исчисления (пример на рис. 14.5 — для ячейки памяти 0x34 в области ввода/вывода).

Содержимое ячейки памяти в окне **Memory** можно легко изменить. Введенные значения расцениваются как шестнадцатеричные числа, при этом новое значение попадает в ячейку памяти сразу же после каждого нажатия клавиши. Если это нежелательно (например, в случае применения эмулятора при доступе на запись к регистру UDR приемопередатчика UART инициируется новая передача, хотя в этом случае байт еще неполный), то новое значение можно альтернативно вводить в текстовом поле. Для того чтобы открыть это поле, следует дважды щелкнуть мышью на соответствующей ячейке памяти. В этом случае вводимое значение записывается в ячейку только тогда, когда текстовое поле будет закрыто по нажатию клавиши <Enter>. Ошибочный ввод можно отменить с помощью клавиши <Esc> (при этом текстовое поле тоже будет закрыто).

Если вводимые значения должны интерпретироваться как шестнадцатеричные или двоичные числа, то они всегда сопровождаются префиксом "0x" или "0b" соответственно. Если такой префикс отсутствует или указано "0d", то значение рассматривается как десятичное число.

Значения, которые изменились с момента последней операции отладки, отображаются красным шрифтом. Адреса восьмиразрядных ячеек памяти и ASCII-символы выделены серым фоном, а адреса 16-разрядных ячеек памяти — голубым фоном.

Если разместить указатель в пределах окна **Memory** и щелкнуть правой кнопкой мыши, то в позиции указателя появится контекстное меню (рис. 14.6), которое позволяет изменить систему счисления (команды **Hexadecimal/Decimal/Binary/None**); отобразить дополнительную колонку с ASCII-символами (команда **Show ASCII**); представить данные в виде 16-разрядных слов (команда **Show as Word**); указать количество байт, отображаемых в одной строке — от 2 до 32 (команда **Bytes per line**); изменить порядок следования старшего и младшего байтов (команда **Swap Bytes**), а также отобразить адреса байтов вместо адресов слов (команда **Byte Addresses**).

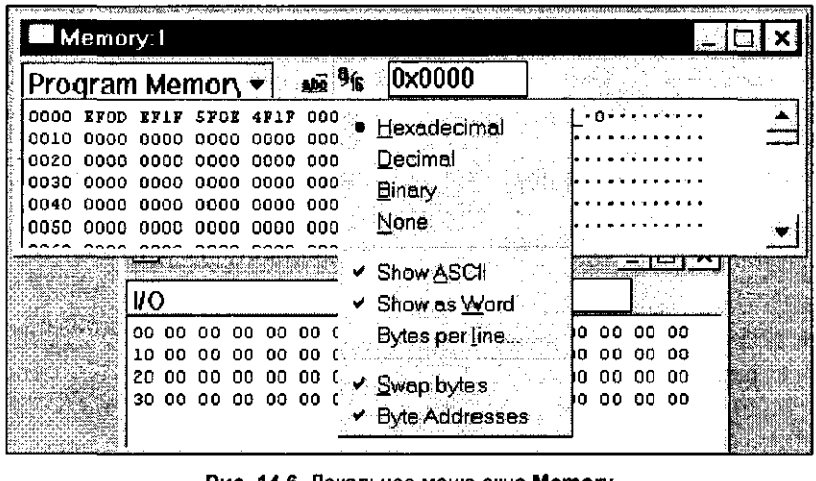


Рис. 14.6. Локальное меню окна **Memory**

Команды **Swap Bytes** и **Byte Addresses** доступны только для памяти программ. Пользователь может загрузить данные в формате Intel Hex в память SRAM или EEPROM или считать данные из памяти этого типа в файл. Для этого следует выбрать в меню **File** команду **Up/Download Memories**.

Размер, расположение и вид области памяти, выбранные в окне **Memory**, сохраняются для дальнейшего вызова соответствующего объектного файла.

Окно IO

Окно **IO** служит для того, чтобы контролировать содержимое регистров ввода/вывода в целевой системе и, при необходимости, изменять его. При этом одновременно могут быть открыты сразу несколько окон **IO** (рис. 14.7).

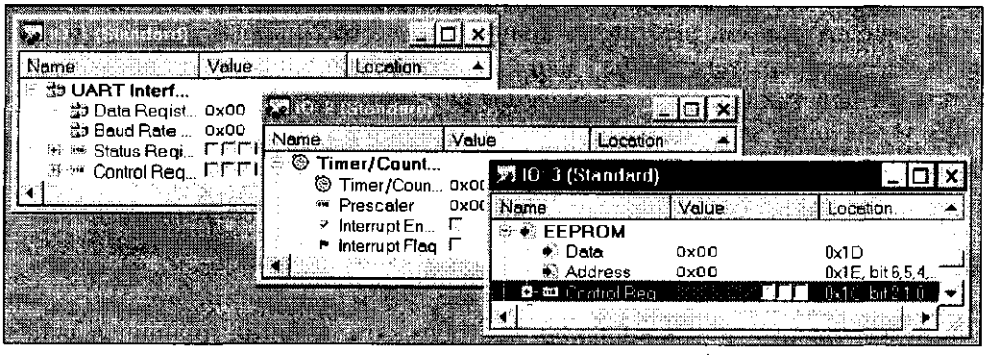


Рис. 14.7. Три одновременно открытых окна IO AVR-Studio

Окно **IO** состоит из трех колонок (рис. 14.8). Левая колонка содержит имена регистров или групп регистров. Если щелкнуть мышью по пиктограмме развертывания или дважды щелкнуть на имени, то содержимое соответствующего регистра отобразится детально. Повторный двойной щелчок возвращает элемент колонки в исходное состояние.

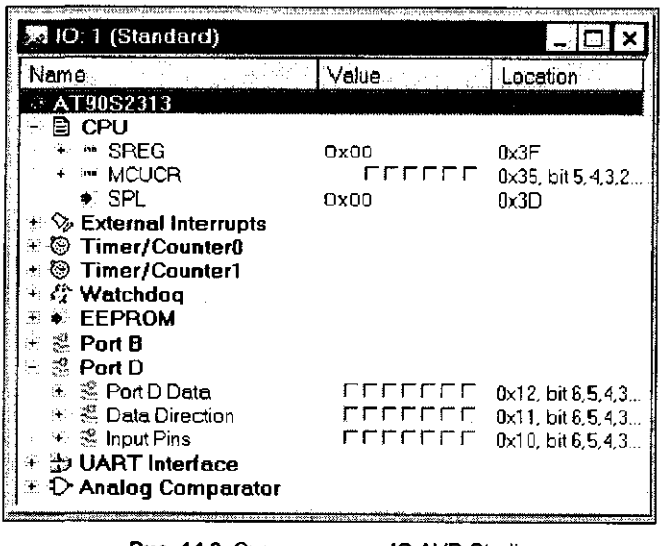


Рис. 14.8. Структура окна IO AVR-Studio

Средняя колонка показывает содержимое регистра, которое может изменяться пользователем. В правой колонке находятся адреса однобайтовых регистров и дополнительно номера маскированных разрядов. Двойной щелчок в этой колонке открывает диалоговое окно свойтв для выбранного пункта.

Содержимое регистра может отображаться по-разному. Если на имени регистра щелкнуть правой кнопкой мыши, то откроется диалоговое окно, в котором пользователь может сделать выбор между шестнадцатеричным, десятичным или двоичным представлением. Дополнительно можно воспользоваться отображением в виде флажка (см. регистры порта D на рис. 14.8).

Содержимое ячейки памяти можно изменить в окне **IO** или просто переписав его, или дважды щелкнув мышью на старом значении. В обоих случаях появляется текстовое поле, содержащее старое значение. Если вводимые значения должны интерпретироваться как шестнадцатеричные или двоичные числа, то они всегда сопровождаются префиксом "0x" или "0b" соответственно. Если такой префикс отсутствует или указано "0d", то значение рассматривается как десятичное число. Вводимое значение записывается только тогда, когда текстовое поле будет закрыто по нажатию клавиши <Enter>. Ошибочный ввод можно отменить с помощью клавиши <Esc> (при этом текстовое поле тоже будет закрыто).

Значения, которые изменились с последней операции отладки. отображаются красным шрифтом. Пользователь может конфигурировать окно **IO** на свой вкус. Неиспользуемые пункты-регистры можно удалить или добавить дополнительные. Кроме того, пункты можно скрывать и отображать, а также индивидуально настраивать свойства каждого отдельного пункта.

Для того чтобы добавить новый пункт-регистр, необходимо выделить пункт в месте вставки и нажать клавишу <Insert> или выбрать команду **Add** контекстного меню. В результате AVR-Studio добавит в список неинициализированный подпункт выбранного пункта-регистра и откроет для него окно свойств.

Для того чтобы удалить пункт-регистр, его следует выделить и затем нажать клавишу <Delete> или выбрать команду **Remove** контекстного меню. AVR-Studio попросит пользователя еще раз подтвердить операцию удаления.

Для того, чтобы скрыть пункт-регистр, его следует выделить и затем нажать клавишу <H> или выбрать команду **Hide** контекстного меню. Скрытый пункт-регистр отображается по команде **Show** этого меню, в результате выполнения которой AVR-Studio показывает список всех скрытых в данный момент пунктов. Требуемый пункт выбирается и затем отображается нажатием кнопки **OK**.

Пункты-регистры могут перемещаться или копироваться из одного окна **IO** в другое. Для этого должен быть активизирован режим перетаскивания элементов мышью с помощью щелчка на пункте контекстного меню **Item Enable**.

В этом случае каждый пункт-регистр вместе со всеми его подпунктами может быть перемещен в том же окне **IO** с помощью перетаскиванием мышью при нажатой левой кнопке мыши. Если во время отпускания левой кнопки мыши удерживается нажатой клавиша <Ctrl>, то создается копия регистра. Иначе обстоит дело при перетаскивании из одного окна **IO** в другое. Здесь, по умолчанию, регистр копируется, а перемещается только тогда, когда во время отпускания левой кнопки мыши удерживается нажатой клавиша <Ctrl> .

Свойства окна IO

Каждый пункт-регистр в окне **IO** обладает так называемым набором свойств, который необходим для корректного отображения содержимого регистров. Свойства можно просматривать и изменять с помощью контекстного меню.

Свойства состоят из поля имени **Name** и короткого описания пункта **Description**, которые отображаются в виде всплывающего меню, когда указатель мыши расположен над пунктом; адреса ввода/вывода регистр **I/O Address**; значения регистра **Unmasked Value**; флажка **Emphasized**, определяющего будет ли пункт выделен в окне **IO**; и флажка **Display Name Only**, задающего отображение в окне **IO** только имени пункта.

Битовой маской **Mask** в свойствах можно "затемнить" отдельные разряды регистра (например, разряд 7 порта D в AT90S1200 и AT90S2313). Флажок **Use As Normal Mask** задает обычное объединение по логическому "И" маски и регистра, при котором разряды маски, содержащие лог. 0, обеспечивают "затемнение" соответствующих разрядов регистра. Длина регистра, как и прежде, остается равной восьми битам. Флажок **Exclude Masked Bits** задает пропуск разрядов регистра согласно маске, в результате чего его длина уменьшается.

Команда контекстного меню **View Mode** управляет отображением содержимого регистра (**Hexadecimal/Decimal/Binary/Checkbox(es)**), а с помощью команды **Icon Selector** пункту можно назначить какую-либо пиктограмму.

Текущую конфигурацию окна **IO** можно в любой момент сохранить, выбрав в контекстном меню команду **Save View Configuration**. Подобным же образом, сохраненная конфигурация может быть загружена по команде **Load View Configuration**. Кроме того, в момент закрытия проекта AVR-Studio автоматически сохраняет конфигурацию окон **IO** в той же папке, что и сам проект. По умолчанию, файл конфигурации имеет расширение `*.aio`. Если проект позже снова открывается, то окно **IO** устанавливается в сохраненной крнфигурации.

Окно ***Watch***

Окно **Watch** служит для отображения типов, значений и адресов таких объектов как, например, переменные в программе на С. Поскольку AVR-ассемблер, описанный в главе 13, такую информацию не генерирует, то окно **Watch** имеет смысл использовать только при отладке программ, написанных на языке высокого уровня.

Окно **Watch** состоит из четырех колонок. В первой указано имя объекта, за которым ведется наблюдение, во второй — тип объекта (Integer, unsigned Char и т.д.), в третьей — текущее значение, а в четвертой — адреса объектов. Пользователь может добавлять новые объекты в окно **Watch** (команда **Add Watch**), а также удалять их по одному (команда **Delete Watch**) или все сразу (команда **Delete All**).

Окно ***Trace***

Окно **Trace** (рис. 14.9) позволяет пользователю просматривать ход выполнения программы. Его можно открыть с помощью соответствующего пункта меню **View**.

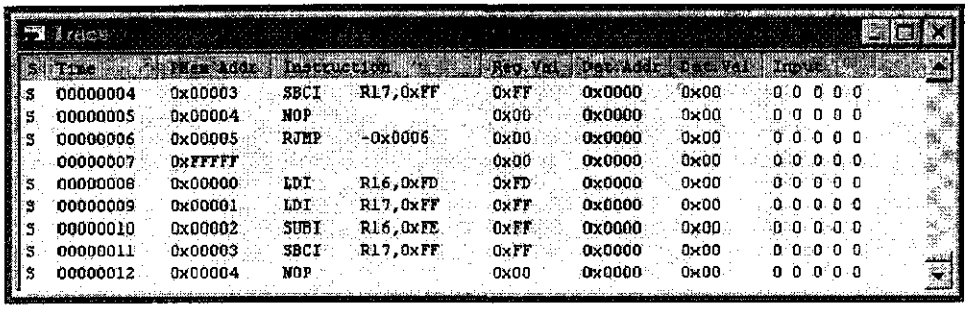


Рис. 14.9. Окно **Trace** AVR-Studio

Окно **Trace** доступно только тогда, когда AVR-Studio работает в режиме симулятора. Самая левая колонка на рис. 14.9 определяет точки синхронизации кода. Если на значке синхронизации дважды щелкнуть мышью, то курсор установится на соответствующей команде в окне исходного кода. Объем буфера трассировки — 32 К x 96 бит.

*Окно **Message***

Окно **Message** выдает пользователю сообщения Studio. После сброса его содержимое очищается.

Меню AVR-Studio

*Меню **File***

С помощью команд меню **File** можно привычным для Windows образом открывать (команда **Open**) и опять закрывать (команда **Close**) файлы.

AVR-Studio воспринимает следующие форматы:

- объектные файлы, сгенерированные AVR- ассемблером (расширение `*.obj`);
- файлы в формате Intel Hex (расширение `*.hex`);
- файлы в формате IAR UBROF (расширение `*.d90`);
- файлы в формате COFF (Common Object File Format) (расширение `*.cof`).

Когда среда AVR-Studio открывает какой-либо файл, она ищет файл с таким же именем, но расширением `*.avd`, который всегда создается в момент закрытия AVR-Studio объектного файла. Он содержит информацию о проекте (например, точки останова и конфигурация окон). Если этот файл отсутствует, то AVR-Studio устанавливает только окно исходного кода.

Команда **Reload** выполняет перезагрузку объектного файла. При этом состояние открытого объектного файла проверяется за несколько секунд. Если обнаружены какие-либо изменения, то об этом появляется сообщение в окне **Message** и пользователю задается вопрос: хочет ли он произвести перегрузку файла для его обновления.

Остальные команды меню **File** аналогичны командам любых других приложений Windows, и потому останавливаться на них в данном случае нет необходимости.

Меню правки Edit

Команды, входящие в меню **Edit**, аналогичны командам любых других приложений Windows, и потому останавливаться на них в данном случае нет необходимости.

*Меню **Debug** для управления ходом программы*

Команда **Reset** (комбинация клавиш <Shift+F5>) выполняет сброс целевой системы. Текущее выполнение программы останавливается. После сброса программа выполняется до первой команды в окне исходного кода. Все окна после сброса обновляются.

Команда **Go** (клавиша <F5>) запускает программу на выполнение. Программа выполняется до тех пор, пока она не будет остановлена пользователем командой **Break** или не встретится точка останова.

Команда **Break** (комбинация клавиш <Ctrl+F5>) останавливает выполнение программы. Все окна после этого обновляются.

Команда **Trace Into** (клавиша <F11>) выполняет команду в окне исходного кода, после чего все окна обновляются. Если речь идет о команде `rcall`, то выполнение программы ответвляется в подпрограмму.

Команда **Step Over** (клавиша <F10>) выполняет команду в окне исходного кода, после чего все окна обновляются. Если речь идет о команде `rcall`, то подпрограмма выполняется так, как будто она — единичная команда. Если встречается точка останова, то выполнение программы останавливается.

Команда **Step Out** (комбинация клавиш <Shift+F11>) запускает программу и выполняет ее до тех пор, пока не встретится окончание текущей функции или подпрограммы. Если ход выполнения находится в области основной программы, то программа будет выполняться до тех пор, пока не будет остановлена пользователем командой **Break** или не встретит точку останова.

Команда **Run To Cursor** (клавиша <F7>) запускает программу, которая выполняется до тех пор, пока не будет достигнута позиция курсора в окне исходного кода. Если встречается точка останова, то выполнение программы не останавливается. Если позиция курсора не достигается никогда, то программа выполняется до тех пор, пока не будет остановлена командой **Break**. После выполнения команды все окна обновляются.

Команда **Multi Trace Into** выполняет предопределенное число раз команду **Trace Iinto**. В подменю **Debug Options** устанавливается, какое число раз выполняется команда и должны ли обновляться окна после каждого выполнения команды **Trace Into** или только после последнего. Если в программе встречается точка останова, то ее выполнение останавливается.

Команда **Auto Trace Into** выполняет повторяющиеся команды **Trace Into**. Длительность паузы после каждой команды **Trace Into** устанавливается в подменю **Debug Options**. После каждого выполнения все окна обновляются. Ко-

манда **Auto Trace Into** выполняется до тех пор, пока пользователь не выполнит команду **Break** или не встретится точка останова.

В опциях отладки **Debug Options** устанавливается число шагов и частота обновления окон для команды **Multi Ttrace Into**, а также длительность паузы для команды **Auto Trace Into**.

Флажок **Break enabled during STEP OVER** разрешает/запрещает останов программы при встрече точки останова во время выполнения команд **Step Over**, **Step Out** и **Run To Cursor**. Флажок **Reload always available** активирует функцию перезагрузки в меню **File** или при изменении объектного файла, или постоянно.

*Меню **Breakpoint***

Команда **Toggle Breakpoint** (клавиша <F9>) меняет состояние точки останова для команды, в которой находится курсор (добавить/удалить).

Команда **Clear All Breakpoints** удаляет все точки останова программы.

Команда **Show List** открывает диалоговое окно **Breakpoint**, в котором пользователь может просмотреть все определенные точки останова, добавить новые (команда **New Breakpoint**), удалить ненужные (команда **Remove**), а также активировать (команда **Enable**) или отключить (команда **Disable**) те или иные точки останова. Активные точки останова помечаются в окне программ черным, а неактивные— пустым квадратиком (см. рис. 14.2).

*Меню **Trace & Triggers***

Трассировка начинается в тот момент, когда по ходу выполнения программы встречается отметка "Trace On" и прекращается по достижению отметки "Trace Off". Отметки "Trace On" и "Trace Off" могут устанавливаться в каждой строке окна исходного кода с помощью команды **Toggle trace & trigger**. С левой стороны выбранной строки появляется отметка "T", и открывается диалоговое окно. Отметка начала трассировки устанавливается с помощью флажка **Trace On**, а отметка о завершении трассировки — с помощью флажка **Trace Off**. Для того чтобы изменить установки, можно дважды щелкнуть мышью на отметке **T**. В результате опять откроется диалоговое окно, в котором можно внести требуемые изменения.

Команда **Toggle Trace & Trigger** (клавиша <F8>) определяет точки трассировки для команды, на которой установлен курсор. Если в этой строке уже установлена точка трассировки, то она удаляется.

Команда **Clear All Trace & Trigger** удаляет все точки трассировки программы.

Команда **Clear Trace Memory** инициализирует буфер трассировки.

Команда **Search In Trace Buffer** открывает диалоговое окно поиска в буфере трассировки. С помощью этого окна можно найти заданные комбинации значений в ранее сохраненном ходе выполнения программы.

*Меню **Watch***

Когда в AVR-Studio тестируется программа на С, для наблюдения за различными объектами можно воспользоваться окном **Watch**.

Команда **Add Watch** добавляет новый объект в список наблюдения. Если при выполнении команды **Add Watch** окно **Watch** закрыто, то оно открывается и одновременно отобрадаются уже определенные контрольные значения.

Команда **Delete Watch** удаляет выделенный объект из списка наблюдения.

Команда **Delete All** удаляет все объекты из списка наблюдения.

Меню ***Options***

Опции эмулятора (команда **Emulator Options**) доступны только тогда, когда к Studio подключен один из AVR-эмуляторов. В этой книге они подробно не рассматриваются.

Опции симулятора (команда **Simulator Options**) доступны тогда, когда AVR-Studio работает в режиме симулятора. Эта команда открывает диалоговое окно, с помощью которого пользователь может выбирать между девятью различными стандартными конфигурациями для установки объема памяти и архитектурных деталей целевого контроллера.

Опции протоколирования симулятора порта (команда **Simulator Port Logging Options**) доступны, когда AVR-Studio работает в режиме симулятора. Эта команда открывает диалоговое окно, с помощью которого пользователь может установить, действия каких портов он хотел бы протоколировать, и в каком файле эти данные должны сохраняться. Каждая строка журнального файла представлена в формате "ЦИКЛ (десятичное значение): БАЙТ_ДАННЫХ (шестнадцатеричное значение)". Если содержимое порта за время тактового цикла не меняется, то в журнал ничего не записывается. Журнальный файл обнуляется после каждого сброса программы.

Опции воздействий симулятора (команда **Simulator Stimuli Options**) доступны, когда AVR-Studio работает в режиме симулятора. Команда открывает диалоговое окно, которое дает пользователю возможность в установленные моменты времени подавать данные на выводы порта. После того как пользователь указал порт, он должен дать имя файлу данных. Каждая строка файла воздействий имеет формат "ЦИКЛ (десятичное значение): БАЙТ_ДАННЫХ (шестнадцатеричное значение)". Опция воздействий влияет только на выводы порта, определенные как входы.

Меню ***Views***

Команда **Toolbars** дает пользователю возможность отображать/скрывать имеющиеся в распоряжении три панели инструментов. Команда **Statusbar** скрывает/отображает строку состояния.

Команды подменю **View**:

- — **Registers** (комбинация клавиш <Alt+0>);
- — **Watch** (комбинация клавиш <Alt+1>);
- — **Messages** (комбинация клавиш <Alt+2>);
- — **Processor** (комбинация клавиш <Alt+3>);
- — **New Memory View** (комбинация клавиш <Alt+4>);
- — **New IO View** (комбинация клавиш <Alt+5>).

Эти команды выполняют те же действия что и соответствующие кнопки панели инструментов: открывают или закрывают соответствующее окно. Исключение составляют команды вызова диалоговых окон **Trace** (см. рис. 14.9) и **Terminal IO**, для которых кнопок на панели инструментов нет.

Окно **Terminal** предоставляет в распоряжение пользователя интерфейс терминала в случае использования Studio для тестирования программы на С, транслированной С-компилятором от IAR, при условии, что компилятор сгенерировал информацию для терминала (по умолчанию), замещающую функции `getchar()` и `putchar()` специальными системными вызовами.

*Меню **Window** и **Help***

Команды, входящие в меню **Window** и **Help**, аналогичны командам любых других приложений Windows, и потому останавливаться на них в данном случае нет необходимости.

Модули ввода/вывода симулятора

Порты моделируются как при работе с реальным контроллером. Имитируется также задержка на 1,5 тактового импульса, как в стандартной логике порта.

Симулятор поддерживает таймер/счетчик 0. Если с помощью меню **Options** был выбран стандартный AVR, то вектор прерываний T/C0 и внешний тактовый вход таймера устанавливаются соответственно выбранному типу микроконтроллера. Если был выбран тип **Custom**, то принимается тип AT90S1200.

Симулятор поддерживает таймер/счетчик 1. Если с помощью меню **Options** был выбран стандартный AVR, содержащий T/C1, то вектор прерываний T/C1, внешний тактовый вход таймера, вывод захвата на входе (Input Capture Pin) и выходные выводы устанавливаются соответственно выбранному типу. Для типов AT90S8515 и AT90S4414 выводу PD4 назначается функция ICP, а выводу PD6 — функция OC1B. Если был выбран тип **Custom**, то вектор прерываний и внешний тактовый вход таймера принимаются как для AT90S1200.

Симулятор поддерживает приёмопередатчик UART. Если с помощью меню **Options** был выбран стандартный AVR, содержащий UART, то вектор прерываний UART и выводы приема/передачи устанавливаются соответственно выбранному типу. Запись данных в регистр данных UDR через симулятор передачу не инициирует, поэтому запись в UDR должна выполняться прикладной программой. Если был выбран тип **Custom**, то UART недоступен.

Симулятор поддерживает последовательный интерфейс SPI. Если с помощью меню **Options** был выбран стандартный AVR, содержащий SPI, то вектор прерываний SPI и выводы SCK/MISO/MOSI/SS устанавливаются соответственно выбранному типу. Регистр данных SPI показывает содержимое регистра приема. Запись данных в регистр данных SPI через симулятор передачу не инициирует даже в режиме "Master", поэтмоу запись в регистр данных SPI должна выполняться прикладной программой. Если был выбран тип **Custom**, то SPI недоступен.

Симулятор поддерживает внешние прерывания. Если с помощью меню **Options** был выбран стандартный AVR, то векторы прерываний устанавливаются со-

ответственно выбранному типу. Если был выбран тип **Custom**, то доступен только вектор прерываний AT90S1200.

Симулятор поддерживает память EEPROM. Для упрощения, длительность доступа на запись задана 16-тактовыми циклами — значение, которое намного меньше, чем в реальном AVR. Если был выбран тип **Custom**, то доступ на запись осуществляется как в модели AT90S1200, без гарантии надежности других стандартных микроконтроллеров AVR.

15 Набор STK200 для тестирования и записи в память собственных программ

Тому, кто хочет избежать лишних затрат на профессиональный эмулятор, однако нуждается в экспериментальной плате для тестирования и программирования разработок с использованием микроконтроллеров семейства AVR, можем порекомендовать набор STK200 от Atmel (рис. 15.1), имеющий очень приемлемую цену.

В поставку STK200 входят следующие компоненты (рис. 15.1).

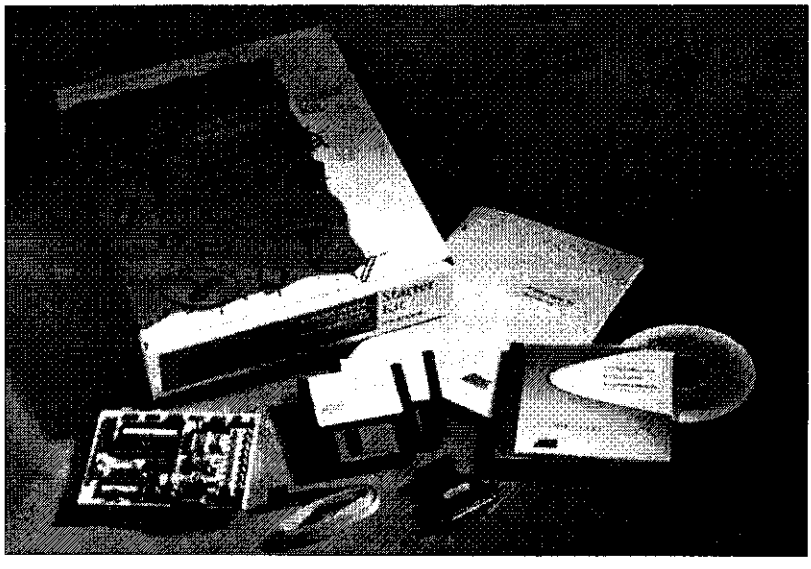


Рис. 15.1. Состав комплекта STK200

- Печатная плата со сквозными металлизированными отверстиями, на которой смонтированы колодки под корпуса DIL для подключения микроконтроллеров AVR базовой серии (DIL20, DIL40), а также для будущих моделей семейства AVR — колодки под корпуса DIL8 и DIL28, и DIL40 — для микроконтроллера с АЦП. Кроме того, плата содержит:
  - интерфейс SPI, с помощью которого микроконтроллер можно легко программировать через параллельный порт ПК с помощью входящего в поставку программного обеспечения ISP (**I**n **S**ystem **P**rogramming — внутрисистемное программирование);
  - кварцевый генератор для формирования тактов системной синхронизации микроконтроллера;
  - интерфейс RS232;
  - выводы для подключения интеллектуального ЖКИ;
  - две колодки, делающие возможным расширение SRAM;

- восемь кнопок, которые могут соединяться через перемычки с портом D;
- восемь светодиодов, которые могут соединяться через перемычки с портом B;
- схема формирования сигнала сброса и обнаружения провалов питания;
- стабилизатор напряжения, который может переключаться между V_{CC} = 5 В и V_{CC} = 3,3 В.

- Защитный ключ-заглушка для подключения к параллельному порту ПК.
- Плоский кабель для подсоединения защитного ключа-заглушки к SPI-интерфейсу STK200.
- Гибкий диск 3,5", содержащий программное обеспечение ISP.
- Компакт-диск с техническими данными и рекомендациями по применению всего спектра продукции компании Atmel.
- Руководство по обслуживанию STK200 (на английском языке).

Набор сконструирован таким образом, что с его помощью могут тестироваться даже будущие модели микроконтроллеров AVR. По этой причине программное обеспечение ISP постоянно обновляется и может быть бесплатно загружено с домашней Web-страницы крмпании Atmel (`www.atmel.com`).

Описание аппаратной части STK200

Монтаж элементов на плате STK200 показан на рис. 15.2.

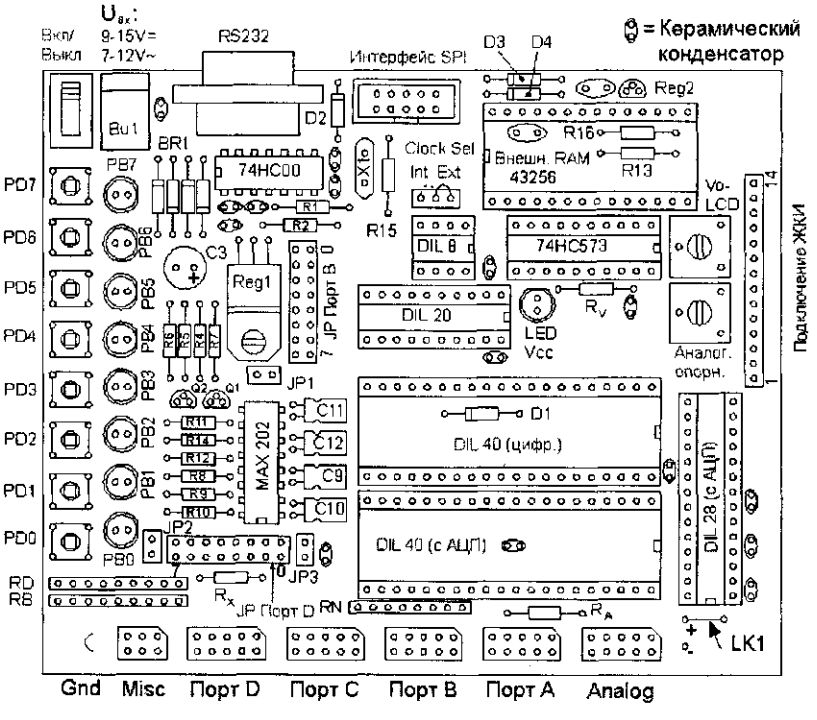


Рис. 15.2. Монтажная сема модуля STK200

Как можно понять из рисунка, на плате расположено несколько перемычек, с помощью которых могут конфигурироваться различные режимы работы и питающие напряжения. Интерфейс ISP подсоединяется к параллельному порту ПК с помощью 10-жильного шлейфа. Для функционирования STK200 в поставку входит защитный ключ-заглушка, через который шлейф подсоединяется к ПК.

Организация питания STK200

В качестве рабочего напряжения модуль STK200 требует или нестабилизированное постоянное напряжение в диапазоне 9...15 В, или, как альтернативу, — переменное напряжение в диапазоне 7...12 В, которое подается на плату через стандартный штекер диаметром 2,1 мм, подключенный в гнездо Bu1. Входное напряжение на плате выпрямляется и стабилизируется, как показывает рис. 15.3.

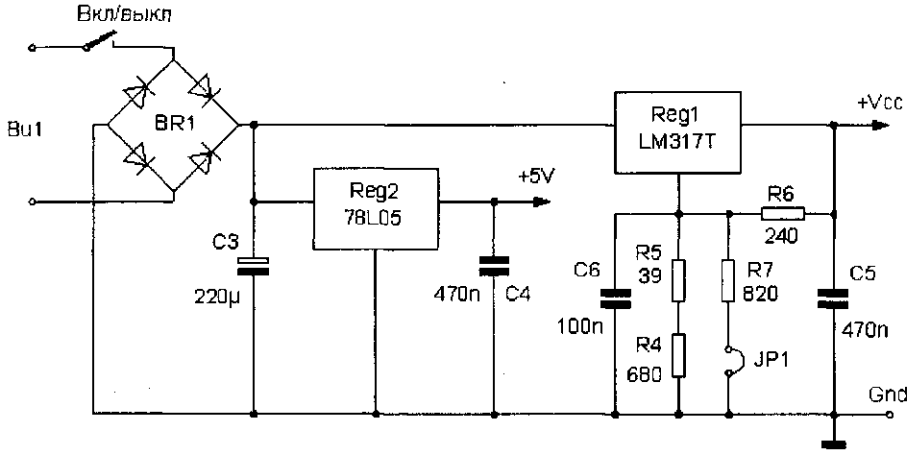


Рис. 15.3. Стабилизация напряжения в STK200

Стабилизатор Reg1 стабилизирует рабочее напряжение V_{CC} платы, которое с помощью перемычки JP1 переключается между значениями 3,3 В и 5 В. Второй стабилизатор Reg2 выдает вспомогательное напряжение +5 В для интерфейса ЖКИ и RS232-драйвера MAX202. К этому напряжению также подключены через свои гасящие резисторы светодиоды LED_{VCC}, отображающие готовность платы к работе.

Если установлена перемычка JP1, то рабочее напряжение V_{CC} = 3,3 В, а при ее отстутствии V_{CC} = 5 В.

Обнаружение провалов напряжения в STK200

Перемычка JP2 устанавливает порог срабатывания при обнаружении провалов напряжения согласно рис. 15.4 (более подробно этот вопрос обсуждается в разделе "Сброс и обработка прерываний" главы 3).

Для рабочего напряжения V_{CC} = 5 В перемычка JP2 должна быть установлена, а для V_{CC} = 3,3 В, соответственно, — отсутствовать. Если V_{CC} = 3,3 В, а установки JP1 и JP2 не соответствуют требуемым, и в то же время порог для провала напря-

жения установлен для 5 В, то схема находится в постоянном сбросе и не может программироваться.

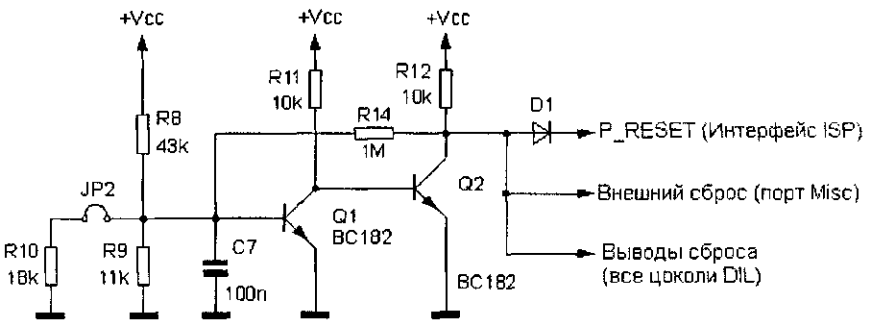


Рис. 15.4. Схема обнаружения провалов напряжения в STK200

Обнаружение провалов напряжения служит для того, чтобы опознавать кратковременные провалы питания и обеспечивать определенное рабочее состояние микроконтроллера через сброс по включению питания (см. главу 3).

Характеристики схемы обнаружения провалов напряжения представлены в табл. 15.1.

Таблица 15.1. Электрическая схема обнаружения провалов напряжения

Номинальное напряжение V_{CC}	*Порог провала*	*Гистерезис*
5 В	4,5 В	0,2 В
3,3 В	2,9 В	0,2 В

Выход схемы обнаружения провалов напряжения подключен к входам RESET всех DIL-колодок на плате и отвечает за обеспечение корректного сброса при включении питания. Благодаря тому, что транзистор Q2 имеет выход с открытым коллектором, через разъем MISC может подаваться внешний сигнал сброса (с активный низким уровнем). Кроме того, сигнал RESET интерфейса ISP подведен через диод D1 к общей точке подключения к Q2, так как программное обеспечение программирования должно быть в состоянии выдавать сигнал сброса на микросхему.

Функции перемычек STK200

В табл. 15.2 перечислены функции перемычек на плате STK200, и их возможные комбинации установки.

Таблица 15.2. Назначение перемычек на плате STK200

Перемычка	*Описание*	*Установлена*	*Отсутствует*	*Начальное состояние*
JP1	Рабочее напряжение	3,3 В	5 В	отсутствует
JP2	Порог обнаружения провала напряжения	4,5 В	2,9 В	установлена
JP3	Вход RS232 RxD	Вкл.	Откл.	отсутствует
JP-BoardB	Подключение светодиодов	Вкл.	Откл.	установлена
JP-BoardD	Подключение кнопок	Вкл.	Откл.	установлена

Таблица 15.2. Окончание

Перемычка	*Описание*	*Установлена*	*Отсутствует*	*Начальное состояние*
Clock Sel *)	Соединяет вывод PB3 цоколя DIL8 с кварц. осциллятором STK200 или с разъемом B	INT = внутр. тактирование (PB3 свободен)	EXT = внешнее тактирование по PB3	EXT
Дорожка LK1	Опорное напряжение для микроконтроллера AVR с ЦАП	Цела: опорн. напр. через потенциом. STK200	Отсоединена: внеш. оп. напр. через разъем Analog	цела

*) Микроконтроллеры AVR с корпусом DIL 8 могут по выбору питаться от внутреннего RC-осциллятора. Для этого перемычка Clock Sel устанавливается в положение INT. В этом случае пользователь получает в распоряжение выход PB3, на который при установке перемычки Clock Sel в положение EXT подавался бы внешний тактовый сигнал.

Подключения портов STK200

Подобно линии сброса, схемы портов для расположенных на плате DIL-панелей также разведены и соединены с разъемами A...D внизу платы STK200 (см. рис. 15.2).

Следует учитывать, что на плате в каждый момент времени может быть размещен только один микроконтроллер в одной из DIL-панелей. Если будет включено более одного микроконтроллера AVR, то это может привести к их ненормальному функционированию и повреждению из-за возникших лишних связей по печатным проводникам.

На всех микроконтроллерах семейства AVR для однозначного определения вывода 1 нанесено изображение треугольника. Кроме того, на их торце есть специальный вырез.

При включении микроконтроллера на плату STK200 следует убедиться, что вырез на микросхеме и насечка на цоколе размещены друг над другом. Никогда не включайте и не снимайте микросхемы при включенном питании!

40-контактная колодка для микроконтроллеров AVR с АЦП имеет другое расположение выводов в сравнении с размещенной рядом 40-контактной колодкой для исключительно цифровых микросхем, к которым относятся все представители базовой серии. По этой причине микроконтроллеры базовой серии AVR никогда не следует включать в колодку для микроконтроллеров с АЦП.

Назначение контактов в разъемах, предназначенных для подключения внешних схем, показано на рис. 15.5.

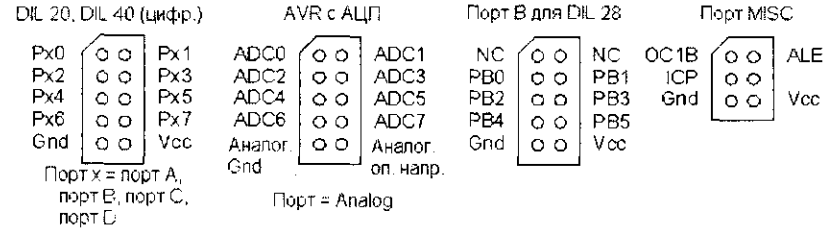


Рис. 15.5. Назначение контактов в разъемах платы STK200 (вид сверху)

Базовой серии семейства AVR, рассматриваемой в этой книге, соответствуют разъемы, показанные на рис. 15.5 слева. Порты B и D 20- и 40-контактных колодок подведены с помощью печатных проводников к одноименным разъемам. Последовательно с линиями PB7, PB6 и PB5 для защиты интерфейса ISP включены сопротивления из массива резисторов по 10 кОм каждый (см. рис. 15.6), поэтому эти выводы следует использовать как входы или выходы с нагрузкой менее 500 мкА при V_{CC} = 5 В и, соответственно, — 300 мкА при V_{CC} = 3,3 В.

Порты A и C присутствуют только для 40-контактных DIL-колодок и также соединены печатными проводниками с обозначенными соответствующим образом разъемами. Наряду с линиями ввода/вывода, каждый разъем подразумевает наличие цепей Gnd и V_{CC} для обеспечения питанием подключенной схемы. Поскольку для стабилизатора напряжения Reg1 на плате STK200 не предусмотрен радиатор охлаждения, то потребление тока внешней схемой обязательно должно составлять мене 100 мА. Если это ограничение не может быть выдержано, то внешняя схема должна питаться от отдельного источника питания.

Через разъем MISC к схеме может быть подведен внешний сигнал сброса (открытый коллектор, низкий активный уровень). Кроме того, эти разъемы дополняются сигналами OC1B, ALE и ICP микроконтроллеров AT90S8515 и AT90S4414.

Интерфейс ISP STK200

Схема интерфейса ISP (In System Programming) показана на рис. 15.6.

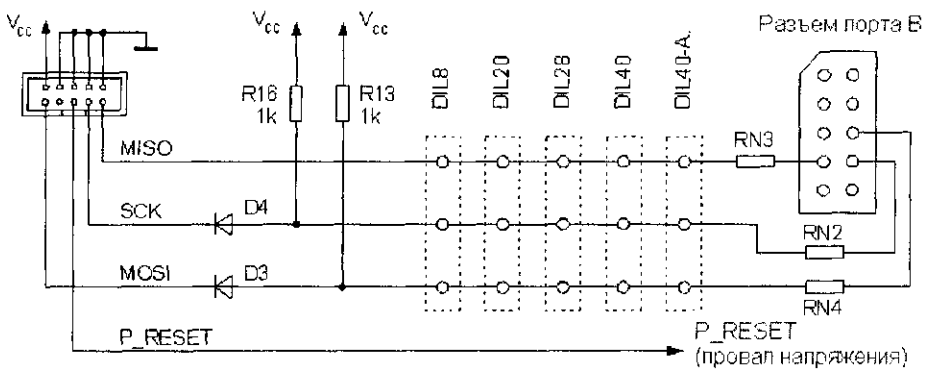


Рис. 15.6. Интерфейс ISP STK200

Для того чтобы защитить подключенный ПК и ключ защиты, линии SCK, MOSI и P_RESET (см. рис. 15.4) снабжены диодами и подтягивающими резисторами. Линии для последовательного программирования микроконтроллера AVR соединены с выводами SPI всех DIL-колодок на плате.

Защитные резисторы RN2...RN4 находятся в массиве резисторов RN (4 x 10 кОм), который изображен на рис. 15.2 под разъемом DIL 40 для микроконтроллера AVR с АЦП.

RS232-интерфейс STK200

Схема RS232-интерфейса показана рис. 15.7.

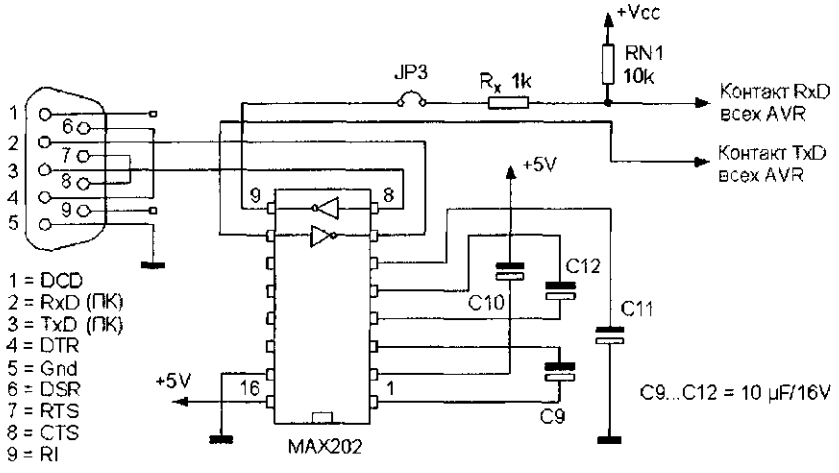


Рис. 15.7. Интерфейс RS232 платы STK200

MAX202 — это улучшенный вариант известной микросхемы MAX232, содержащий схему накопления заряда для формирования вспомогательных напряжений +10 В и –10 В для RS232-интерфейса, а также магистральный усилитель-формирователь для преобразования уровня TTL в уровень RS232 и наоборот.

Для передачи данных необходим девятижильный кабель с разъемом Sub D, а линии TxD и RxD не должны быть перекрестными (не используйте нуль-модемный кабель!).

Если интерфейс RS232 не используется, линия RxD может быть разорвана для защиты микросхемы. Для этого необходимо извлечь перемычку JP3. Подтягивающий резистор RN1 расположен в массиве резисторов RN (4 x 10 кОм), который на рис. 15.2 изображен под разъемом DIL 40 для микроконтроллера AVR с АЦП.

Поскольку сигналы RxD и TxD интерфейса RS232 подаются на выводы PD0 и PD1 микроконтроллера AVR, рекомендуется в случае использования интерфейса изолировать кнопки PD0 и PD1. Для этого следует извлечь соответствующие перемычки в группе JP-PortD.

Линии обмена с квитированием, предусмотренные интерфейсом RS232, в STK200 не применяются. Если скорость передачи определяется на основании тактовой частоты 4 МГц кварцевого осциллятора, встроенного на плату, то погрешность находится в диапазоне 0,2...2,1% (см. табл. 6.2 в главе 6). Если эти значения неприемлемы, то для формирования тактов системной синхронизации должен применяться другой кварц.

ЖКИ-интерфейс STK200

Плата STK200 может применяться для подключения стандартного жидкокристаллического индикатора с контроллером Hitachi HD44780. Для этого используют 14-контактный разъем, расположенный на рис. 15.2 в правом верхнем углу. Подстроечным резистором, который на рис. 15.2 обозначен как "Vo-LCD", можно изменять напряжение V_o в пределах 0...+5 В для установки контрастности инди-

катора. Более ранние ЖКИ зачастую требовали отрицательного напряжения V_0 0...–5 В, которое отсутствует в STK200. В этом случае цепь V_0 на STK200 должна быть разорвана, а V_0 — подаваться извне на вывод 3 ЖКИ.

Управление индикатором может быть реализовано в режиме отображения в памяти или в режиме ввода/вывода. В режиме отображения в памяти к индикатору происходит обращение как к внешнему ОЗУ по адресам согласно табл. 15.3.

Таблица 15.3. Адреса доступа к ЖКИ-модулю в режиме отображения в памяти

Функция	/WR	/RD	Адрес (двоичный)	Адрес (шестн.)
Запись команды	0	1	10xx xxxx xxxx xxxx	8000
Чтение адреса и флага занятости	1	0	10xx xxxx xxxx xxxx	8000
Запись данных	0	1	11xx xxxx xxxx xxxx	C000
Чтение данных	1	0	11xx xxxx xxxx xxxx	C000

Размерность элементов на рис. 15.8 дана для обеспечения работы с тактовой частотой 4 МГц. Для другой частоты функционирование в режиме отображения в памяти не может быть гарантировано.

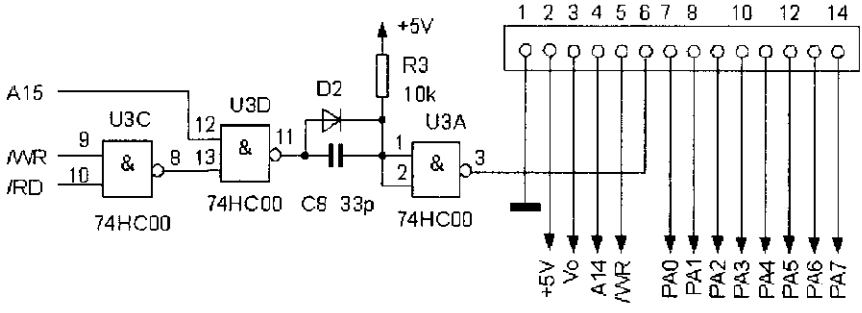


Рис. 15.8. ЖКИ-интерфейс STK200

Адресная линия A15 активирует ЖКИ-модуль, когда выдается адрес, больший или равный $8000. Таким образом адреса для внешнего ОЗУ ограничиваются диапазоном $0000–$7FFF.

Выбор регистра ЖКИ для записи значения определяется сигналом A14 на выводе RS (Register Select, контакт 4) модуля. Если A14 имеет низкий логический уровень, то значение записывается в регистр команд, в противном случае — в регистр данных. При доступе на чтение считывается счетчик адреса и флаг занятости (A14 = 0) или же регистр данных (A14 = 1).

Режим чтения или записи определяется логическим уровнем на входе R/W модуля (контакт 5). Лог. 0 означает доступ на запись, а лог. 1 — на чтение. На основании этого получают адреса доступа к ЖКИ-модулю в режиме отображения в памяти, перечисленные в табл. 15.3.

Собственный строб-импульс, по которому происходит обмен данными между микроконтроллером AVR и контроллером HD44780 по выводам PA0...PA7, формируется одновибратором, построенным на элементах U3A, C8 и R3.

Если в режиме отображения в памяти нет необходимости, то в режиме ввода/вывода разряды различных портов могут устанавливаться и сбрасываться обычным образом.

Встроенный кварцевый осциллятор STK200

Для генерирования тактов ситемной синхронизации на плате STK200 присутствует кварцевый осциллятор, генерирующий импульсы с частотой 4 МГц, построенный на логическом элементе "И-НЕ" U3B (рис. 15.9). Сгенерированные импульсы подаются через резистор R15 на вход XTAL1 всех микроконтроллеров AVR на плате. Выход XTAL2 осциллятора, интегрированного в микроконтроллер, остается свободным.

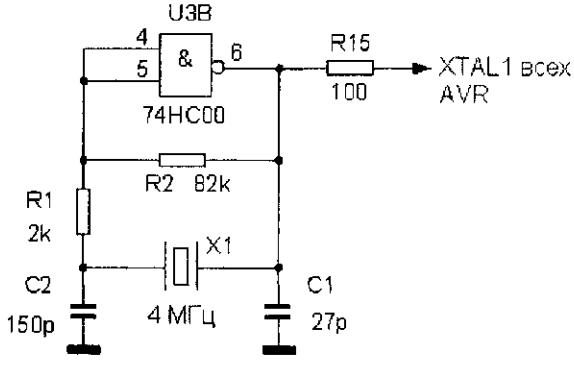


Рис. 15.9. Встроенный кварцевый осциллятор STK200

Восемь индикаторных светодиодов на STK200

На плате STK200 расположены восемь светодиодов, которые пользователь в целях тестирования может подсоединить с помощью набора перемычек JP-PortB к порту B своего микроконтроллера AVR. Если эти светодиоды используются, то порт B с помощью своего регистра направления передачи данных DDRB должен быть сконфигурирован как выход. Восемь токоограничивающих резисторов по 1 кОм каждый расположены в массиве RB, общий вывод которого соединен с V_{CC} (рис. 15.10).

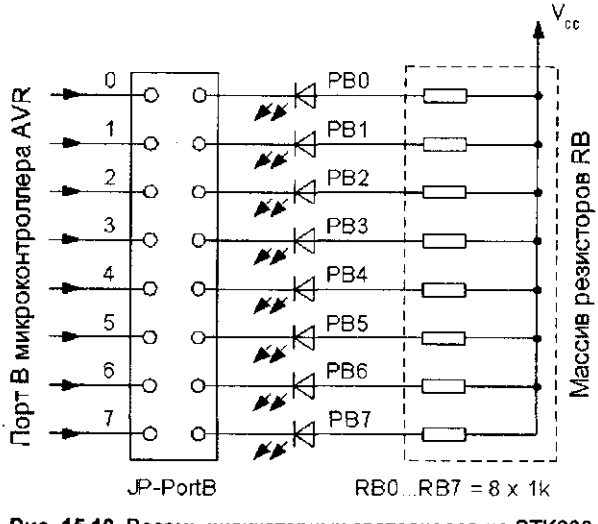


Рис. 15.10. Восемь индикаторных светодиодов на STK200

Восемь кнопочных выключателей на STK200

На плате STK200 находится восемь кнопочных выключателей, которые пользователь в целях тестирования может подсоединить с помощью набора перемычек JP-PortD к порту D своего микроконтроллера AVR. Если эти выключатели используются, то порт D с помощью своего регистра направления передачи данных DDRD должен быть сконфигурирован как вход. Восемь рабочих сопротивлений по 10 кОм каждое расположены в массиве резисторов RD, общий вывод которого соединен с V_{CC} (рис. 15.11).

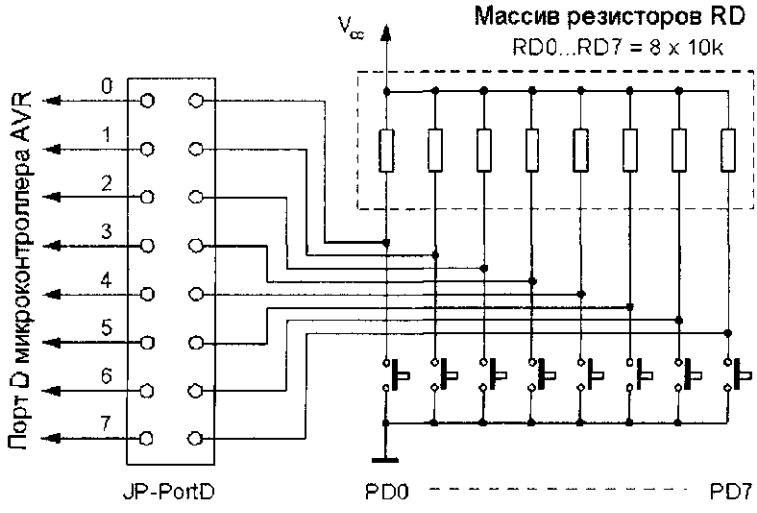


Рис. 15.11. Восемь кнопочных выключателей на STK200

Расширение памяти в STK200

На плате STK200 есть разъемы для расширения памяти (см. рис. 3.6 в главе 3). В качестве элемента памяти используется микросхема типа 43256 и 62256 объемом 32 K x 8 бит.

Время доступа микросхемы должно лежать в пределах 70 нс для частоты системной синхронизации 4 МГц STK200. Если применяется более медленная память, то необходимо добавлять состояния ожидания (см подраздел "Внешняя память SRAM" главы 3).

Программное обеспечение для STK200

Программное обеспечение ISP для STK200 базируется на проектном подходе, что присуще всем современным системам разработки высокого уровня. Говоря обобщенно, это означает, что программа управляет совокупностью разнотипной информации. В случае микроконтроллеров AVR, для программирования доступна память программ, память EEPROM, а также разряды предохранения и блокировки. Все эти данные, принадлежащие определенному проекту, среда проектирования сводит для пользователя воедино.

Перед процессом программирования должен или быть открыт уже существующий проект, или создан новый. Для этого могут применяться команды меню **Project** среды ISP или одна из кнопок: **New Project** или **Open Project** (см. рис. 15.13). Как только проект открыт, могут загружаться файлы данных с помощью команды меню **File → Load** или пиктограммы **Open File**.

Программа самостоятельно распознает формат файла `.hex`: Intel-Hex, Motorola S или Atmel Generic (см. главу 13) — и загрузит его в окно, открытое в данный момент. Выбор активного окна (памяти программ или EEPROM) осуществляется обычным для Windows способом: с помощью щелчка мышью или команды меню **Windows**.

Загруженные данные отображаются синим шрифтом, пустое окно (без данных) содержит $FF в черном шрифте во всех ячейках памяти.

Меню программного обеспечения ISP

Функции, входящие в меню **File**, **Windows** и **Help**, типичны для окружения Windows, и их названия говорят сами о себе, поэтому останавливаться на них мы не будем. Характерным для ISP является меню **Project**, с помощью команд которого можно запустить, открыть, сохранить и закрыть проект, хотя, по сути, оно подобно меню **File** других приложений Windows.

Меню **Buffer** позволяет пользователю редактировать содержимое окна, переходить к определенному адресу в пределах буфера, искать байт данных или формировать контрольную сумму содержимого буфера.

С помощью меню **Program** программируются данные в памяти микроконтроллера AVR. Наиболее простой путь "прожига" данных — функция **Auto-Program**, которую можно активировать или через меню **Program** (клавиша <F5>), или же через пиктограмму **Autoprogram**.

При выборе команды **Autoprogram** выполняются все функции, предварительно заданные в меню **Autoprogram Options** (рис. 15.12). С помощью флажков пользователь может выбирать выполняемые действия.

В том случае, если программное обеспечение ISP не находит защитного ключа-заглушки на параллельном порте, на экране появится сообщение об ошибке "Dongle not found". Если это сообщение появляется, хотя ключ и подсоединен, то следует проверить корректность выбора параллельного порта.

После проверки защитного ключа-заглушки программное обеспечение пытается считать идентификационные байты сигнатуры микроконтроллера AVR. В случае неудачной попытки также будет выдано соответствующее сообщение.

Само собой разумеется, отдельные шаги программирования можно выполнять вручную. Для этого используют команды меню **Program**:

- **Device Empty Check** — проверяет, все ли ячейки памяти микроконтроллера AVR содержат значение "$FF";
- **Erase** — стирает всю память микроконтроллера AVR (содержимое = $FF);
- **Program Device** — программирует память программ микроконтроллера AVR;

- **Program EEPROM** — программирует память EEPROM микроконтроллера AVR;
- **Verify Device** — сверяет каждый участок в буфере с соответствующим участком в памяти программ микроконтроллера AVR;
- **Verify EEPROM** — сверяет каждый участок в буфере с соответствующим участком в памяти EEPROM микроконтроллера AVR;
- **Read Device** — считывает все ячейки флэш-памяти микроконтроллера AVR, и помещает результат в буфер по соответствующим адресам;
- **Read EEPROM** — считывает все ячейки памяти EEPROM, и помещает результат в буфер по соответствующим адресам;
- **Program Lockbits** — программирует разряды блокировки согласно установкам среды проектирования;
- **Health Check** — тестирует микроконтроллер AVR на способность функционировать;
- **Device Checksum** — вычисляет контрольную сумму для содержимого микроконтроллера AVR.

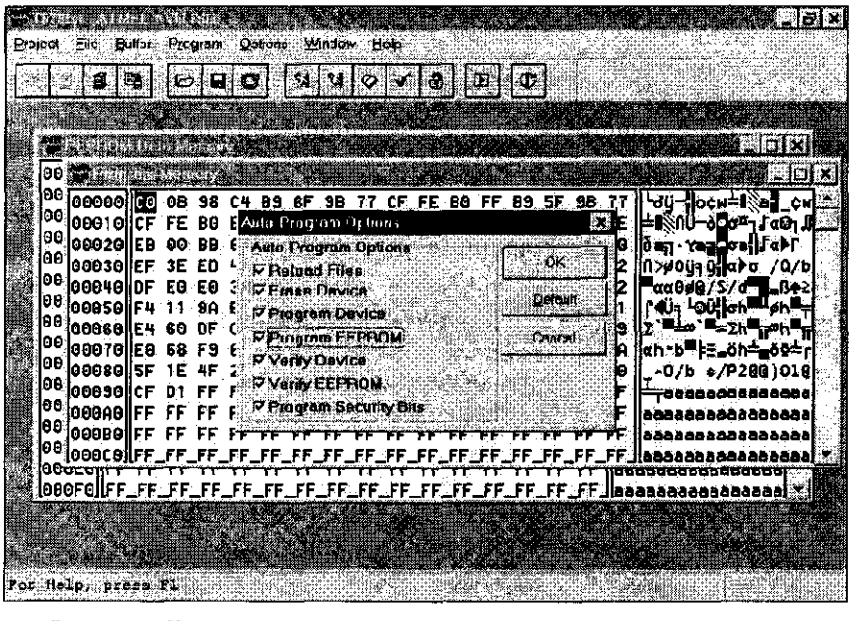


Рис. 15.12. Типичное представление экрана программого обеспечения ISP с активным меню **Auto-Program Options**

Успешно запрограммированные данные отображаются в соответствующем окне зеленым цветом, а данные, запрограммированные с ошибками, будут выделены красным цветом.

Меню **Options** позволяет настроить такие параметры как использование параллельного порта, шрифт, отключение теста сигнатуры (**Advanced**), опции SPI и цветовое оформление.

Панель инструментов среды ISP

Наиболее важные команды меню могут также выполняться с помощью пиктограмм (кнопок) панели инструментов (рис. 15.13).

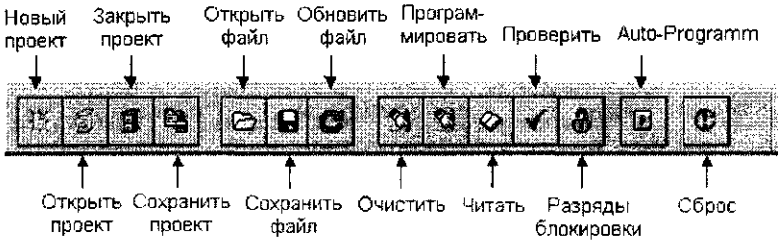


Рис. 15.13. Панель инструментов среды ISP

При программировании с помощью кнопки панели инструментов программируется память команд или EEPROM — в зависимости от того, какое окно активно в данный момент. При чтении содержимого микроконтроллера AVR с помощью кнопки панели инструментов считывается содержимое памяти программ или EEPROM — в зависимости от того, какое окно активно в данный момент.

16 Применение микроконтроллеров семейства AVR

В этой главе будут рассмотрены примеры применения основных функций, реализованных в базовой серии микроконтроллеров AVR. Наряду с использованием таймера (включая режимы сравнения и захвата), портов, интерфейса SPI, аналогового компаратора и "спящего" режима, будет затронут доступ к памяти EEPROM. При этом, благодаря множеству конкретных примеров программ, даже новички смогут быстро освоить методы применения микроконтроллеров AVR на практике.

Для модели AT90S1200, которая не может "похвастаться" такой же богатой палитрой аппаратных компонентов как другие представители семейства AVR, будет рассмотрена исключительно программная реализация полнодуплексного приемопередатчика UART, позволяющего передавать данные со скоростью 9600 бод, интерфейсов SPI и I^2C, автоперезагрузки таймера, а также функции широтно-импульсного модулятора на базе таймера/счетчика T/C0.

Средства двоично-десятичной арифметики

В том случае, если требуется часто выводить из программы на дисплей шестнадцатеричные значения как десятичные или же непосредственно производить вычисления с двоично-десятичными числами в обход шестнадцатеричной системы счисления, можно или постоянно выполнять преобразования, или же (что предпочтительнее) воспользоваться какой-нибудь небольшой библиотекой подпрограмм, содержащей средства двоично-десятичной (BCD — **B**inary **C**oded **D**ecimal) арифметики.

Предлагаем следующий набор подпрограмм, который может пригодиться читателю в его собственных проектах.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\161BCD.asm`

```
;**** Средства BCD-арифметики ***************************************
;*
;* Подпрограммы BCD-арифметики.
;* HexBCD: преобразовывает шестнадцатеричное слово (целочисленное,16 разрядов)
;* в 5-разрядное BCD-число
;* XBruchD: преобразовывает простую шестнадцатеричную дробь (4 знака после
;*          запятой) в BCD-дробь
;* BCDHex: преобразовывает BCD-число (5 целочисленных разрядов) в
;*         шестнадцатеричное слово (16 разрядов)
;* BCDAdd: BCD-сложение двухразрядных чисел
;* BCDSub: BCD-вычитание двухразрядных чисел
;*
;**********************************************************************

.nolist
```

```
.include "8515def.inc"
.list

;***** Регистровые переменные
.def    CpyL = r11      ; Младший байт промежуточного буфера
.def    CpyH = r12      ; Старший байт промежуточного буфера
.def    BCD0 = r13      ; BCD-разряды 2 и 1
.def    BCD1 = r14      ; BCD-разряды 4 и 3
.def    BCD2 = r15      ; BCD-разряды 6 и 5
.def    hexL = r16      ; Младший байт шестнадцатеричного слова
.def    hexH = r17      ; Старший байт шестнадцатеричного слова
.def    cnt = r18       ; Счетчик
.def    tmp1 = r19      ; Рабочий регистр
.def    tmp2 = r20      ; Рабочий регистр
.def    Digt = r21      ; Рабочий регистр
.def    BCDa = r22      ; 1. BCD-число
.def    BCDb = r23      ; 2. BCD-число

RESET:
000000 c073 rjmp Haupt ; Переход к главной программе

;***************************************************************************
;* HexBCD - 16-разрядное двоичное число (целое) -> BCD.
;* Эта подпрограмма преобразовывает 16-разрядное шестнадцатеричное число,
;* хранимое в hexH:hexL, в 5-разрядное упакованное BCD-число, сохраняемое
;  в BCD2:BCD1:BCD0.
;***************************************************************************
HexBCD:                         ; xxxx (Hex) -> 0abcde (BCD)
000001 d044     rcall Div10     ; hexH:hexL / 10, остаток - в tmp1
000002 2ed3     mov BCD0,tmp1   ; Результат - в hexH:hexL, tmp1 = разряд 1
000003 d042     rcall Div10     ; Остаток = разряд 2
000004 9532     swap tmp1       ; Разряд 2 - в старший полубайт
000005 2ad3     or BCD0,tmp1    ; Объединяем разряды 2 и 1 в BCD0
000006 d03f     rcall Div10     ; Остаток = разряд 3
000007 2ee3     mov BCD1,tmp1   ; В BCD1
000008 d03d     rcall Div10     ; Остаток = разряд 4
000009 9532     swap tmp1       ; Разряд 4 - в старший полубайт
00000a 2ae3     or BCD1,tmp1    ; Объединяем разряды 4 и 3 в BCD1
00000b 2ef0     mov BCD2,hexL   ; Разряд 5 - в BCD2, старший полубайт = 0
00000c 9508 ret

;***************************************************************************
;* XBruchD - 16-разрядная двоичная дробь -> BCD-дробь.
;* Эта подпрограмма преобразовывает 16-разрядную двоичную дробь "0,abcd"
;* (4 шестнадцатеричных знака после запятой), хранимую в регистровой паре
;* hexH:hexL, в восьмиразрядную упакованную BCD-1hj,m "0,654321", сохраняемую
;* в регистрах BCD2:BCD1:BCD0.
;***************************************************************************
XBruchD:                        ; 0,abcd (HEX) -> 0,654321 (BCD)
00000d d04e     rcall Mul10     ; Умножаем hexH:hexL на 10
00000e 9532     swap tmp1       ; Переполнение в tmp1 = разряд 6
00000f 2ef3     mov BCD2,tmp1   ; Разряд 6 - в BCD2
000010 d04b     rcall Mul10     ; Переполнение в tmp1 = разряд 5
000011 2af3     or BCD2,tmp1    ; Объединяем разряды 6 и 5 в BCD2
000012 d049     rcall Mul10     ; Переполнение в tmp1 = разряд 4
000013 9532     swap tmp1       ; Разряд 4 - в старший полубайт
```

```
000014 2ee3    mov BCD1,tmp1  ; Разряд 4 - в BCD1
000015 d046    rcall Mul10    ; Переполнение в tmp1 = разряд 3
000016 2ae3    or BCD1,tmp1   ; Объединяем разряды 4 и 3 в BCD1
000017 d044    rcall Mul10    ; Переполнение в tmp1 = разряд 2
000018 9532    swap tmp1      ; Разряд 2 - в старший полубайт
000019 2ed3    mov BCD0,tmp1  ; Разряд 2 - в BCD0
00001a d041    rcall Mul10    ; Переполнение в tmp1 = разряд 1
00001b 2ad3    or BCD0,tmp1   ; Объединяем разряды 2 и 1 в BCD0
00001c 9508 ret

;***************************************************************************
;* BCDHex - 5-разрядное целое BCD-число -> шестнадцатеричное слово (16 бит).
;* Эта подпрограмма преобразовывает 5-разрядное BCD-число < 65536, хранимое в
;* регистрах BCD2:BCD1:BCD0, в 4-хзначковое шестнадцатеричное число, хранимое
;* в регистровой паре hexH:hexL.
;***************************************************************************
BCDHex:                        ; 0abcde (BCD) -> xxxx (Hex)
00001d 2711    clr hexH
00001e 2d0f    mov hexL,BCD2  ; hexH:hexL = a
00001f 2d5e    mov Digt,BCD1
000020 d04c    rcall MulAddH  ; hexH:hexL = 10a+b
000021 2d5e    mov Digt,BCD1
000022 d04b    rcall MulAddL  ; hexH:hexL = (10a+b)10+c
000023 2d5d    mov Digt,BCD0
000024 d048    rcall MulAddH  ; hexH:hexL = ((10a+b)10+c)10+d
000025 2d5d    mov Digt,BCD0
000026 d047    rcall MulAddL  ; hexH:hexL = (((10a+b)10+c)10+d)10+e
000027 9508    ret

;***************************************************************************
;* BCDAdd - сложение двух BCD-чисел.
;* Складываются два упакованных двухзначковых BCD-числа BCDa и BCDb.
;* Результат сохраняется в регистровой паре BCDb:BCDa.
;***************************************************************************
BCDAdd:                        ; BCDa + BCDb
000028 0f67    add BCDa,BCDb  ; Двоичное сложение чисел
000029 2777    clr BCDb       ; Обнуляем BCDb (старший байт)
00002a f408    brcc BA1       ; Переход, если флаг переноса C=0
00002b 9573    inc BCDb       ; Записываем BCD-переполнение
BA1:
00002c f025    brhs BA2       ; Переход, если флаг половинного переноса H=1
00002d 5f6a    subi BCDa,-6   ; Проверка: прибавляем 6 к младшей цифре
00002e f41d    brhc BA3       ; Если H=0, то младшая цифра > 9
00002f 5066    subi BCDa,6    ; Выполняем обратную проверку
000030 c001    rjmp BA3
BA2:                           ; Требуется BCD-коррекция:
000031 5f6a    subi BCDa,-6   ; Прибавляем к младшей цифре 6
BA3:
000032 5a60    subi BCDa,-$60 ; Проверка: прибавляем 6 к старшей цифре
000033 f418    brcc BA5       ; Если C=0, то старшая цифра > 9
000034 ff70    sbrs BCDb,0    ; Если при двоичном сложении нет переполнения,
000035 5660    subi BCDa,$60  ;   выполняем обратную проверку
000036 9508    ret
BA5:
000037 9573    inc BCDb       ; BCD-переполнение
```

```
000038 9508    ret

;***************************************************************************
;* BCDSub - вычитание для двух BCD-чисел
;* Упакованное двухзнаковое BCD-число BCDb вычитается из такого же числа BCDa.
;* Результат сохраняется в BCDa.
;***************************************************************************
BCDSub:                         ; BCDa - BCDb
000039 1b67    sub BCDa,BCDb    ; Двоичное вычитание
00003a 2777    clr BCDb
00003b f408    brcc BS0         ; Если флаг переноса C=1 (BCDa < BCDb), то
00003c 9573    inc BCDb         ;   записываем в BCDb заем
BS0:
00003d f40d    brhc BS1         ; Переход, если флаг половинного переноса H=0
00003e 5066    subi BCDa,$06    ; BCD-коррекция: вычитаем 6 из младшей цифры
BS1:
00003f ff70    sbrs BCDb,0      ; Пропускаем следующую команду, если есть заем
000040 9508    ret              ; Выход из подпрограммы
000041 5660    subi BCDa,$60    ; BCD-коррекция: вычитаем 6 из старшей цифры
000042 e071    ldi BCDb,1       ; Проверка: установка заема в BCDb
000043 f408    brcc SB2         ; Если не возникает заема, то
000044 e071    ldi BCDb,1       ;   выполняем обратную проверку
SB2:
000045 9508    ret

;***************************************************************************
;* Вспомогательная программа
;***************************************************************************
Div10:                          ; Деление hexH:hexL на 10,
                                ; результат - в hexH:hexL, остаток - в tmp1
000046 2733    clr tmp1
000047 0f00    lsl hexL
000048 1f11    rol hexH
000049 1f33    rol tmp1         ; Деление старших трех разрядов на 10
00004a 0f00    lsl hexL         ;   может дать в результате
00004b 1f11    rol hexH         ;   только 0
00004c 1f33    rol tmp1
00004d 0f00    lsl hexL
00004e 1f11    rol hexH
00004f 1f33    rol tmp1
000050 e02d    ldi cnt,13       ; Счетчик для оставшихся 13 разрядов
Loop10:
000051 0f00    lsl hexL         ; Сдвиг делимого влево на 1 разряд
000052 1f11    rol hexH
000053 1f33    rol tmp1         ; Переполнение в tmp1
000054 503a    subi tmp1,10     ; Проверка: переполнение > 10 ?
000055 f010    brlo HB1         ; Переход, если нет
000056 9503    inc hexL         ; Увеличиваем счетчик деления на 1
000057 c001    rjmp HB2
HB1:
000058 5f36    subi tmp1,-10    ; Выполняем обратную проверку
HB2:
000059 952a    dec cnt          ; Декрементируем счетчик разрядов
00005a f7b1    brne Loop10      ; Переход, если еще не все разряды
00005b 9508    ret
```

```
Mul10:                          ; Умножение "hexH:hexL" (abcd) на 10 ...
00005c 2733    clr tmp1         ; ... Переполнение - в tmp1
00005d 0f00    lsl hexL         ; Умножаем на 2
00005e 1f11    rol hexH
00005f 1f33    rol tmp1         ; Переполнение
000060 2eb0    mov cpyL,hexL    ; Сохраняем в буфер
000061 2ec1    mov cpyH,hexH
000062 2f43    mov tmp2,tmp1
000063 0cbb    lsl cpyL         ; Умножаем на 2 (всего - на 4)
000064 1ccc    rol cpyH
000065 1f33    rol tmp1         ; Переполнение
000066 0cbb    lsl cpyL         ; Умножаем на 2 (всего - на 8)
000067 1ccc    rol cpyH
000068 1f33    rol tmp1         ; Переполнение
000069 0d0b    add hexL,cpyL    ; Прибавляем промежуточный результат ( x 2)
00006a 1d1c    adc hexH,cpyH
00006b 1f34    adc tmp1,tmp2    ; Переполнение 0...9 - в tmp1
00006c 9508    ret

MulAddH:                        ; hexH:hexL*10 + старший полубайт (Digt)
00006d 9552    swap Digt        ; Меняем местами старший и младший полубайты
MulAddL:                        ; hexH:hexL*10 + младший полубайт (Digt)
00006e dfed    rcall Mul10      ; Умножаем на 10
00006f 705f    andi Digt,0x0f ; Обнуляем старший полубайт
000070 0f05    add hexL,Digt    ; Прибавляем младший полубайт
000071 f408    brcc MA1         ; Переход, если переполнение
000072 9513    inc hexH         ; Инкрементируем старший байт
MA1:
000073 9508    ret
Haupt:
000074 e032    ldi tmp1,High(RamEnd)
000075 bf3e    out sph,tmp1
000076 e53f    ldi tmp1,Low(RamEnd)
000077 bf3d    out spl,tmp1
000078 ef1e    ldi hexH,$FE
000079 ed0c    ldi hexL,$DC
H1:
00007a df92    rcall XbruchD    ; r17:r16, $0,FEDC = 0,99554
00007b ef1e    ldi hexH,$FE
00007c ed0c    ldi hexL,$DC
00007d df83    rcall HexBCD     ; r17:r16, $FEDC = 65244
00007e df9e    rcall BCDHex     ; BCD2:BCD1:BCD0, 65244 = $FEDC
00007f e968    ldi BCDa,$98
000080 e877    ldi BCDb,$87
000081 dfa6    rcall BCDadd     ; BCDb:BCDa = 01:85
000082 e966    ldi BCDa,$96
000083 e575    ldi BCDb,$55
000084 dfb4    rcall BCDsub     ; BCDb:BCDa = 00:41
000085 e565    ldi BCDa,$55
000086 e976    ldi BCDb,$96
000087 dfb1    rcall BCDsub     ; BCDb:BCDa = 01:59 (-41)
000088 cff1    rjmp H1
000089 0000    nop
```

Преобразование шестнадцатеричного числа в BCD-число

Подпрограмма `HexBCD` выполняет преобразовывание 16-разрядного шестнадцатеричного числа, хранимого в регистровой паре hexH:hexL, в пятиразрядное упакованное BCD-число, сохраняемое в регистрах BCD2:BCD1:BCD0.

Преобразование заключается в нескольких операциях делениях с остатком. Число в шестнадцатеричной системе счисления может быть переведено в двоично-десятичный эквивалент посредством его последовательного деления на основание новой системы счисления (то есть, на $0A). Частное от текущего деления является делимым для следующей операции деления, а соответствующие остатки формируют отдельные цифры BCD-числа: b_N, ..., b_1, b_0. Преобразование завершается, если результат деления равен нулю.

Пример: $1B_h$: $A_h = 2$ Остаток $7 \Rightarrow b_0 = 7$

2_h : $Ah = 0$ Остаток $2 \Rightarrow b_1 = 2$

$1B_h = 27_d$

Преобразование простой двоичной дроби в BCD-дробь

Подпрограмма `XBruchD` выполняет преобразование простой 16-разрядной двоичной дроби "0,wxyz" (4 шестнадцатеричных знака после запятой), хранимой в регистровой паре hexH:hexL, в шестиразрядную упакованную BCD-дробь "0,654321", сохраняемую в регистрах BCD2:BCD2:BCD0.

Преобразование выполняется согласно принципу последовательного умножения. Простая дробь в шестнадцатеричной системе счисления может быть переведена в двоично-десятичный эквивалент посредством последовательного умножения на основание новой системы счисления (то есть, на $0A). Соответствующие переполнения, полученные в результате операций умножения, формируют отдельные цифры BCD-дроби: b_{-1}, b_{-2}, b_{-3}, ... Оставшийся остаток произведения является множимым для следующей операции умножения. Преобразование завершается, если достигнуто желаемое количество знаков после запятой в BCD-дроби.

Пример: $0{,}1B_h \times A_h = 1{,}0E$ Переполнение $1 \Rightarrow b_{-1} = 1$

$0{,}0E_h \times A_h = 0{,}8C$ Переполнение $0 \Rightarrow b_{-2} = 0$

$0{,}8C_h \times A_h = 5{,}78$ Переполнение $5 \Rightarrow b_{-3} = 5$

$0{,}78_h \times A_h = 4{,}B0$ Переполнение $4 \Rightarrow b_{-4} = 4$

...

$0{,}1B_h = 0{,}1054_d$

Преобразование BCD-числа в шестнадцатеричное число

Подпрограмма `BCDHex` выполняет преобразование пятиразрядного BCD-числа $n \leq 65535$, хранимого в регистрах BCD2:BCD1:BCD0, в четырехзначковое шестнадцатеричное число, сохраняемое в регистровой паре hexH:hexL.

Для того чтобы определить соответствующее число, каждая цифра BCD-числа умножается на вес ее разряда в новой системе счисления, и все произведения складываются:

$$n = (b_N\ b_{N-1}\ b_{N-2} \ldots b_1\ b_0)_d = (b_N \times \$0A^N + b_{N-1} \times \$0A^{N-1} + \ldots + b_1 \times \$0A^1 + b_0)_h.$$

Пример: $1234_d = 1 \times \$3E8 + 2 \times \$64 + 3 \times \$0A + 4 = \$4D2$.

Вынося \$0A за скобки, получаем:

$$n = (b_N \times \$0A + b_{N-1}) \times \$0A + b_{N-2}) \times \$0A \ldots + b_1) \times \$0A + b_0)_h.$$

Таким образом, в подпрограмме `BCDHex` к произведению старшей цифры, умноженной на \$0A, прибавляется следующая младшая цифра, затем эта сумма опять умножается на \$0A, и к ней прибавляется следующая младшая цифра и т.д.

Сложение BCD-чисел

В подпрограмме `BCDAdd` наглядно видна разница между архитектурами CISC и RISC. Несмотря на то, что система команд микроконтроллеров AVR не уступает по объему системе команд любого микроконтроллера CISC того же класса, в ней, однако, отсутствует команда `da`, используемая в большинстве CISC-микроконтроллеров для десятичной коррекции после сложения двух BCD-чисел. Так, в микроконтроллере 8051 операция BCD-сложения выполняется с помощью двух команд: `ADD A, Rr` и `DA A`. Размер обеих команд составляет один байт, и на их выполнение в сумме затрачивается два машинных такта, то есть, — 24 тактовых цикла, что при частоте осциллятора 12 МГц соответствует времени 2 мс.

Подпрограмма `BCDAdd`, реализованная в системе команд микроконтроллеров AVR, состоит из 17 команд. При частоте системной синхронизации 8 МГц на выполнение этих 17 команд потребуется в сумме всего лишь 1,375–1,75 мс. Команда `DA A` — это крайний случай, поскольку, эффективность большинства других команд распространенных CISC-микроконтроллеров еще меньше эффективности соответствующих им команд, присутствующих в системе AVR.

Аппаратная реализация BCD-коррекции одной цифры показана на рис. 16.1. Рассмотренная подпрограмма `BCDAdd` является аналогом этой схемы.

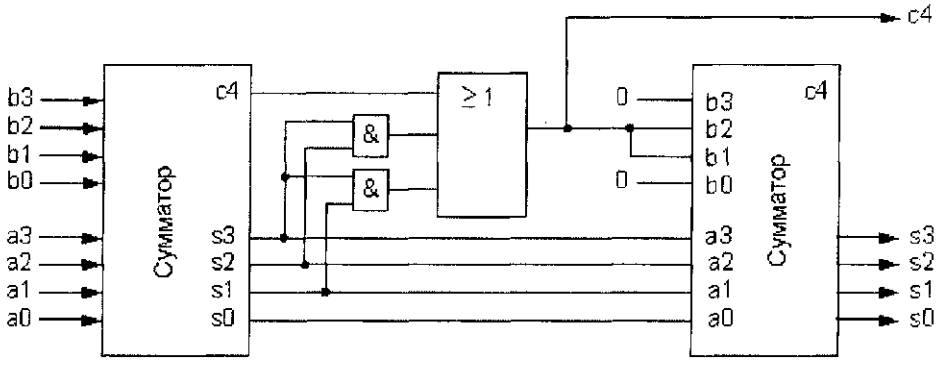


Рис. 16.1. Аппаратная реализация десятичной коррекции после BCD-сложения

Вычитание BCD-чисел

Подпрограмма `BCDSub` выполняет действие, обратное десятичному сложению. Она вычитает BCDb из BCDa и выполняет соответствующую коррекцию результата, сохраненного в BCDa. Если результат отрицательный, то выполняется дополнение на основе 10, и в результате установки младшего разряда в регистре BCDb возникает "заем" из ближайшей старшей позиции (см. пример в главной программе: 55 – 96 = 01:59, что соответствует –49). Благодаря этому, возможно BCD-вычитание, выполняемое над несколькими байтами (155 – 96 = 59).

На прилагаемом к книге компакт-диске в папке `\Atmel\Software` есть еще несколько интересных арифметических подпрограмм:

- `avr200.exe` — умножение и деление многобайтовых значений;
- `avr202.asm` — 16-разрядная арифметика;
- `avr102.asm` — работа с указателями (копирование блоков данных).

Все они могут быть бесплатно загружены с Web-сайта компании Atmel.

Базовые операции ввода/вывода

В этом разделе рассматриваются некоторые подпрограммы ввода/вывода для набора STK200, с помощью которых новичок сможет одновременно познакомиться как с STK200, так и с портами микроконтроллеров семейства AVR.

Классическим "первым примером" при программировании новых микроконтроллеров является включение лампочки. Не будем отступать от этой традиции и рассмотрим следующий пример: при нажатии определенной кнопки модуля STK200 на нем должны загораться светодиодные индикаторы (всего восемь) согласно некоторому шаблону, создавая своеобразное "световое шоу".

Как опрос кнопки, так и выдача различных шаблонов индикации регулируется с помощью прерывания T/C0, которое циклически возникает каждые 20 мс. После входа в подпрограмму обработки прерывания флаги состояния главной программы сбрасываются, а таймер перезагружается.

После этого выполняется опрос кнопок, подключенных к порту D. Алгоритм опроса кнопок показан на рис. 16.2.

При каждом прерывании считывается состояние восьми кнопок, подключенных к порту D. Флаг "Нажата" извещает о том, была ли нажата в момент последнего прерывания какая-либо кнопка. Если этот флаг установлен, то текущее считанное состояние кнопок сопоставляется с тем, которое было сохранено при предыдущем прерывании. Если состояние не изменилось, то оценивается значение флага "Повтор". Если этот флаг установлен, то распознается нажатие той же кнопки, что и при предыдущем прерывании. В том случае, когда флаг "Повтор" сброшен, кнопка распознается нажатой в первый раз, происходит переход в режим индикации, и соответствующая часть подпрограммы указывает на смену режима посредством установки флага "Новый". Дополнительно устанавливается флаг "Повтор", чтобы при следующем прерывании сигнализировать о том, что данное состояние кнопок уже было прочитано ранее.

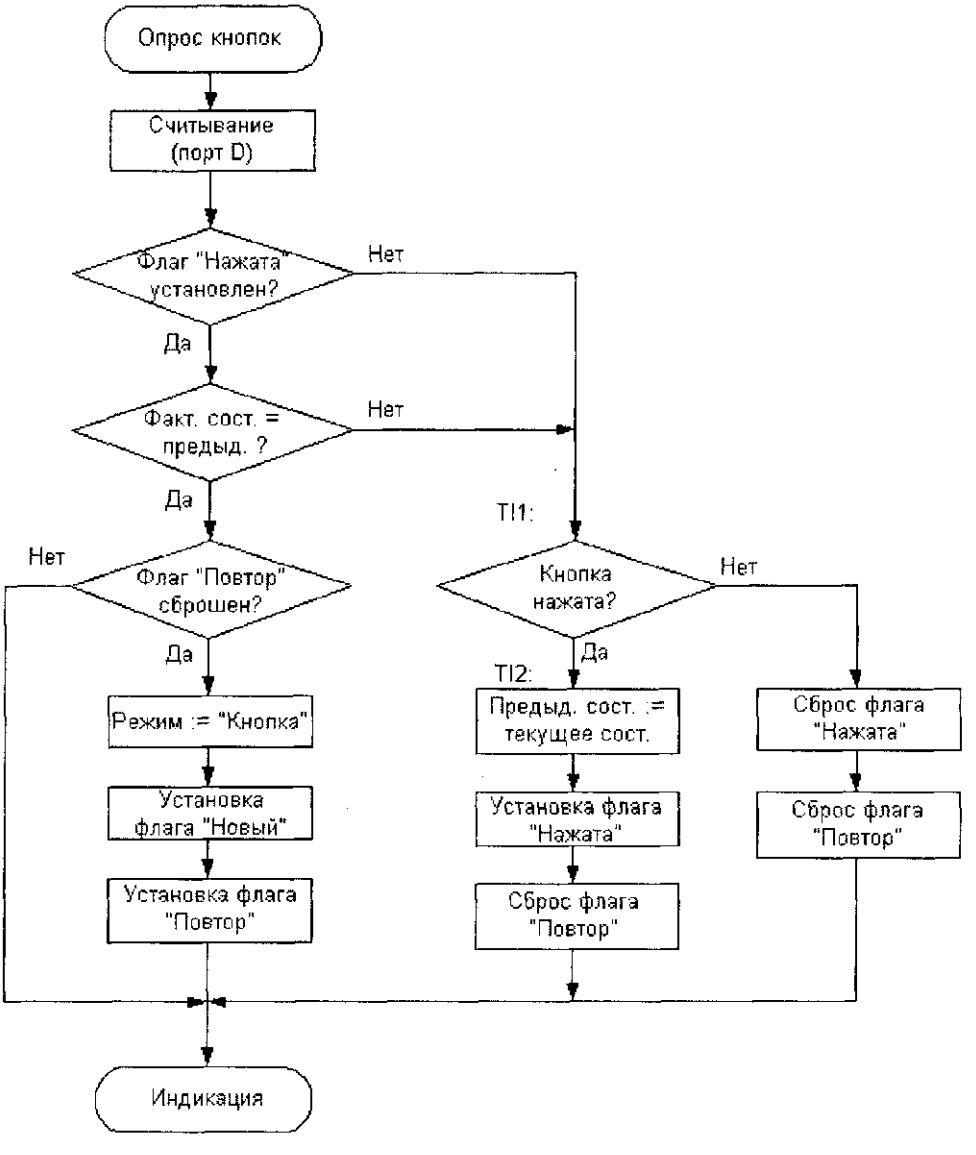


Рис. 16.2. Опрос состояния кнопочного выключателя модуля STK200 и его оценка

Если флагу "Нажата" соответствует уровень лог. 0, то это указывает на то, что при последнем прерывании ни одна кнопка нажата не была, и происходит переход к точке алгоритма TI1, где проверяется, нажата ли на этот раз какая-либо кнопка. Если какая-либо кнопка нажата, то на соответствующем выводе устанавливается уровень лог. 0 (другими словами, состояние $FF указывает на то, что ни одна кнопка нажата не была). Если было определено состояние $FF, то после сброса флагов "Нажата" и "Повтор" опрос можно завершить.

Состояние, отличное от $FF, указывает на то, что была нажата как минимум одна кнопка. В этом случае текущее считанное состояние сохраняется для того, чтобы быть распознанным при последующем прерывании. Также в случае обнаружения нажатия какой-либо кнопки устанавливается флаг "Нажата", а флаг "Повтор" сбрасывается, поскольку данная кнопка была нажата впервые.

Часть подпрограммы обработки прерывания, обозначенная как "Индикация", выполняется только каждым двенадцатым прерыванием, чтобы частота индикации составляла 1/240 мс = 4,17 Гц. В начале этого программного фрагмента выполняется выбор режима индикации и определяется соответствующая часть программы, к которой следует перейти. Для каждого отдельного режима в результате обработки флага "Новый" определяется, имела ли место смена режима (то есть, является ли текущий режим новым). В случае смены режима, инициализируется шаблон индикации, и флаг "Новый" сбрасывается. Если режим не изменился, то новый шаблон индикации формируется на основании старого. Затем полученный шаблон отображается с помощью светодиодных индикаторов, и после восстановления флагов состояния подпрограмма обработки прерывания завершается.

После поступления сигнала сброса выполнение программы начинается с адреса $000 (фрагмент, обозначенный как `Initial`). Здесь прежде всего инициализируется указатель стека, а затем — порты B и D микроконтроллера AVR. По умолчанию, при запуске программы выбран режим 7, поэтому в порт B загружается значение $F0, а посредством записи в регистр направления передачи данных DDRB значения $FF этот порт определяется как выход. Значение $F0 в порту B порождает низкий уровень на выводах PB0...PB3 и тем самым включает соответствующие светоиндикаторы.

Регистр DDRD порта D инициализируется автоматически значением $00 при поступлении сигнала сброса по включению питания и, таким образом, порт D определен как вход.

После сброса всех флагов подпрограммы обработки прерывания T/C0 и установки выбранного по умолчанию режима 7, инициализируется счетчик прерываний `ICnt`. Его назначение — обеспечить обновление шаблона индикации только через каждые 12 прерываний.

Когда поступает сигнал сброса, в регистр TCCR0 автоматически загружается значение $00, и тем самым останавливается таймер/счетчик T/C0. Для разрешения прерывания от T/C0 устанавливается разряд 2 (TOIE0) в регистре TIMSK.

Далее в счетный регистр TCNT0 загружается отрицательное значение длительности интервала. Этот интервал, представленный константой `Time`, должен быть отрицательным, поскольку T/C0 работает как суммирующий счетчик.

На следующем шаге с помощью мультиплексора такт системной синхронизации делится на 1024 и тем самым выбирается входной такт для T/C0. Кроме того, в регистр TCCR0 записывается значение $05 (см. табл. 4.2), и T/C0 начинает счет.

В завершение части инициализации для того, чтобы разрешить все прерывания, устанавливается флаг I в регистре состояния SREG.

В главной части программы, которая начинается с метки `Wait`, реализовано только ожидание прерываний от T/C0. Для того чтобы в этот период микроконтроллер ничего не делал, он переводится в ждущий режим (разряд SM в регистре

MCUCR = 0) посредством установки разряда SE в регистре MCUCR и последующего вызова команды `sleep`. Из "спящего" режима микроконтроллер выходит по прерыванию от T/C0.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\162inout.asm`

```
;**** Подпрограммы ввода/вывода для AVR ********************************
;*
;* Подпрограммы ввода/вывода для модуля STK200, демонстрирующие работу
;* с портами микроконтроллеров семейства AVR. При нажатии некоторой кнопки на
;* STK200 загораются светодиоды в соответствии со следующим шаблоном:
;* Кнопка PD0: Светящаяся "точка" перемещается слева направо
;* Кнопка PD1: Светящаяся "точка" перемещается справа налево
;* Кнопка PD2: Светящаяся "точка" перемещается туда-сюда
;* Кнопка PD3: Светящаяся "полоска" перемещается слева направо
;* Кнопка PD4: Светящаяся "полоска" перемещается справа налево
;* Кнопка PD5: Две светящиеся "точки" перемещаются слева направо
;* Кнопка PD6: Две светящиеся "точки" перемещаются справа налево
;* Кнопка PD7: Попеременно мерцают значения старшего и младшего полубайта
;*             (режим, выбранный по умолчанию)
;*
;* Тактовая частота микроконтроллеров AVR: 4 МГц (стандарт STK200).
;*
;************************************************************************
.nolist
.include "8515def.inc"
.list

.def    Last = r14      ; Предыдущее состояние кнопок
.def    Stat = r15      ; Буфер для временного хранения содержимого SREG
.def    tmpi = r16      ; Рабочий регистр прерывания от таймера
.def    ICnt = r17      ; Счетчик прерываний
.def    Flg = r18       ; Флаги программы
.def    tmph = r19      ; Рабочий регистр главной программы
.def    Mode = r20      ; Режим индикации
.equ    Dwn = 0         ; Флаг 0 ("Нажата") = была нажата некоторая кнопка
.equ    Rpt = 1         ; Флаг 1 ("Повтор") = кнопка опять нажата
.equ    New = 2         ; Флаг 2 ("Новый") = смена режима индикации
.equ    Dir = 3         ; Флаг 3 = направление (0=влево, 1=вправо)
.equ    Time = 80000 / 1024

.cseg

000000 c086    rjmp Initial   ; После сброса - к главной программе
000001 9518    reti           ; Внешнее прерывание 0 не используется
000002 9518    reti           ; Внешнее прерывание 1 не используется
000003 9518    reti           ; Прерывание T/C1 Capture не используется
000004 9518    reti           ; Прерывание T/C1 Comp. A не используется
000005 9518    reti           ; Прерывание T/C1 Comp. B не используется
000006 9518    reti           ; Прерывание по переполнению T/C1 не используется

Timer0:                       ; Прерывание по переполнению T0 (каждые 20 мс)
000007 b6ff    in Stat,SREG   ; Сохраняем флаги главной программы
000008 eb02    ldi tmpi,-Time
000009 bf02    out TCNT0,tmpi ; Перезагружаем T/C0
KeyScan:
```

```
00000a b300    in tmpi,PinD   ; Считываем состояние кнопок
00000b ff20    sbrs Flg,Dwn   ; Пропускаем следующую команду, если при предыду-
                              ; щем прерывании была нажата какая-то кнопка
00000c c007    rjmp TI1       ; Переход, если ни одна кнопка не нажата
00000d 12e0    cpse Last,tmpi ; Пропускаем следующую команду, если все еще
                              ; нажата та же кнопка
00000e c005    rjmp TI1       ; Переход, если не совпадает состояние кнопок
00000f fd21    sbrc Flg,Rpt   ; Пропускаем следующую команду, если флаг
                              ; "Повтор" сброшен
000010 c00a    rjmp Display   ; Переход, если флаг "Повтор" установлен
000011 2f40    mov Mode,tmpi  ; Установка режима
000012 6026    sbr Flg,1<<New | 1<<Rpt  ; Установка флагов "Новый" и "Повтор"
000013 c007    rjmp Display
TI1:
000014 3f0f    cpi tmpi,$FF   ; Кнопка нажата?
000015 f411    brne TI2       ; Переход, если какая-то кнопка нажата
000016 7f2c    cbr Flg,1<<Dwn | 1<<Rpt  ; Сброс флагов "Нажата" и "Повтор"
000017 c003    rjmp Display
TI2:
000018 2ee0    mov Last,tmpi          ; Сохранение состояния кнопок
000019 7f2d    cbr Flg,1<<Rpt         ; Сброс флага "Повтор"
00001a 6021    sbr Flg,1<<Dwn         ; Установка флага "Нажата"

Display:
00001b 951a    dec Icnt               ; Счетчик - 1
00001c f009    breq D1                ; Переход, если равно нулю
00001d c067    rjmp IntEnd            ; Завершение обработки прерывания
D1:
00001e e01c    ldi ICnt,12            ; Новая инициализация счетчика
00001f b308    in tmpi,PortB          ; Считывание состояния светоиндикаторов
000020 ff40    sbrs Mode,0            ; Пропускаем следующую команду,если Бит0=1
000021 c05a    rjmp Mode0
000022 ff41    sbrs Mode,1            ; Пропускаем следующую команду,если Бит1=1
000023 c04e    rjmp Mode1
000024 ff42    sbrs Mode,2            ; Пропускаем следующую команду,если Бит2=1
000025 c033    rjmp Mode2
000026 ff43    sbrs Mode,3            ; Пропускаем следующую команду,если Бит3=1
000027 c028    rjmp Mode3
000028 ff44    sbrs Mode,4            ; Пропускаем следующую команду,если Бит4=1
000029 c01d    rjmp Mode4
00002a ff45    sbrs Mode,5            ; Пропускаем следующую команду,если Бит5=1
00002b c012    rjmp Mode5
00002c ff46    sbrs Mode,6            ; Пропускаем следующую команду,если Бит6=1
00002d c007    rjmp Mode6
Mode7:
00002e ff22    sbrs Flg,New           ; Если флаг "Новый"=1, то - новый режим
00002f c002    rjmp M7
000030 ef00    ldi tmpi,$F0           ; Инициализация шаблона индикации
000031 7f2b    cbr Flg,1<<New         ; Сброс флага "Новый"
M7:
000032 9500    com tmpi               ; Дополнение до 1
000033 bb08    out PortB,tmpi         ; Включаем светодиоды
000034 c050    rjmp IntEnd            ; Готово
Mode6:
000035 ff22    sbrs Flg,New           ; Если флаг "Новый"=1, то - новый режим
000036 c002    rjmp M6
```

```
000037 e303    ldi tmpi,$33          ; Инициализация шаблона индикации
000038 7f2b    cbr Flg,1<<New        ; Сброс флага "Новый"
M6:
000039 0f00    lsl tmpi              ; Сдвиг влево
00003a f408    brcc M6A              ; Переход, если не "столкновение"
00003b 6001    ori tmpi,$01
M6A:
00003c bb08    out PortB,tmpi        ; Включаем светодиоды
00003d c047    rjmp IntEnd           ; Готово
Mode5:
00003e ff22    sbrs Flg,New          ; Если флаг "Новый"=1, то - новый режим
00003f c002    rjmp M5
000040 ec0c    ldi tmpi,$CC          ; Инициализация шаблона индикации
000041 7f2b    cbr Flg,1<<New        ; Сброс флага "Новый"
M5:
000042 9506    lsr tmpi              ; Сдвиг вправо
000043 f408    brcc M5A              ; Переход, если не "столкновение"
000044 6800    ori tmpi,$80
M5A:
000045 bb08    out PortB,tmpi        ; Включаем светодиоды
000046 c03e    rjmp IntEnd           ; Готово
Mode4:
000047 ff22    sbrs Flg,New          ; Если флаг "Новый"=1, то - новый режим
000048 c002    rjmp M4
000049 ef0f    ser tmpi              ; Инициализация шаблона индикации
00004a 7f2b    cbr Flg,1<<New        ; Сброс флага "Новый"
M4:
00004b 0f00    lsl tmpi              ; Сдвиг влево
00004c f008    brcs M4A              ; Переход, если не "столкновение"
00004d 6001    ori tmpi,$01
M4A:
00004e bb08    out PortB,tmpi        ; Включаем светодиоды
00004f c035    rjmp IntEnd           ; Готово
Mode3:
000050 ff22    sbrs Flg,New          ; Если флаг "Новый"=1, то - новый режим
000051 c002    rjmp M3
000052 ef0f    ser tmpi              ; Инициализация шаблона индикации
000053 7f2b    cbr Flg,1<<New        ; Сброс флага "Новый"
M3:
000054 9506    lsr tmpi              ; Сдвиг вправо
000055 f008    brcs M3A              ; Переход, если не "столкновение"
000056 6800    ori tmpi,$80
M3A:
000057 bb08    out PortB,tmpi        ; Включаем светодиоды
000058 c02c    rjmp IntEnd           ; Готово
Mode2:
000059 ff22    sbrs Flg,New          ; Если флаг "Новый"=1, то - новый режим
00005a c002    rjmp M2
00005b ef07    ldi tmpi,$F7          ; Инициализация шаблона индикации
00005c 7f2b    cbr Flg,1<<New        ; Сброс флага "Новый"
M2:
00005d 9408    sec                   ; Устанавливаем флаг переноса
00005e 9507    ror tmpi              ; Сдвиг вправо
00005f fd23    sbrc Flg,Dir          ; Если флаг Dir=0, то сдвиг влево
000060 c00a    rjmp M2B
000061 1f00    rol tmpi              ; Меняем направление на сдвиг влево
```

```
000062 9408    sec                     ; Устанавливаем флаг переноса
000063 1f00    rol tmpi                ; Сдвиг влево
000064 f020    brcs M2A                ; Переход, если не "столкновение"
000065 6028    sbr Flg,1<<Dir          ; Установка флага Dir: сдвиг вправо
000066 9507    ror tmpi                ; Меняем направление на сдвиг вправо
000067 9408    sec                     ; Устанавливаем флаг переноса
000068 9507    ror tmpi                ; Сдвиг вправо
M2A:
000069 bb08    out PortB,tmpi          ; Включаем светодиоды
00006a c01a    rjmp IntEnd             ; Готово
M2B:
00006b f020    brcs M2C                ; Переход, если не "столкновение"
00006c 7f27    cbr Flg,1<<Dir          ; Сброс флага Dir: сдвиг влево
00006d 1f00    rol tmpi                ; Меняем направление на сдвиг влево
00006e 9408    sec                     ; Устанавливаем флаг переноса
00006f 1f00    rol tmpi                ; Сдвиг влево
M2C:
000070 bb08    out PortB,tmpi          ; Включаем светодиоды
000071 c013    rjmp IntEnd             ; Готово
Mode1:
000072 ff22    sbrs Flg,New            ; Если флаг "Новый"=1, то - новый режим
000073 c002    rjmp M1
000074 ef07    ldi tmpi,$F7            ; Инициализация шаблона индикации
000075 7f2b    cbr Flg,1<<New          ; Сброс флага "Новый"
M1:
000076 9408    sec                     ; Устанавливаем флаг переноса
000077 1f00    rol tmpi                ; Сдвиг влево
000078 f008    brcs M1A                ; Переход, если флаг переноса C=1
000079 ef0e    ldi tmpi,$FE            ; Начинаем сначала
M1A:
00007a bb08    out PortB,tmpi          ; Включаем светодиоды
00007b c009    rjmp IntEnd             ; Завершение обработки прерывания
Mode0:
00007c ff22    sbrs Flg,New            ; Если флаг "Новый"=1, то - новый режим
00007d c002    rjmp M0
00007e e70f    ldi tmpi,$7F            ; Инициализация шаблона индикации
00007f 7f2b    cbr Flg,1<<New          ; Сброс флага "Новый"
M0:
000080 9408    sec                     ; Устанавливаем флаг переноса
000081 9507    ror tmpi                ; Сдвиг вправо
000082 f008    brcs M0A                ; Переход, если флаг переноса C=1
000083 e70f    ldi tmpi,$7F            ; Начинаем сначала
M0A:
000084 bb08    out PortB,tmpi          ; Включаем светодиоды
IntEnd:
000085 beff    out SREG,Stat           ; Восстанавливаем регистр состояния
000086 9518    reti

Initial:
000087 e032    ldi tmph,High(RamEnd)
000088 bf3e    out sph,tmph
000089 e53f    ldi tmph,Low(RamEnd)
00008a bf3d    out spl,tmph            ; Инициализация стека
00008b ef30    ldi tmph,$F0            ; Включаем светодиоды 0..3
00008c bb38    out PortB,tmph
00008d ef3f    ldi tmph,$FF
```

```
00008e bb37    out DDRB,tmph          ; Все разряды порта B - выход
00008f 2733    clr tmph               ; Эти команды - не обязательны, поскольку
000090 bb31    out DDRD,tmph          ;    порт D является входом по умолчанию
000091 2722    clr Flg                ; Сбрасываем все флаги
000092 e74f    ldi Mode,$7F           ; По умолчанию выбран шаблон индикации 7
000093 e01c    ldi ICnt,12            ; Инициализация счетчика прерываний
000094 e032    ldi tmph,$02           ; С помощью разряда TOIE0 разрешаем
000095 bf39    out TIMSK,tmph         ;    прерывание по переполнению T/C0
000096 eb32    ldi tmph,-Time         ; Передаем начальное значение для
000097 bf32    out TCNT0,tmph         ;    таймера 0
000098 e035    ldi tmph,$05           ; Устанавливаем коэффициент деления
000099 bf33    out TCCR0,tmph         ;    и тем самым запускаем T/C0
00009a 9478    sei                    ; Общее разрешение прерываний
Wait:
00009b e230    ldi tmph,$20           ; Выбираем ждущий режим
00009c bf35    out MCUCR,tmph         ;   и разрешаем переход в "спящий" режим
00009d 9588    sleep                  ; Ожидаем прерывания от T/C0
00009e 0000    nop
00009f cffb    rjmp Wait
```

Подключение ЖК-модулей

Несмотря на настойчивые попытки различных производителей контроллеров жидкокристаллических (ЖК) модулей перехватить пальму первенства, несменным лидером на рынке дешевых ЖК-модулей по-прежнему остается контроллер HD44780 от компании Hitachi. Это является достаточным основанием для того, чтобы избрать его в качестве примера для изучения процесса управления ЖК-модулями с помощью микроконтроллеров AVR. На прилагаемом к книге компакт-диске в файле `\daten\hd44780.pdf` находится техническое описание последнего варианта этого контроллера: HD44780U.

Из-за недостатка места мы не сможем здесь рассмотреть все технические характеристики, параметры и временные диаграммы HD44780, однако функции выводов в разъеме подключения (рис. 16.3), а также основные свойства и команды этого модуля все же рассмотрим.

Вывод	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Функ.	Gnd	V_{CC}	V_O	RS	R/W	E	D0	D1	D2	D3	D4	D5	D6	D7

Рис. 16.3. Стандартное расположение выводов типичного ЖК-модуля

Подключение модуля HD44780

Питающее напряжение составляет +5 В ± 10%. На вывод V_O для регулировки контрастности табло в большинстве ЖК-модулей должно подаваться напряжение 0...+5 В. Это напряжение легко получить с помощью подстроечного резистора (обычно сопротивлением 10 кОм), включенного между "землей" и выводом V_{CC}. Следует учитывать, что в более старых ЖК-табло на вывод V_O должно было подаваться отрицательное напряжение 0...–5 В, поэтому для определения правильной полярности в случаях, вызывающих сомнение, обращайтесь к спецификациям.

Регистр модуля HD44780, в который будут записываться значения, выбирают с помощью вывода RS (**R**egister **S**elect). Если в режиме доступа на запись на этот вывод подан сигнал низкого уровня, то записывается команда, в противном случае происходит запись в регистр данных. В режиме доступа на чтение состоянию RS = 0 соответствует комбинация флага занятости (Busy) и счетчика адреса, а состоянию RS = 1 — чтение из регистра данных.

Режим чтения или записи определяется по уровню сигнала на выводе R/W (вывод 5). Лог. 0 соответствует записи, а лог. 1 — чтению. Таким образом, получаем комбинацию сигналов для доступа к ЖК-модулю, представленную в табл. 16.1.

Таблица 16.1. Комбинации сигналов для доступа к ЖК-модулю

Функция	*RS*	*R/W*
Запись команды (табло гаснет, курсор смещается...)	0	0
Чтение флага занятости (разряд 7) и счетчика адреса (разряды 6...0)	0	1
Запись байта в память отображаемых данных или знакогенератора	1	0
Чтение байта из памяти отображаемых данных или знакогенератора	0	0

После того как управляющие сигналы RS и R/W стабильно установлены на протяжении как минимум 140 нс, микроконтроллер AVR подает на вывод 6 (E — Enable) модуля строб-импульс положительной полярности длительностью минимум 450 нс, по которому в контроллер HD44780 записываются биты данных, выставленные микроконтроллером AVR на выводы D0...D7, или же, наоборот, — микроконтроллер AVR считывает биты данных, выставленные на тех же выводах контроллером HD44780. До появления на выводе E ниспадающего фронта данные должны быть стабильны на протяжении минимум 195 нс.

Регистры модуля HD44780

Модуль HD44780 содержит два регистра, с помощью которых и происходит взаимодействие со внешним миром:

- регистр команд IR (**I**nstruction **R**egister);
- регистр данных DR (**D**ata **R**egister).

В регистр IR контроллер HD44780 принимает команды от микроконтроллера AVR, определяющие смещение курсора, гашение табло или установку адреса индикатора, в то время как регистр DR служит для промежуточного хранения данных, которые затем с помощью внутренних операций автоматически передаются в память отображаемых данных (DD — **D**isplay **D**ata) или в память знакогенератора (CG — **C**haracter **G**enerator).

Содержимое регистра IR не может быть прочитано микроконтроллером AVR. Возможно чтение только флага занятости (разряд 7) и текущего состояния внутреннего счетчика адреса AC (**A**ddress **C**ounter) (разряды 6...0).

Установленный флаг занятости (лог. 1) означает, что модуль HD44780 занят выполнением внутренней операции и в данный момент к приему не готов. Соответственно, с помощью лог. 0 флаг занятости сигнализирует о том, что модуль готов к приему.

Передача данных в HD44780 в тот момент, когда установлен флаг занятости, в некоторых случаях может даже привести к повреждению модуля, поэтому перед доступом на запись пользователь должен обязательно опрашивать флаг занятости в ожидании появления лог. 0.

Считывание данных из памяти DD или CG выполняется через регистр DR. После того как микроконтроллер AVR записывает некоторый адрес в регистр IR, байт данных, расположенный в памяти по этому адресу, с помощью внутренней операции переписывается в регистр DR. Процесс чтения завершается считыванием байта данных из этого регистра. Когда текущая операция чтения завершена, с помощью функции автоинкремента в регистр DR записывается байт данных, извлеченный из памяти по следующему адресу, который будет использован при очередном считывании.

Счетчик адреса, по сути, состоит из двух частей и содержит текущий адрес как в памяти DD, так и в памяти CG. После того как в регистр IR записана команда установки *а*дреса DD или CG, этот адрес с помощью внутренней операции автоматически переносится в соответствующий счетчик. Когда запись в память DD/CG (или чтения из нее) завершена, соответствующий счетчик автоматически инкрементируется (или декрементируется, если такой режим был выбран ранее с помощью специальной команды).

Память модуля HD44780

В модуле HD44780 используется две разные памяти:

- DD-RAM — для хранения отображаемых данных;
- CG-RAM — для хранения битовых комбинаций, которые соответствуют матрице размером 5x8 или 5x10 (определяет форму символа).

Доступ как к одной, так и к другой памяти осуществляется по текущему адресу, хранимому в счетчике адреса.

Емкость памяти DD составляет 80 знаков, представленных в 8-разрядной ASCII-кодировке. Из них на однострочном табло могут быть одновременно отображены только восемь символов (рис. 16.4), однако с помощью операции сдвига последующие символы также могут оказаться в отображаемой области.

1	2	3	4	5	6	7	8	9		79	80	Позиция на табло
00	01	02	03	04	05	06	07	08	---	4E	4F	Адрес DD-RAM

Рис. 16.4. На табло ЖК-модуля могут отображаться только 8 знаков из 80

Вид табло, соответствующего рис. 16.4, после одной операции сдвига влево показан на рис. 16.5 вверху, а после одной операции сдвига вправо — на рис. 16.5 внизу.

01	02	03	04	05	06	07	08	09	---	4F	00	Адрес DD-RAM
4F	00	01	02	03	04	05	06	07	---	4D	4E	Адрес DD-RAM

Рис. 16.5. Табло после сдвига влево (вверху) и вправо (внизу)

Вероятно, в будущем будут, в основном, применяется двухстрочные табло по 16 символов в каждой строке. В этом случае распределение адресов памяти соответствует рис. 16.6.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		39	40	Позиция на табло
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	---	26	27	Адрес DD-RAM
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	---	66	67	Адрес DD-RAM

Рис. 16.6. Двухстрочное табло по 16 символов в каждой строке

Первая строка начинается с адреса 00_h, а вторая — с адреса 40_h (64_d). Память разбита на две половины по 40_d (28_h) байт каждая, причем между последним адресом первой строки и первым адресом второй строки есть разрыв в 24_d байта.

После сдвига влево первая строка начинается с адреса $01, а вторая строка — с адреса $41, а после сдвига вправо первая строка начинается с адреса $27, а вторая строка — с адреса $67.

Знакогенератор модуля HD44780

Знакогенератор — это память типа ROM, предназначенная для хранения битовых комбинаций, соответствующих матрице размерами 5x8 или 5x10. В этой памяти можно хранить информацию о 208 (в случае матрицы 5x8) или 32 (в случае матрицы 5x10) символов. Распределение знаков показано на рис. 16.7, причем четыре старших разряда символа определяют его размещение по горизонтали, а четыре младших — по вертикали (например, "A" = $41).

Разработка собственных символов

У пользователя также есть возможность создавать собственные символы для отображения на табло. Для этого служит память CG-RAM, в которую можно записать битовый шаблон для восьми (в случае матрицы 5x8) или четырех (в случае матрицы 5x10) разных знаков. Для отображения собственных символов пользователя зарезервированы ASCII-коды $00–$07 (методика создания символов для матрицы 5x8 проиллюстрирована на рис. 16.8).

Поскольку разряд 3 в коде любого знака не используется, также становятся досупны символы с кодами $08–$0F. Как показано на рис. 16.7, все эти коды соответствуют памяти ROM знакогенератора.

Левый столбец на рис. 16.8 содержит ASCII-код символов. Старший полубайт, по определению, для всех знаков содержит нуль. Разряды 0...2 ASCII-кода соответствуют разрядам 3...5 адреса CG-RAM, показанного в среднем столбце. Разряды 0...2 адреса CG-RAM задают адрес строки в битовой комбинации (правый столбец).

Сам символ задается построчно. Восьмой строке соответствует позиция курсора. Той позиции в матрице, в которой должна отображаться точка на табло, назначается 1. Таким образом, информация о символе хранится в разрядах 0...4 восьми байтов данных. Например, символу "R" соответствуют восемь байтов, расположенных в CG-памяти по адресам $00–$07, которые содержат значения $1E, $11, $11, $1E, $14, $12, $11 и $00 (см. рис. 16.8).

Сброс табло

В модуле HD44780 используется встроенная схема сброса по включению питания. При этом реализуются следующие функции:

- начальная инициализация: DL = 1 — 8-разрядный интерфейс; N = 0 — однострочное табло; F = 0 — матрица 5x8;
- управление табло: D = 0 — табло отключено; C = 0 — курсор отключен; B = 0 — мерцание отключено;
- очистка табло;
- режим ввода данных: I/D = 1 — инкремент на 1; S = 0 — нет сдвига табло.

Рис. 16.7. Набор символов модуля HD44780 — знакогенератор

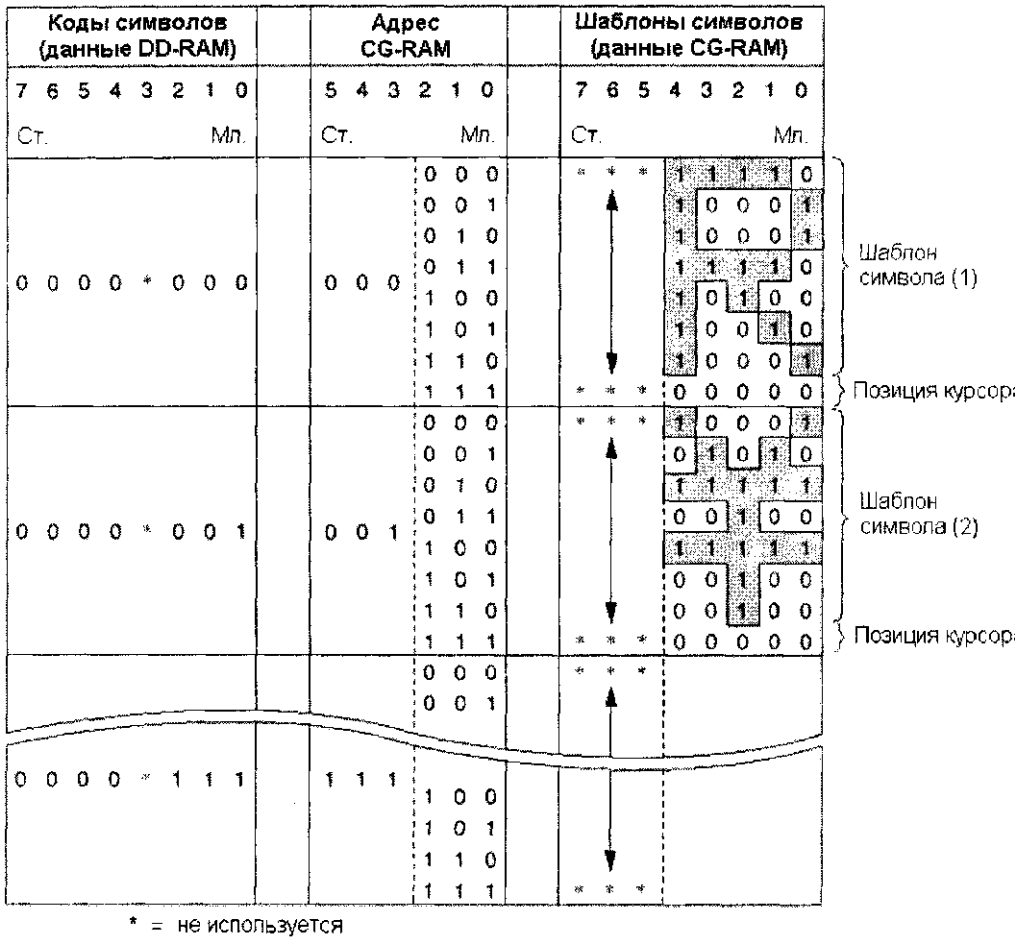


Рис. 16.8. Создание собственного шаблона символа

Однако, если питающее напряжение в течение 0,1...10 мс не достигнет уровня 4,5 В, внутренняя инициализация не сможет быть выполнена корректно. В этом случае необходимо следовать процедуре инициализации, показанной на рис. 16.9 (последние четыре команды этой процедуры идентичны тем, которые выполняются при инициализации по включению питания).

Как сказано в описании команды "Установка функции" (см. табл. 16.2), для экономии контактов ввода/вывода можно воспользоваться 4-хразрядной шиной данных. В этом случае каждая команда и каждый байт данных должны передаваться в два этапа через разряды D7...D4 (вначале — старший, а затем — младший полубайт), а разряды D3...D0 не используются.

Поскольку в таком варианте полубайты передаются непосредственно друг за другом, флаг занятости может быть опрошен только после передачи всей команды. Процедура инициализации для такого случая показана на рис. 16.10.

Система команд HD44780

Все функции управления табло реализуются с помощью команд, перечисленных в табл. 16.2. Следует учесть, что после записи или чтения байта данных адрес памяти DD или CG автоматически инкрементируется или декрементируется по ниспадающему фронту сигнала занятости. Однако новое значение в счетчик адреса AC записывается только спустя время $t_{ADD} = 1{,}5/ f_{OSC}$, где f_{OSC} — тактовая частота HD44780.

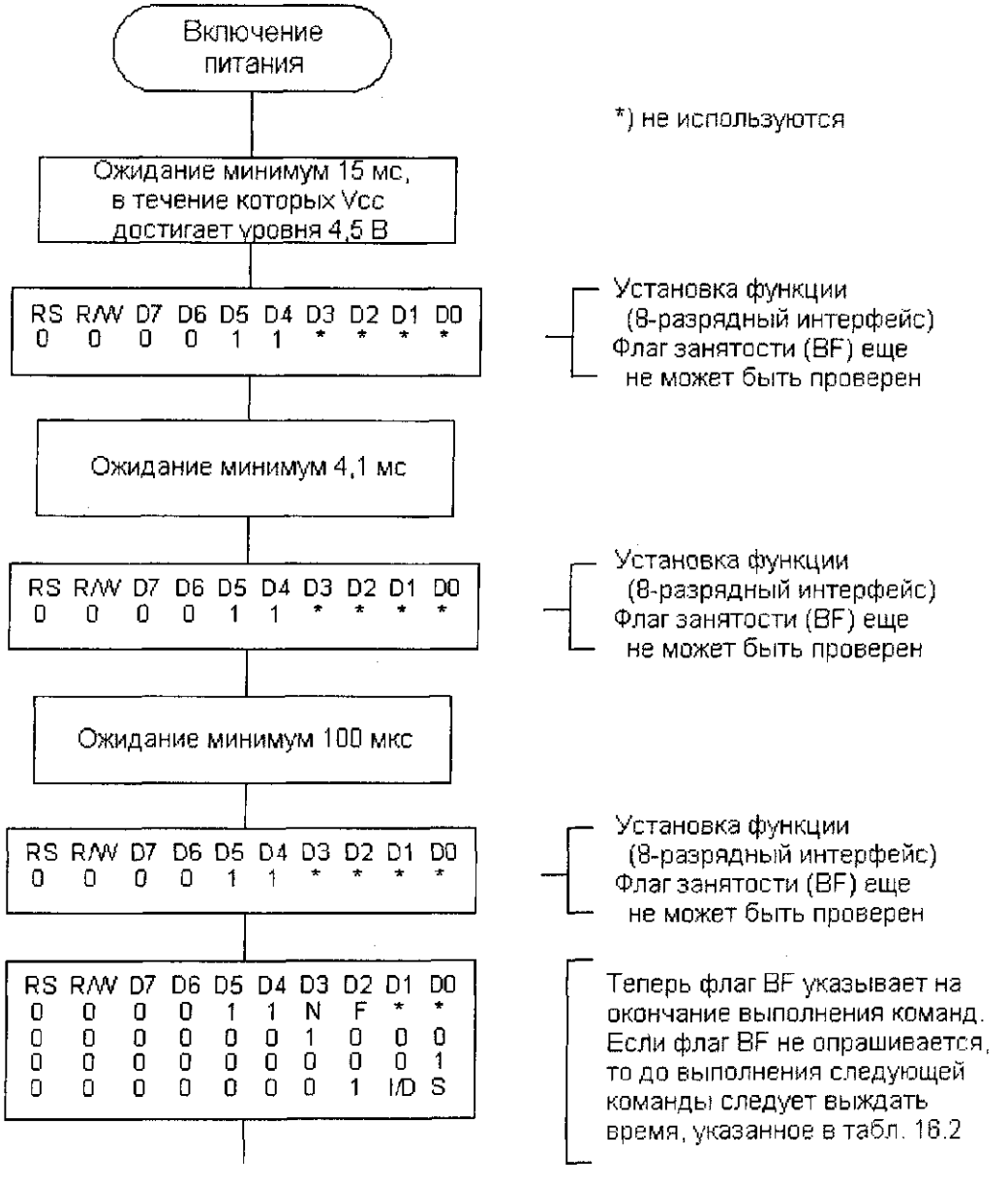


Рис. 16.9. Инициализация модуля HD44780 для 8-разрядного интерфейса

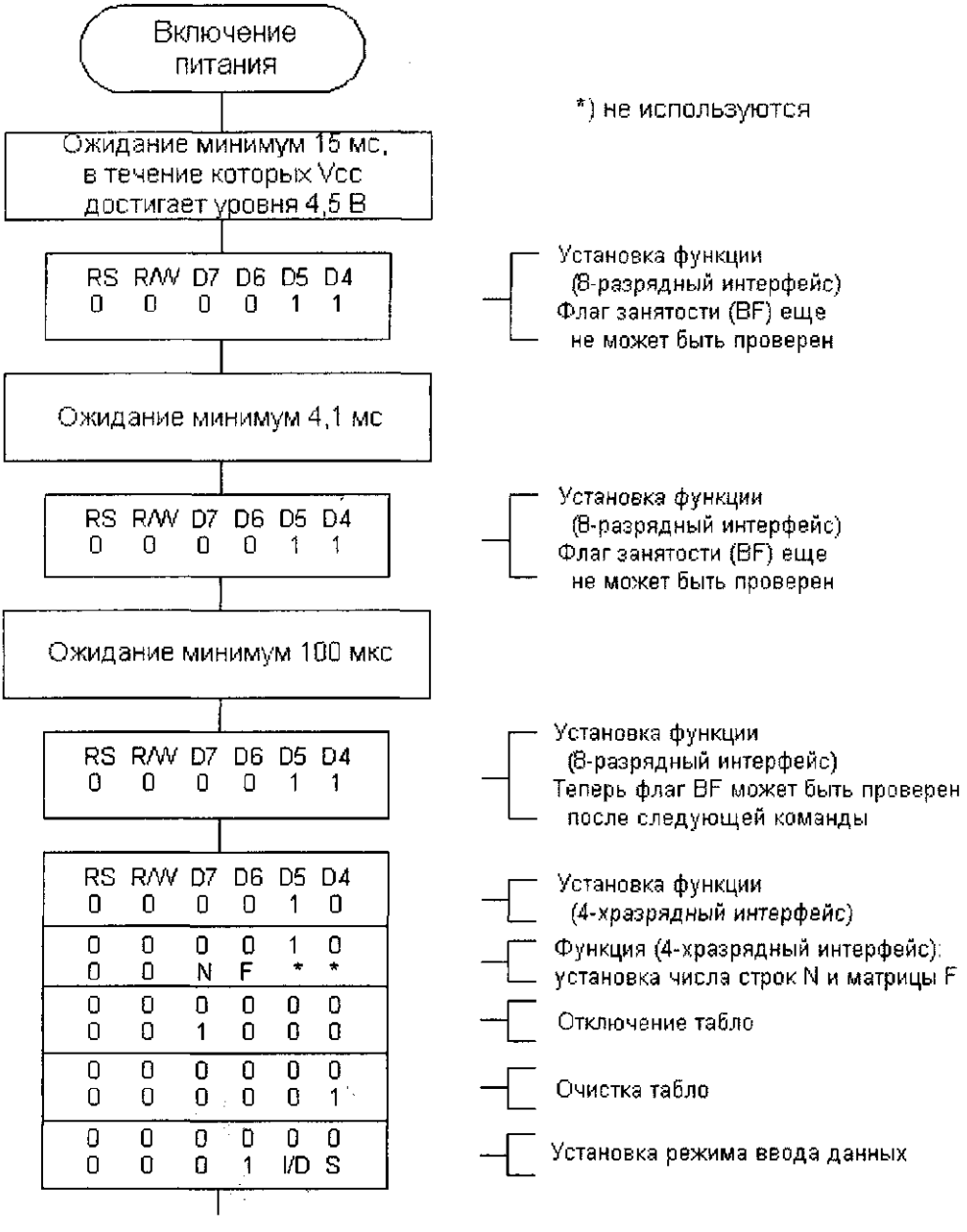


Рис. 16.10. Инициализация модуля HD44780 для 4-хразрядного интерфейса

Таблица 16.2. Система команд HD44780

Команда	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Описание	Время выполнения (макс.) при f_{osc} = 160 / 250 / 270 кГц
Очистка табло	0	0	0	0	0	0	0	0	0	1	Очищает память DD посредством записи $20 (знак пробела) и установки счетчика памяти DD в значение $00 (начало)	4,9 / 1,64 / 1,52 мс
Курсор в начало	0	0	0	0	0	0	0	0	1	*	Устанавливает курсор на адрес $00 (начало). Все сдвиги текста отменяются	4,9 / 1,64 / 1,52 мс
Режим ввода данных	0	0	0	0	0	0	0	1	I / D	S	Инкремент (I/D = 1) или декремент (I/D = 0) счетчика адреса на 1 после операции записи в память DD/CG или чтения из нее. Активизируется курсор и функция мерцания. Если S = 1, то содержимое табло сдвигается вправо (I/D = 1) или влево (I/D = 0)	120 / 40 / 37 мкс
Настройка табло	0	0	0	0	0	0	1	D	C	B	Включение (D = 1) или выключение (D = 0) табло. Отображение (C = 1) или сокрытие (C = 0) курсора. Если B = 1, то символ, на который указывает курсор, мерцает	120 / 40 / 37 мкс
Сдвиг табло или курсора	0	0	0	0	0	1	S / C	R / L	*	*	S/C=0, R/L=0 — курсор сдвигается влево (AC=AC–1); S/C=0, R/L=1 — курсор сдвигается вправо (AC=AC+1). S/C=1, R/L=0 — текст на табло сдвигается влево; S/C=1, R/L=1 — текст на табло сдвигается вправо	120 / 40 / 37 мкс
Установка функции (инициализация)	0	0	0	0	1	DL	N	F	*	*	DL = 1 — 8-разрядная шина данных; DL = 0 — 4-хразрядная шина данных. N = 1 — двухстрочное табло; N = 0 — однострочное табло, матрица 5x8 (F = 0) или 5x10 (F = 1)	120 / 40 / 37 мкс
Установка адреса памяти CG	0	0	0	1	ACG						Установка счетчика адреса памяти CG в 6-разрядный адрес "ACG"	120 / 40 / 37 мкс
Установка адреса памяти DD	0	0	1	ADD							Установка счетчика адреса памяти DD в 7-разрядный адрес "ADD"	120 / 40 / 37 мкс
Чтение флага занятости и адреса	0	1	BF	AC							Считывает флаг занятости и содержимое счетчика адреса AC. Тип считываемого адреса (DD или CG) определяется предыдущей командой установки адреса	0 / 0 / 0 мкс
Запись байта в память DD или CG	1	0	Байт данных								Записывает байт данных в память DD или CG, что определяется предыдущей командой установки адреса	120 / 40 / 37 мкс
Чтение байта из памяти DD или CG	1	1	Байт данных								Считывает байт данных из памяти DD или CG, что определяется предыдущей командой установки адреса	120 / 40 / 37 мкс

*) не используются

Пример для подключения двухстрочного табло по 16 знаков в строке

Подключим табло к ЖК-интерфейсу модуля STK200 в стандартном режиме ввода/вывода. В нем пользователю предоставляется возможность не только обычного вывода текста, но также определения восьми собственных знаков, сдвига курсора и реализации функции "мерцания". Описанные ниже подпрограммы полностью могут быть использованы читателем при разработке собственных проектов.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\163LCD.asm`

```
;**** Управление ЖК-модулем с помощью микроконтроллеров AVR ***************
;*
;* Управление двухстрочным ЖК-табло(16 знаков в строке,
;* тип контроллера - HD44780) через ЖК-интерфейс модуля STK200.
;* Тактовая частота AVR: 4 МГц (стандарт STK200).
;* Подпрограммы:
;* InitLCD: Инициализация ЖК-интерфейса для 8-разрядных символов
;* SendCom: Передает команду на ЖК-модуль
;* SendDat: Передает символ на ЖК-модуль
;* OutText: Передает текстовую строку на ЖК-модуль
;* DefChar: Определяет пользовательский символ
;*
;*****************************************************************************
.nolist
.include "8515def.inc"
.list
;***** Регистровые переменные
.def  tmp1 = r16      ; Рабочий регистр 1
.def  tmp2 = r17      ; Рабочий регистр 2
.def  prm1 = r18      ; Передаваемый параметр
.def  tim1 = r19      ; Счетчик цикла 1
.def  tim2 = r20      ; Счетчик цикла 2
.def  Cnt  = r21      ; Вспомогательный счетчик

.equ  RS   = 6        ; Register Select = разряд 6 порта C (A14)
.equ  Ena  = 7        ; Enable Impuls = разряд 7 порта C (A15)
.equ  RW   = 6        ; R/W = разряд 6 порта D (/WR)
.equ  RD   = 7        ; /RD = разряд 7 порта D
.equ  BF   = 7        ; Флаг занятости = разряд 7 порта A

.cseg

000000 c06c   rjmp Initial          ; После сброса - к главной программе

InitLCD:
000001 d05c   rcall wait5ms
000002 d05b   rcall wait5ms
000003 d05a   rcall wait5ms         ; Задержка на 15 мс после подачи питания
000004 e320   ldi prm1,$30          ; $30: установка функции, 8 разрядов
000005 d018   rcall SendCom         ; Отправка команды на ЖК-модуль
000006 d057   rcall wait5ms         ; Задержка на 5 мс
000007 e320   ldi prm1,$30          ; $30: установка функции, 8 разрядов
000008 d015   rcall SendCom         ; Повторяем команду
000009 d054   rcall wait5ms         ; Задержка на 5 мс
00000a e320   ldi prm1,$30          ; $30: установка функции, 8 разрядов
```

```
00000b d012    rcall SendCom          ; Повторяем команду
00000c d051    rcall wait5ms          ; Задержка на 5 мс
00000d d038    rcall WaitBusy         ; Ожидаем готовности ЖК-модуля к приему
00000e e328    ldi prm1,$38           ; $38: функция = 8 разрядов, 2 строки, 5x8
00000f d00e    rcall SendCom          ; Отправляем команду
000010 d049    rcall wait150us        ; Задержка на 150 мкс
000011 d034    rcall WaitBusy         ; Ожидаем готовности ЖК-модуля к приему
000012 e028    ldi prm1,$08           ; $08: отключение табло, курсора, мерцания
000013 d00a    rcall SendCom          ; Передаем команду
000014 d045    rcall wait150us        ; Задержка на 150 мкс
000015 d030    rcall WaitBusy         ; Ожидаем готовности ЖК-модуля к приему
000016 e021    ldi prm1,$01           ; $01: очистка табло
000017 d006    rcall SendCom          ; Передаем команду
000018 d045    rcall wait5ms          ; Задержка на 5 мс
000019 d02c    rcall WaitBusy         ; Ожидаем готовности ЖК-модуля к приему
00001a e026    ldi prm1,$06           ; $06: режим инкремента адреса, без сдвига
00001b d002    rcall SendCom          ; Передаем команду
00001c d03d    rcall wait150us        ; Задержка на 150 мкс
00001d 9508    ret

SendCom:                              ; Передача в ЖК-модуль команды
00001e 98ae    cbi PortC,RS           ; Register Select = команда
00001f d004    rcall Ausgabe
000020 9508    ret

SendDat:                              ; Передача в ЖК-модуль байта данных
000021 9aae    sbi PortC,RS           ; Register Select = байт данных
000022 d001    rcall Ausgabe
000023 9508    ret

Ausgabe:                              ; Длина данных = 8 бит
000024 bb2b    out PortA,prm1         ; Выдача команды/данных на порт A
000025 9aaf    sbi PortC,Ena          ; Подаем импульс разрешения
000026 0000    nop
000027 0000    nop                    ; Длительность импульса - минимум 450 нс
000028 98af    cbi PortC,Ena          ; Снимаем импульс разрешения
000029 9508    ret

OutText:                              ; Вывод символа <cnt> на табло
00002a 0fee    lsl ZL
00002b 1fff    rol ZH                 ; Указатель Z - на 1 позицию влево
OT1:
00002c 95c8    lpm                    ; Загружаем код символа в r0
00002d 2d20    mov prm1,r0            ; Копируем код символа в prm1
00002e d017    rcall WaitBusy         ; Ожидаем готовности ЖК-модуля к приему
00002f dff1    rcall SendDat          ; Передаем символ, автоинкремент адреса
000030 9631    adiw ZL,1              ; Инкрементируем указатель Z
000031 955a    dec cnt                ; Счетчик - 1
000032 f7c9    brne OT1               ; Выводим все символы на ЖК-табло
000033 9508    ret

DefChar:                              ; Определение пользовательских символов
000034 0f22    lsl prm1               ; Умножаем prm1 на 2
000035 0f22    lsl prm1               ; Умножаем prm1 на 4
000036 e0f4    ldi ZH,high(CharTab)   ; Адрес старшего байта шаблона символа
000037 0f22    lsl prm1               ; prm1 x 8
```

```
000038 2fe2    mov ZL,prm1             ; Z -> адрес символа
000039 1fff    rol ZH                  ; Указатель Z - на 1 позицию влево
00003a 6420    sbr prm1,1<<6           ; Разряд 6 = 1 -> адрес в памяти CG
00003b d00a    rcall WaitBusy          ; Ожидаем готовности ЖК-модуля к приему
00003c dfe1    rcall SendCom           ; Устанавливаем счетчик адреса памяти CG
00003d e058    ldi cnt,8
DC1:
00003e 95c8    lpm                     ; Загружаем в r0 шаблон символа
00003f 2d20    mov prm1,r0             ; Копируем шаблон символа в prm1
000040 d005    rcall WaitBusy          ; Ожидаем готовности ЖК-модуля к приему
000041 dfdf    rcall SendDat           ; Передаем шаблон, автоинкремент адреса
000042 9631    adiw ZL,1               ; Инкремент указателя Z
000043 955a    dec cnt                 ; Счетчик - 1
000044 f7c9    brne DC1                ; Копируем все 8 символов в память CG
000045 9508    ret
WaitBusy:                              ; Ожидание готовности ЖК-модуля к приему
000046 2700    clr tmp1
000047 bb0a    out DDRA,tmp1           ; Порт A = вход
000048 98ae    cbi PortC,RS            ; Register Select = команда
000049 9a96    sbi PortD,RW            ; Направление = чтение из ЖК-модуля
WB1:
00004a 9aaf    sbi PortC,Ena           ; Подаем импульс разрешения
00004b 0000    nop
00004c 0000    nop                     ; После макс. 320 нс стабильности данных
00004d b309    in tmp1,PinA            ; считываем BF и значение счетчика адреса
00004e 98af    cbi PortC,Ena           ; Снимаем импульс разрешения
00004f 0000    nop
000050 fd07    sbrc tmp1,BF            ; Пропускаем следующую команду, если
                                       ; ЖК-модуль готов (BF=0)
000051 cff8    rjmp WB1
000052 9896    cbi PortD,RW            ; Направление записи = в ЖК-модуль
000053 ef0f    ser tmp1
000054 bb0a    out DDRA,tmp1           ; Порт A = выход
000055 9508    ret

Wait50us:                              ; Такт системной синхронизации = 4 МГц
000056 e431    ldi tim1,65             ; Загрузка счетчика
Wait51:
000057 953a    dec tim1
000058 f7f1    brne Wait51
000059 9508    ret

Wait150us:                             ; Такт системной синхронизации = 4 МГц
00005a dffb    rcall wait50us
00005b dffa    rcall wait50us
00005c dff9    rcall wait50us
00005d 9508    ret

Wait5ms:                               ; Такт системной синхронизации = 4 МГц
00005e e644    ldi tim2,100            ; Загрузка счетчика
Wait501:
00005f dff6    rcall wait50us
000060 954a    dec tim2
000061 f7e9    brne Wait501
000062 9508    ret
```

```
Wait38ms:                               ; Такт системной синхронизации = 4 МГц
000063 e040    ldi tim2,0               ; Загрузка счетчика
Wait381:
000064 dff5    rcall wait150us
000065 954a    dec tim2
000066 f7e9    brne Wait381
000067 9508    ret

Wait5s:                                 ; Такт системной синхронизации = 4 МГц
000068 e852    ldi cnt,130              ; Загрузка счетчика
Wait5s1:
000069 dff9    rcall wait38ms
00006a 955a    dec cnt
00006b f7e9    brne Wait5s1
00006c 9508    ret

Initial:
00006d e002    ldi tmp1,High(RamEnd)
00006e bf0e    out sph,tmp1
00006f e50f    ldi tmp1,Low(RamEnd)
000070 bf0d    out spl,tmp1             ; Инициализируем стек
000071 e30f    ldi tmp1,$3F             ; Разряды 7..6 = 0, остальные = 1
000072 bb02    out PortD,tmp1           ; /RD,/WR = 0, остальные разряды порта D=1
000073 bb05    out PortC,tmp1           ; Ena, RS = 0, остальные разряды порта C=1
000074 ec00    ldi tmp1,$C0             ; Разряд 7 = 1, разряд 6 = 1
000075 bb01    out DDRD,tmp1            ; Разряды 7..6 - выходы, остальные - входы
000076 bb04    out DDRC,tmp1            ; Разряды 7..6 - выходы, остальные - входы
000077 ef0f    ser tmp1                 ; Устанавливаем tmp1
000078 bb0a    out DDRA,tmp1            ; Порт A - выход
000079 df87    rcall InitLCD            ; Инициализируем ЖК-модуль
00007a 2711    clr tmp2                 ; Счетчик символов
Ini1:
00007b 2f21    mov prm1,tmp2            ; ... переносим как код знака
00007c dfb7    rcall DefChar            ; Определяем символ для этого кода
00007d 9513    inc tmp2
00007e 3018    cpi tmp2,8               ; = 8?
00007f f7d9    brne Ini1                ; Переход, если нет

Haupt:
000080 e02c    ldi prm1,$0C             ; Включаем табло, откл. курсор и мерцание
000081 dfc4    rcall WaitBusy           ; Ожидаем готовность ЖК-модуля к приему
000082 df9b    rcall SendCom            ; Передаем команду
000083 e021    ldi prm1,$01             ; Очищаем табло, курсор - в начало
000084 dfc1    rcall WaitBusy           ; Ожидаем готовность ЖК-модуля к приему
000085 df98    rcall SendCom            ; Передаем команду
000086 e0f4    ldi ZH,high(Text3)
000087 e3e0    ldi ZL,low(Text3)        ; Z указывает на начало текста
000088 e150    ldi cnt,16               ; Текст состоит из 16 символов
000089 dfa0    rcall OutText            ; Выводим Text3

TestBlinken:
00008a e420    ldi prm1,64              ; Адрес = начало второй строки
00008b 6820    sbr prm1,1<<7            ; Устанавливаем разряд 7 -> память DD
00008c dfb9    rcall WaitBusy           ; Ожидаем готовность ЖК-модуля к приему
00008d df90    rcall SendCom            ; Устанавливаем указатель адреса в DD-RAM
00008e e0f4    ldi ZH,high(Text4)
```

```
00008f e3e8    ldi ZL,low(Text4)      ; Z указывает на начало текста
000090 e150    ldi cnt,16             ; Текст состоит из 16 символов
000091 df98    rcall OutText          ; Выводим Text4
000092 e425    ldi prm1,64+5          ; Адрес 6-го символа во второй строке
000093 6820    sbr prm1,1<<7          ; Устанавливаем разряд 7 -> память DD
000094 dfb1    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
000095 df88    rcall SendCom          ; Устанавливаем указатель адреса в DD-RAM
000096 dfaf    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
000097 e02d    ldi prm1,$0D           ; Включаем табло, мерцание, откл. курсор
000098 df85    rcall SendCom          ; Передаем команду
000099 dfce    rcall Wait5s           ; Задержка на 5 секунд
00009a e124    ldi prm1,$14           ; Смещаем курсор вправо
00009b dfaa    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
00009c df81    rcall SendCom          ; Передаем команду
00009d dfca    rcall Wait5s           ; Задержка на 5 секунд
00009e e124    ldi prm1,$14           ; Смещаем курсор вправо
00009f dfa6    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
0000a0 df7d    rcall SendCom          ; Передаем команду
0000a1 dfc6    rcall Wait5s           ; Задержка на 5 секунд

TestCursor:
0000a2 e02e    ldi prm1,$0E           ; Табло и курсор - вкл., мерцание - откл.
0000a3 dfa2    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
0000a4 df79    rcall SendCom          ; Передаем команду
0000a5 e420    ldi prm1,64            ; Адрес = начало второй строки
0000a6 6820    sbr prm1,1<<7          ; Устанавливаем разряд 7 -> память DD
0000a7 df9e    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
0000a8 df75    rcall SendCom          ; Устанавливаем указатель адреса в DD-RAM
0000a9 e0f4    ldi ZH,high(Text5)
0000aa e4e0    ldi ZL,low(Text5)      ; Z указывает на начало текста
0000ab e150    ldi cnt,16             ; Текст состоит из 16 символов
0000ac df7d    rcall OutText          ; Выводим Text5
0000ad e421    ldi prm1,64+1          ; Адрес 2-го символа во второй строке
0000ae 6820    sbr prm1,1<<7          ; Устанавливаем разряд 7 -> память DD
0000af df96    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
0000b0 df6d    rcall SendCom          ; Устанавливаем указатель адреса в DD-RAM
0000b1 dfb6    rcall Wait5s           ; Задержка на 5 секунд

Sonderzeichen:
0000b2 df93    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
0000b3 e021    ldi prm1,$01           ; Очищаем табло
0000b4 df69    rcall SendCom          ; Передаем команду
0000b5 df90    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
0000b6 e02c    ldi prm1,$0C           ; Табло - вкл., курсор и мерцание - откл.
0000b7 df66    rcall SendCom          ; Передаем команду
0000b8 e0f4    ldi ZH,high(Text1)
0000b9 e2e0    ldi ZL,low(Text1)      ; Z указывает на начало текста
0000ba e150    ldi cnt,16             ; Текст состоит из 16 символов
0000bb df6e    rcall OutText          ; Выводим Text1
0000bc e420    ldi prm1,64            ; Адрес = начало второй строки
0000bd 6820    sbr prm1,1<<7          ; Устанавливаем разряд 7 -> память DD
0000be df87    rcall WaitBusy         ; Ожидаем готовность ЖК-модуля к приему
0000bf df5e    rcall SendCom          ; Устанавливаем указатель адреса в DD-RAM
0000c0 e0f4    ldi ZH,high(Text2)
0000c1 e2e8    ldi ZL,low(Text2)      ; Z указывает на начало текста
0000c2 e150    ldi cnt,16             ; Текст состоит из 16 символов
```

```
0000c3 df66    rcall OutText          ; Выводим Text2 (8 служебных знаков)
0000c4 dfa3    rcall Wait5s           ; Задержка на 5 секунд
0000c5 dfa2    rcall Wait5s           ; Задержка на 5 секунд
0000c6 cfb9    rjmp Haupt             ; Бесконечный цикл

.org $400
CharTab:
.db $04,$04,$1F,$04,$04,$00,$1F,$00 ; Код символа $00: символ "+/-"
000400 0404
000401 041f
000402 0004
000403 001f
.db $0C,$02,$04,$08,$0E,$00,$00,$00 ; Код символа $01: символ "^2"
000404 020c
000405 0804
000406 000e
000407 0000
.db $0C,$02,$0C,$02,$0C,$00,$00,$00 ; Код символа $02: символ "^3"
000408 020c
000409 020c
00040a 000c
00040b 0000
.db $18,$06,$01,$06,$18,$00,$1F,$00 ; Код символа $03: символ ">="
00040c 0618
00040d 0601
00040e 0018
00040f 001f
.db $03,$0C,$10,$0C,$03,$00,$1F,$00 ; Код символа $04: символ "<="
000410 0c03
000411 0c10
000412 0003
000413 001f
.db $04,$04,$1F,$04,$1F,$04,$04,$00 ; Код символа $05: символ "неравно"
000414 0404
000415 041f
000416 041f
000417 0004
.db $00,$1F,$00,$1F,$00,$1F,$00,$00 ; Код символа $06: символ "равно"
000418 1f00
000419 1f00
00041a 1f00
00041b 0000
.db $15,$15,$15,$15,$15,$15,$15,$00 ; Код символа $06: символ "3 штриха"
00041c 1515
00041d 1515
00041e 1515
00041f 0015
Text1:
.db "Sonderzeichen: "
000420 6f53
000421 646e
000422 7265
000423 657a
000424 6369
000425 6568
000426 3a6e
```

```
000427 2020
Text2:
.db 0,' ',1,' ',2,' ',3,' ',4,' ',5,' ',6,' ',7,' '
Text3:
.db " Display-Test "
Text4:
.db "Char 6-8 blinken"
Text5:
.db "Cursor-Pos. = 65"
```

Описание подпрограмм

Подпрограмма `InitLCD` инициализирует интерфейс с ЖК-модулем для обмена 8-разрядными данными. Принцип соответствует рис. 16.9. Для перестраховки перед передачей каждой из четырех последних команд выжидается требуемое время и опрашивается состояние флага занятости.

Подпрограммы `SendCom` и `SendDat` передают на ЖК-модуль через подпрограмму `Ausgabe` команду или байт данных соответственно. При этом до того, как микроконтроллер AVR сможет вывести восемь бит через порт A, должен быть подан соответствующий сигнал RS. Период следования импульсов разрешения E в подпрограмме `Ausgabe` определяется для частоты такта системной синхронизации 4 МГц. При более высокой частоте необходимо с целью задержки вставить соответствующее количество дополнительных команд `nop`.

Подпрограмма `OutText` выводит на табло количество символов, заданное счетчиком `cnt`. Адрес первого символа определяется с помощью указателя Z. Сами тексты, размещенные в памяти команд в виде ASCII-констант, с помощью команды `lpm` загружаются в регистр `r0` и выводятся в ЖК-модуль.

Подпрограмма `DefChar` дает пользователю возможность определить собственные служебные знаки. Шаблон символа (см. подраздел "Разработка собственных символов" и рис. 16.8) должен быть определен в исходном коде с помощью директивы ассемблера `.db`. ASCII-код определенных таким образом служебных знаков ($00...$07) передается в подпрограмму `DefChar` с помощью параметра `prm1`. Подпрограмма `DefChar` на основании адреса таблицы служебных символов `CharTab` и ASCII-кода вычисляет смещение внутри `CharTab` — смещение адреса в памяти знакогенератора ЖК-модуля, по которому должны быть записаны восемь строк шаблона. Далее в цикле все восемь строк кода символа передаются в память CG. Указатель Z, определяющий смещение адреса в таблице `ChatTab`, должен циклически увеличиваться на 1. Адрес в памяти CG увеличивается автоматически, благодаря функции автоинкремента модуля HD44780.

Подпрограмма `WaitBusy` считывает состояние флага занятости и ожидает, пока оно не примет уровень лог. 0, указывающий на готовность модуля к приему. По умолчанию, для ЖК-модуля установлен режим доступа "Запись", поскольку все подпрограммы, кроме `WaitBusy`, выполняют запись в него. По этой причине в начале подпрограммы `WaitBusy` направление передачи данных для порта A должно быть изменено на "Вход", а сигнал на выходе R/W — на "Чтение" (лог. 1). Поскольку должен читаться регистр команд, дополнительно должен быть установлен в лог. 0 выход RS. Затем состояние модуля считывается до тех пор, пока

флаг занятости не примет уровень лог. 0. В завершение подпрограммы `WaitBusy` восстанавливается значения по умолчанию для порта A и для вывода R/W ("Запись").

Подпрограммы `Wait50us`, `Wait150us`, `Wait5ms`, `Wait38ms` и `Wait5s` выполняют задержку определенной длительности на основании частоты такта системной синхронизации 4 МГц. Для других значений частоты в эти подпрограммы должны быть внесены соответствующие коррективы

Описание главной программы

После сброса по включению питания происходит ветвление по адресу $000, которому соответствует команда перехода к метке `Initial`. После инициализации стека сразу же выполняется конфигурирование всех портов ввода/вывода для подключения ЖК-модуля, а также направление передачи данных по умолчанию. Затем ЖК-модуль настраивается на обмен по 8-разрядной шине данных, и определенные пользователем служебные символы передаются в знакогенератор (память CG).

Сама главная программа работает в бесконечном цикле. Вначале она включает табло, отключает курсор и функцию мерцания и сбрасывает индикацию. При этом курсор устанавливается в "начальную" позицию (адрес первого символа первой строки). Затем подпрограмма `OutText` выводит строку `Text3`, определенную в памяти команд с помощью директивы ассемблера `.db`.

Далее следует часть программы под названием `TestBlinken`, в которой на табло выводится разъясняющая строка `Text4`, после чего в течение пяти секунд мерцает последовательность из шестого, седьмого и восьмого символов второй строки табло. По сути, непосредственный опрос флага занятости после пятисекундной задержки не обязателен, и здесь он используется только для полноты примера.

В части программы, названной `TestCursor`, отображается курсор, на табло выводится информационная строка `Text5`, и курсор на пять секунд устанавливается в позиции второго символа второй строки.

В части программы под названием `Sonderzeichen` на табло вначале выводится сообщение о том, что далее последуют служебные символы ("Sonderzeichen"), а затем наблюдателю предоставляется возможность полюбоваться этими символами в течение десяти секунд. На этом бесконечный цикл завершается, и выполнение программы начинается сначала.

Формирование импульсов определенной длины с помощью T/C0

В программе для микроконтроллера AT90S1200 на выводе PD0, подключенном к внешнему устройству, уровень лог. 1 должен сохраняться на протяжении 20 мс, после чего этот вывод опять может перейти в состояние лог. 0. Такт системной синхронизации **Ф** составляет 12 МГц, и, следовательно, длительность периода составляет 83 1/3 нс. Фаза лог. 1 на выводе PD0 допускает незначительное отклонение в 0,5%.

Делитель частоты при установке мультиплексора Mux0 не сбрасывается, потому нельзя гарантировать, что длительность первого тактового импульса будет полной! В худшем случае длительность тактового сигнала делителя при запуске таймера T/C0 получается меньше длительности периода импульса. В таком случае первый тактовый сигнал практически не оказывает влияния на длительность импульса, что будет показано ниже в примере.

Для реализации поставленной задачи в счетный регистр TCNT0 необходимо загрузить значение \$15 (-235_d), а затем с помощью мультиплексора Mux0 выбрать больший коэффициент деления (**Ф**/1024). Таким образом, вход T/C0 будет работать на тактовой частоте 12 МГц / 1024 = 11,719 КГц. После 235-го импульса возникает переполнение счетчика из \$FF в \$00, в результате чего в регистре TIFR устанавливается флаг TOV0. Это соответствует отрезку времени в 83 1/3 нс · 1024 · 235 = 20,053 мс. Если в худшем случае первый тактовый сигнал делителя частоты не эффективен, то длительность импульса составит лишь 83 1/3 нс · 1024 · 234 = 19,968 мс. Оба результата находятся в пределах допуска 0,5% (+0,27% или -0,16%).

Если бы подобное отклонение не допускалось, то для переключения счетчика пришлось бы ожидать нарастающего фронта импульса, поскольку в этом случае начинается новый период такта делителя частоты.

Описание программы

При поступлении сигнала сброса регистр TCCR0 автоматически инициализируется значением \$00, в результате чего T/C0 останавливается. После сброса по включению питания происходит ветвление по адресу \$000, где находится команда перехода к метке `Initial` (обозначает часть инициализации). В части программы, выполняющей инициализацию, в первую очередь определяется как выход разряд 0 порта D. Для этого в соответствующий разряд регистра направления передачи данных DDRD записывается лог. 1. Остальные разряды порта D действуют как входы, поскольку при поступлении сигнала сброса по включению питания во всех них уже был записан лог. 0.

На этом инициализация используемого в данном примере регистра завершена. Главная часть программы (метка `Haupt`) начинается с цикла, символизирующего временную задержку перед выполнением последующих команд. Он необходим для того, чтобы корректно обработать различие в длительности импульсов, поскольку после сброса по включению питания начинает работать делитель частоты, формирующий входной такт таймера.

После выхода из этого цикла в счетный регистр TCNT0 загружается отрицательное значение длительности интервала. Этот интервал, представленный константой `Time`, должен быть отрицательным, поскольку T/C0 работает как суммирующий счетчик. Затем, в соответствии с поставленной задачей, разряд 0 порта D устанавливается в лог. 1 (нарастающий фронт импульса).

В следующей части программы с помощью мультиплексора выбирается входной такт для T/C0 — такт системной синхронизации **Ф**, деленный на 1024. Затем в регистр TCCR0 записывается значение \$05 (см. табл. 4.2), и T/C0 начинает счет.

Далее, начиная с метки `Wait`, расположен цикл, в котором с помощью опроса флага TOV0 реализовано ожидание переполнения счетчика. Как только этот флаг будет установлен, разряд 0 порта D сбрасывается в лог. 0 (ниспадающий фронт

импульса). В завершение флаг TOV0 сбрасывается, чтобы опять сделать возможным ожидание переполнения T/C0.

Имя файла на прилагаемом к книге компакт-диске: \Programm\164TC0.asm

```
;**** Создание импульсов с помощью T/C0 ***********************************
;*
;* Формирование на выводе 0 порта D импульсов (лог. 1) длительностью 20 мс
;* с помощью T/C0 микроконтроллера AT90S1200.
;*
;* Тактовая частота микроконтроллера: 12 МГц.
;*
;*****************************************************************************

.nolist
.include "1200def.inc"
.list

.def    Work1 = r16     ; Рабочий регистр
.equ    Time = 235      ; Количество тактовых импульсов таймера
.equ    Mux = 5         ; Mux0 = 5 для деления частоты на 1024

Reset:                          ; Сброс по включению питания
000000 c003    rjmp Initial     ; Адрес 000 - переход к части инициализации
000001 9518    reti             ; Адреса 001 - 003 - прерывания (не используем)
000002 9518    reti
000003 9518    reti
Initial:
000004 e001    ldi Work1,$01 ; Загрузка направления передачи данных
000005 bb01    out DDRD,Work1; Порт D: разряд 0 - выход, остальные - входы
Haupt:
000006 ef0a    ldi Work1,250
Schleife:
000007 0000    nop
000008 950a    dec Work1
000009 f7e9    brne Schleife          ; Символический цикл задержки
Impuls:
00000a e105    ldi Work1,-Time        ; Загружаем интервал для таймера
00000b bf02    out TCNT0,Work1        ; Инициализируем счетчик
00000c 9a90    sbi PortD,PD0          ; Устанавливаем разряд 0 порта D в лог. 1
Beginn:
00000d e005    ldi Work1,Mux          ; Устанавливаем адрес Mux0
00000e bf03    out TCCR0,Work1        ; Запускаем таймер, коэф. деления = 1024
Wait:
00000f b708    in Work1,TIFR          ; Извлекаем содержимое регистра TIFR
000010 ff01    sbrs Work1, TOV0       ; Пропускаем следующую команду, если
                                      ;   возникло переполнение таймера
000011 cffd    rjmp Wait              ; Дальнейшее ожидание переполнения
000012 9890    cbi PortD, 0           ; Порт D: разряд 0 сбрасываем в лог. 0
Ende:
000013 e002    ldi Work1, (1<<TOV0)   ; Загружаем битовую позицию флага TOV0
000014 bf08    out TIFR, Work1        ; Сбрасываем флаг TOV0
000015 0000    nop                    ; Символическое продолжение программы
```

На рис. 16.11 показано окно AVR-Studio, в котором эмулирован ход выполнения программы на 250-м проходе начального цикла задержки.

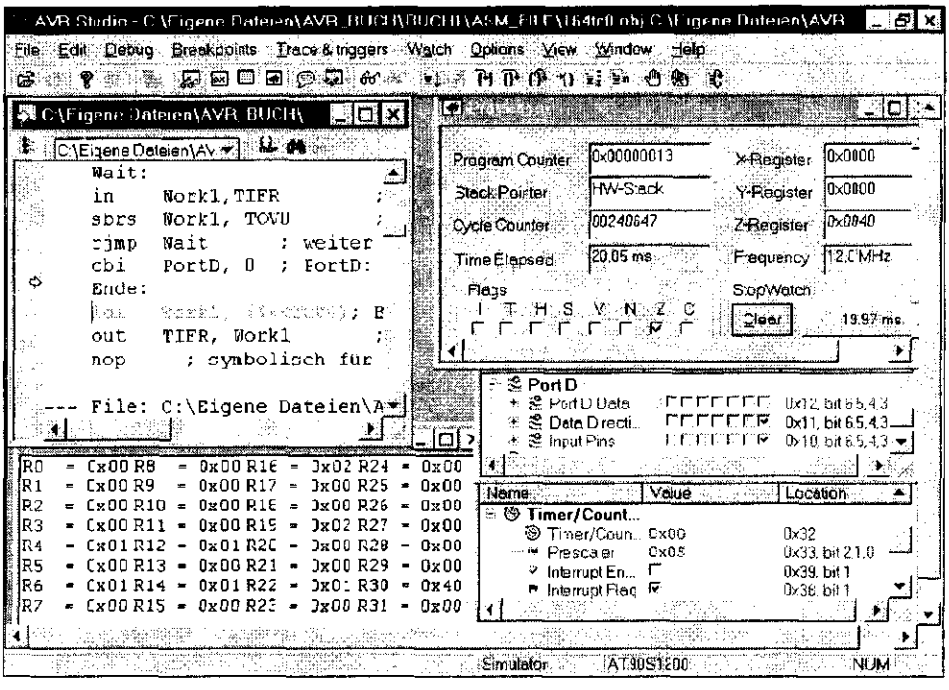


Рис. 16.11. Эмуляция хода выполнения программы на 250-м проходе начального цикла задержки

По нарастающему фронту импульса (метка `Beginn`) секундомер (поле **Stop-Watch**) в окне **Processor** был сброшен в нуль. До момента достижения метки `Ende` (ниспадающего фронта импульса) прошло 19,97 мс. Таким образом, длительность импульса лежит в допустимых пределах 19,9...20,1 мс.

Выполним второй тестовый прогон, предварительно внеся коррективы в программу, чтобы для инициализации переменной `Work1` после метки `Haupt` использовалось значение 1, а не 250. Задержка, соответствующая значению 250, составляла почти полный период такта делителя частоты, и потому, согласно измерениям секундомера AVR-Studio между метками `Beginn` и `Ende`, была получена длительность импульса 19,97 мс. Как и следовало ожидать, длительность импульса при втором прогоне составила 20,05 мс, поскольку в вычислениях была задействована почти полная длительность первого тактового сигнала.

Программная реализация автоматической перезагрузки T/C0

В базовой серии микроконтроллеров AVR автоматическая перезагрузка T/C0 не предусмотрена. Это означает, что в результате переполнения (возникает при поступлении очередного тактового импульса, когда счетный регистр содержит значение $FF) состояние счетчика всегда переходит в $00, и никакое другое значение аппаратно в счетный регистр автоматически не загружается. Тем не менее, это можно выполнить программно, как будет показано в следующем примере программы для микроконтроллера AT90S1200.

В этом примере на выводах PD4, PD5 и PD6 порта D должны вырабатываться импульсы с частотой 50, 25 и 12,5 кГц соответственно. Период для наибольшей частоты составляет 1/50 кГц = 20 мкс. Таким образом, для переключения уровня на выводе 4 порта D прерывание от таймера/счетчика T/C0 должно возникать каждые 10 мкс. Старший полубайт порта D каждые 10 мкс декрементируется на 1, потому изменение логического уровня на выводе 4 порта D происходит при каждом прерывании, на выводе 5 — при каждом втором, а на выводе 6 — при каждом четвертом прерывании.

Временная диаграмма уровней напряжения на выходах PD4, PD5 и PD6 порта D представлена на рис. 16.12.

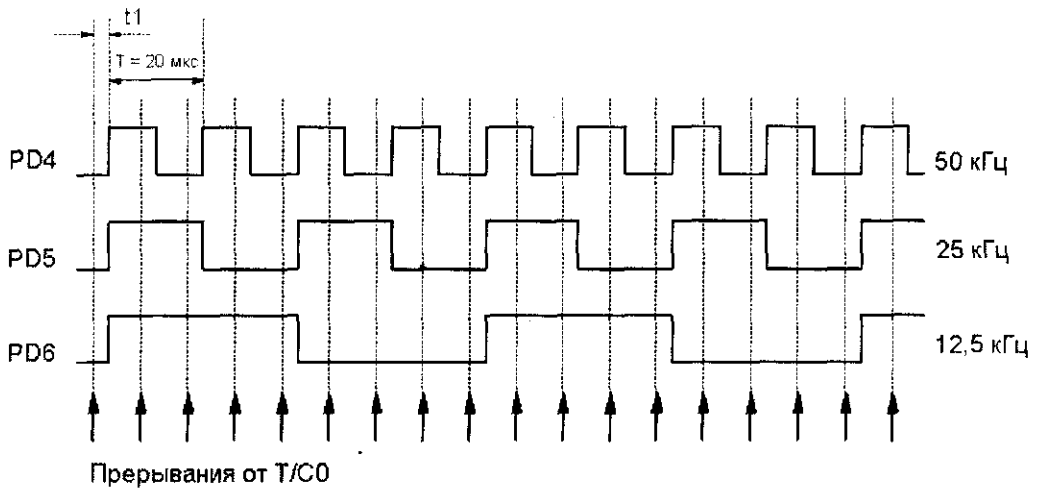


Рис. 16.12. Временная диаграмма уровней напряжения на выходах порта D

Промежуток времени от появления запроса на прерывание до вывода нового состояния счетчика в порт D символически обозначен на рис. 16.12 как t1 и обусловлен временными затратами на выполнение команд по адресам $002–$008, а также временем, затрачиваемым на переход к подпрограмме обработки прерывания.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\165TC0.asm`

```
;**** Программная реализация автоматической перезагрузки T/C0 **********
;*
;* При каждом прерывании от T/C0 в счетный регистр микроконтроллера
;* AT90S1200 программно загружается значение автоперезагрузки.
;*
;* Тактовая частота микроконтроллера: 12 МГц.
;*
;***************************************************************************
.nolist
.include "1200def.inc"
.list

.def    STAT = r15      ; Регистр для хранения состояния главной программы
.def    WorkI = r16     ; Рабочий регистр подпрограммы обработки прерывания
.def    WorkH = r17     ; Рабочий регистр главной программы
```

```
.def    Count = r18     ; Счетчик прерываний

.equ    Time = 120      ; Количество тактовых импульсов до следующего прерывания
                        ; 120 импульсов при F = 12 МГц -> интервал 10 мкс

Reset:
000000 c00b    rjmp Initial          ; Переход к части инициализации
000001 9518    reti                  ; Внешнее прерывание 0 (не используется)
Timer_Int:                           ; Прерывание от таймера 0 в AT90S1200
000002 b6ff    in STAT,SREG          ; Сохраняем флаги главной программы
000003 b702    in WorkI,TCNT0        ; Извлекаем текущее содержимое счетчика
000004 5705    subi WorkI,Time-3     ; Устанавливаем таймер по-новому:
000005 bf02    out TCNT0,WorkI       ;   длительность интервала минус время
                                     ;   выполнения первых трех команд
000006 b302    in WorkI,PortD        ; Извлекаем содержимое порта D
000007 5100    subi WorkI,$10        ; Декрементируем старший полубайт на 1
000008 bb02    out PortD,WorkI       ; Выводим новое содержимое порта D
000009 952a    dec Count             ; Декрементируем счетчик интервалов
00000a beff    out SREG,STAT         ; Восстанавливаем старые флаги
00000b 9518    reti                  ; Конец подпрограммы обработки прерывания
Initial:                             ; Инициализация регистра ввода/вывода
00000c e010    ldi WorkH,$00         ; На всех выводах порта D - лог. 0
00000d bb12    out PortD,WorkH       ; Вывод в порт D
00000e e71f    ldi WorkH,$7F         ; Загружаем направление передачи данных
00000f bb11    out DDRD,WorkH        ; Порт D: все разряды - выходы
000010 e818    ldi WorkH,-Time       ; Загружаем интервал (отрицательный,
                                     ;    потому что счетчик - суммирующий)
000011 bf12    out TCNT0,WorkH       ; Инициализируем таймер, T=10мкс при 12МГц
000012 e011    ldi WorkH,1
000013 bf13    out TCCR0,WorkH       ; Запускаем таймер (с тактом сист. синхр.)
000014 e012    ldi WorkH,2
000015 bf19    out TIMSK,WorkH       ; Разрешаем прерывание от таймера
000016 ef2b    ldi Count,251         ; Инициализируем счетчик интервалов
000017 9478    sei                   ; Общее разрешение прерываний (разряд I)
Haupt:
000018 cfff    rjmp Haupt            ; Главная программа: ожидание прерывания
```

Описание подпрограммы обработки прерывания

Переполнение счетчика из $FF в $00 устанавливает флаг TOV0, и происходит переход к подпрограмме обработки прерывания по адресу $002 (метка `Timer_Int`). При этом флаг TOV0 автоматически сбрасывается. Первая команда в подпрограмме обработки прерывания сохраняет регистр состояния, содержащий флаги главной программы.

Счетчик TCNT0 после входа в подпрограмму обработки прерывания принимает начальное состояние $04 или $05, в зависимости от того, сколько тактовых импульсов требуется для выполнения команды в момент возникновения переполнения: один или два. Такое начальное значение обусловлено тем, что для реакции на запрос на прерывание требуется четыре тактовых импульса.

Для того чтобы загрузить точное значение интервала, из текущего содержимого регистра TCNT0, которое до этого момента хранилось в рабочем регистре `WorkI`, вычитается предустановленная величина длительности `Time`. Принимая во

внимание время выполнения трех команд, расположенных по адресам $002–$004, значение интервала, представленное константой Time, прежде, чем оно будет вычтено из текущего состояния счетчика и загружено в счетный регистр, дополнительно уменьшается на 3.

Далее вычисляется новое состояние счетчика, которое затем выводится в порт D. Счетчик интервалов Count, который постоянно декрементируется, используется в программе только для проверки точности следования интервалов в среде AVR-Studio. Перед возвращением в главную программу восстанавливаются флаги, а также вновь устанавливается общее разрешение прерываний с помощью команды reti.

Время выполнения подпрограммы обработки прерывания таймера 0

На выполнение девяти команд подпрограммы (без reti) требуется девять тактовых циклов. Кроме того, следует учесть время на переход к подпрограмме и время, затрачиваемое на выполнение команды reti (четыре такта). В результате подпрограмма обработки прерывания занимает в общей сложности 17 тактовых импульсов, что при такте системной синхронизации **Ф** = 12 МГц составляет 1,417 мкс. Поскольку подпрограмма обработки прерывания вызывается только каждые 10 мкс, на выполнение оставшейся части программы остается 85,8% эффективного времени работы системы.

Описание главной программы

При поступлении сигнала сброса регистр TCCR0 автоматически инициализируется значением $00, тем самым останавливая T/C0. В результате сброса по включению питания происходит ветвление по адресу $000, где расположена команда перехода к метке Initial. В части инициализации в первую очередь сбрасываются в лог. 0 все разряды выводов порта D, а затем все разряды порта D определяются как выходы посредством записи лог. 1 в регистр направления передачи данных DDRD. Далее счетный регистр TCNT0 инициализируется отрицательным значением (длительность интервала, представленная константой Time, должна быть отрицательной, поскольку T/C0 работает как суммирующий счетчик).

В следующей части в качестве входного такта для T/C0 с помощью мультиплексора выбирается непосредственно значение такта системной синхронизации **Ф**, после чего в регистр TCCR0 записывается значение $01 (см. табл. 4.2), и T/C0 начинает счет. Для того чтобы разрешить прерывания от T/C0, должен быть установлен разряд TOIE0 в регистре TIMSK, а для общего разрешения прерываний — флаг I в регистре состояния SREG.

Еще до завершения инициализации регистра, используемого в данном программном примере, переменная-счетчик Count, которая предназначена исключительно для отладки, инициализируется значением 251. Главная часть программы, которая начинается с метки Haupt, представляет собой бесконечный цикл в ожидании прерывания от T/C0.

На рис. 16.13 представлено окно AVR-Studio, в котором эмулирован ход выполнения программы.

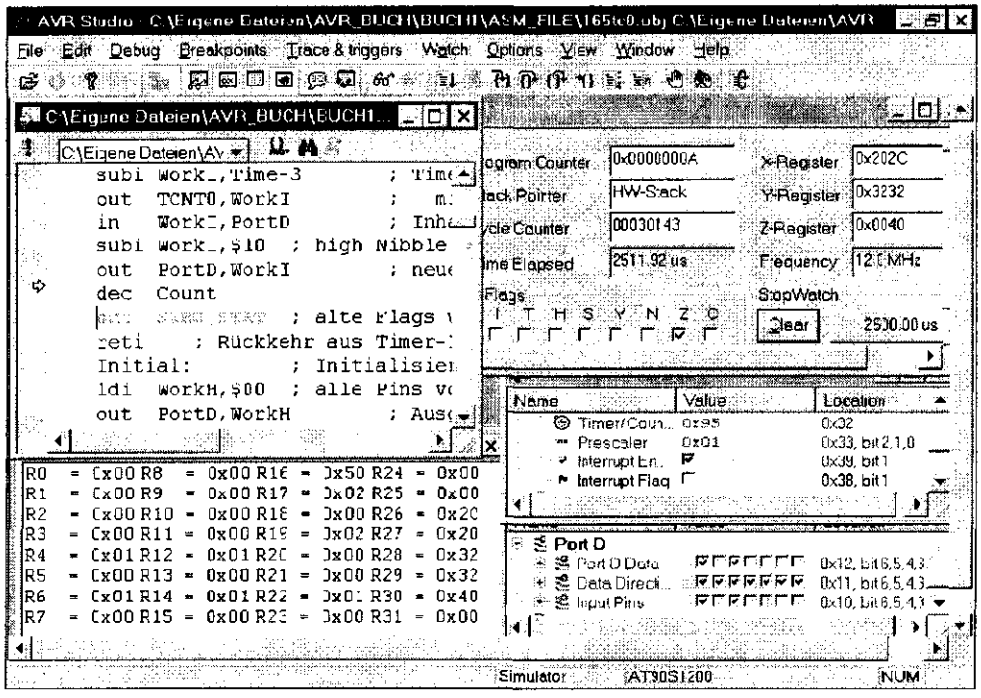


Рис. 16.13. Эмуляция хода выполнения программы в AVR-Studio

При первом вызове подпрограммы обработки прерывания секундомер (поле **StopWatch**) в окне **Processor** после выполнения команды `dec Count` был сброшен в нуль. К моменту, показанному на рис. 16.13, было подсчитано 250 вызовов прерывания (r18 =0), на что ушло 2,5 мс — время, которое соответствует заданному периоду следования прерываний 10 мкс.

Данная программа с помощью набора STK200 была введена в микроконтроллер AT90S1200, после чего были замерены частоты на выходах PD6, PD5 и PD4. Полученные результаты в точности соответствовали постановке задачи.

Выработка с помощью T/C1 импульсов с частотой 50 Гц и коэффициентом заполнения 0,025

Микроконтроллер AT90S8515, работающий на частоте системной синхронизации **Ф** = 4 МГц, должен создавать на своем выходе OC1A импульсы с частотой 50 Гц и коэффициентом заполнения g = 0,025. Требуемая временн*ая* диаграмма уровня напряжения на выходе OC1A показана на рис. 16.14.

Непосредственно с помощью таймера/счетчика T/C1 получить частоту 50 Гц с таким коэффициентом заполнения из такта системной синхронизации 4 МГц невозможно. Однако с помощью разрядов CS10...CS12 регистра TCCR1B в качестве входного такта для T/C1 можно выбрать такт системной синхронизации, деленный на 8. Таким образом, поскольку частота входных тактирующих импульсов составляет 500 кГц, для T/C1 достаточно коэффициента деления 10000. Для того чтобы добиться требуемого коэффициента заполнения g = 0,025, продолжительность вы-

сокого уровня выходного сигнала должна составлять 250 тактов, а низкого уровня — 9750 тактов.

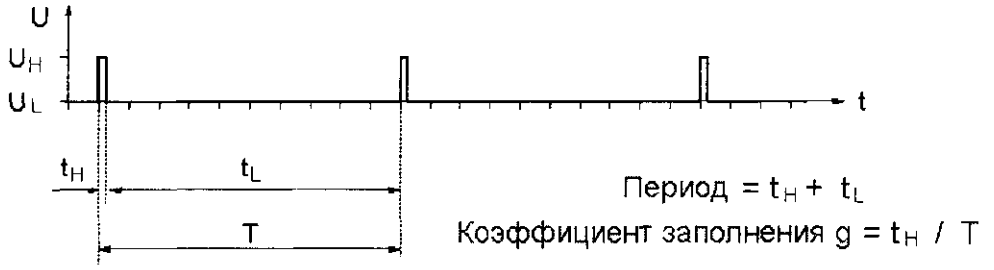


Рис. 16.14. Требуемая временная диаграмма уровня напряжения на выходе OC1A

Для решения этой задачи воспользуемся регистром сравнения OCR1A таймера/счетчика T/C1. В подпрограмме обработки прерывания, которая вызывается при совпадении состояния счетчика T/C1 с содержимым регистра сравнения A, к текущему содержимому регистра OCR1A прибавляется попеременно 250 и 9750, и соответствующим образом меняется уровень сигнала на выходе OC1A (рис. 16.15).

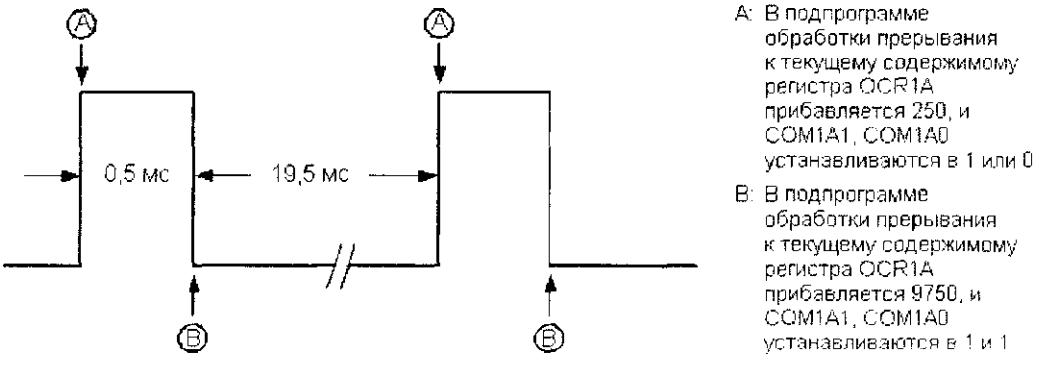


Рис. 16.15. Установка значения регистра OCR1A, а также COM1A1, COM1A0 в процессе выполнения подпрограммы обработки прерывания CompareA

Переполнение, возникающее во время прибавления константы к текущему содержимому регистра OCR1A, на эту функцию никак не влияет, поскольку содержимое счетного регистра TCNT1 также переполняется из $FFFF в $0000.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\166TC1.asm`

```
;**** Частота 50 Гц, g=0,025 с помощью T/C1 ****************************
;*
;* На выходе OC1A микроконтроллера AT90S8515 получаем выходные импульсы
;* с частотой 50 Гц и коэффициентом заполнения g = 0,025.
;*
;* Тактовая частота микроконтроллеров AVR: 4 МГц (Стандарт STK200)
;*
;***********************************************************************
.nolist
.include "8515def.inc"
.list
```

```
.def    STAT = r15      ; Регистр для хранения состояния главной программы
.def    WorkH = r16     ; Рабочий регистр для главной программы
.def    WorkI = r17     ; Рабочий регистр для прерывания от таймера
.def    AddLo = r18     ; Вспомогательный регистр для прибавления константы

.equ    Impuls = 250    ; Число тактовых сигналов, соответствующее импульсу
.equ    Pause = 9750    ; Число тактовых сигналов, соответствующее паузе

Reset:
000000 c018    rjmp Initial             ; Переход к части инициализации
000001 9518    reti                     ; Внешнее прерывание 0 (не используется)
000002 9518    reti                     ; Внешнее прерывание 1 (не используется)
000003 9518    reti                     ; Прерывание по захвату (не используется)
Timer1CompA:                            ; Прерывание CompareA для AT90S8515
000004 b6ff    in STAT,SREG             ; Сохраняем флаги главной программы
000005 b51f    in WorkI,TCCR1A          ; Регистр управления A счетчика T/C1
000006 ff16    sbrs WorkI,COM1A0        ; Пропускаем следующую команду, если
                                        ; Com1A0 = 1
000007 c007    rjmp TC1                 ; Пропускаем следующую команду, если
                                        ; Com1A0 = 0
000008 7b1f    cbr WorkI,1<<COM1A0      ; При следующем прерывании OC1A -> 0
000009 bd1f    out TCCR1A,WorkI         ; Сохраняем новое значение
00000a b52a    in AddLo,OCR1AL          ; Младший байт OCR1A
00000b 5026    subi AddLo,Low(-Impuls)  ; Прибавляем младший байт значения
                                        ; длительности импульса
00000c b51b    in WorkI,OCR1AH          ; Старший байт OCR1A
00000d 4f1f    sbci WorkI,High(-Impuls) ; Прибавляем старший байт значения
                                        ; длительности импульса
00000e c006    rjmp IntEnde
TC1:
00000f 6410    sbr WorkI,1<<COM1A0      ; При следующем прерывании OC1A -> 1
000010 bd1f    out TCCR1A,WorkI         ; Сохраняем новое значение
000011 b52a    in AddLo,OCR1AL          ; Младший байт OCR1A
000012 5e2a    subi AddLo,Low(-Pause)   ; Прибавляем младший байт значения
                                        ; длительности импульса
000013 b51b    in WorkI,OCR1AH          ; Старший байт OCR1A
000014 4d19    sbci WorkI,High(-Pause)  ; Прибавляем старший байт значения
                                        ; длительности импульса
IntEnde:
000015 bd1b    out OCR1AH,WorkI         ; Назад - старший байт нового значения
000016 bd2a    out OCR1AL,AddLo         ; Назад - младший байт нового значения
000017 beff    out SREG,STAT            ; Восстанавливаем старые флаги
000018 9518    reti                     ; Выход из обработки прерывания CompareA

Initial:                                ; Инициализация регистра ввода/вывода
000019 e002    ldi WorkH,High(RamEnd)
00001a bf0e    out sph,WorkH
00001b e50f    ldi WorkH,Low(RamEnd)
00001c bf0d    out spl,WorkH            ; Инициализируем стек
00001d 2700    clr WorkH                ; Все разряды = 0
00001e bb02    out PortD,WorkH          ; Вывод в порт D
00001f e200    ldi WorkH,$20            ; Разряд 5 - в 1, остальные - в 0
000020 bb01    out DDRD,WorkH           ; OC1A (PD5) - выход, остальные - входы
000021 e206    ldi WorkH,High(Pause)    ; Старший байт значения паузы
000022 bd0b    out OCR1AH,WorkH         ;   в старший байт регистра сравнения A
```

```
000023 e106    ldi WorkH,Low(Pause)  ; Младший байт значения паузы
000024 bd0a    out OCR1AL,WorkH      ;   в младший байт регистра сравнения A
000025 ec00    ldi WorkH,$C0         ; OC1A -> 1
000026 bd0f    out TCCR1A,WorkH      ; Сохраняем новое значение
000027 e002    ldi WorkH,$02         ; Входной такт T/C1 = такт сист. синхр./8
000028 bd0e    out TCCR1B,WorkH      ; Запускаем T/C1
000029 e400    ldi WorkH,$40
00002a bf09    out TIMSK,WorkH       ; Разрешаем прерывание CompareA
00002b 9478    sei                   ; Общее разрешение прерываний (разряд I)
Haupt:
00002c cfff    rjmp Haupt            ; Главная часть программы: ожидаем
                                     ; возникновения прерывания
```

Описание подпрограммы обработки прерывания

При совпадении содержимого счетного регистра и регистра сравнения A выполнение программы переходит к метке `Timer1CompA` по адресу $004 (подпрограмме обработки прерывания). При этом автоматически сбрасывается флаг OCF1A в регистре TIFR, и выход OC1A переключается в соответствии с разрядами COM1A1 и COM1A0 регистра TCCR1A.

Первая команда в подпрограмме сохраняет регистр состояния, который в данный момент содержит флаги главной программы. Далее проверяется текущая фаза: импульс или пауза. Если COM1A0 соответствует низкий уровень, то после входа в подпрограмму сразу же устанавливается в лог. 0 выход OC1A. В этом случае текущей является фаза паузы, и содержимое регистра сравнения A необходимо увеличить на длительность паузы, прибавив к нему константу `Pause`. Поскольку в системе команд отсутствует команда прибавления константы, это сложение реализовано через вычитание отрицательного значения. Кроме того, COM1A0 переходит в состояние высокого уровня, поскольку при следующем сопоставлении содержимого счетного регистра и регистра сравнения A в момент завершения паузы должен последовать нарастающий фронт импульса.

Если же COM1A0, наоборот, соответствует высокий уровень, то после входа в подпрограмму обработки прерывания выход OC1A переключается в лог. 1. В этом случае текущей является фаза импульса, и содержимое регистра сравнения A необходимо увеличить на длительность импульса, прибавив к нему константу `Impulse`. Кроме того, COM1A0 переходит в состояние низкого уровня, поскольку при следующем сопоставлении содержимого счетного регистра и регистра сравнения A в момент завершения импульса должен последовать ниспадающий фронт паузы.

Начиная с метки `IntEnde`, новое вычисленное значение записывается в регистр сравнения A, и восстанавливаются флаги главной программы. Наконец, по команде `reti` вновь активизируется общее разрешение прерываний, и происходит возврат к главной программе.

Описание главной программы

При поступлении сигнала сброса регистр TCCR1B автоматически инициализируется значением $00, останавливая тем самым счетчик T/C1. По сбросу при

включении питания происходит переход по адресу $000 к части инициализации, обозначенной меткой `Initial`. После инициализации стека в первую очередь устанавливаются в лог. 0 все разряды выходов порта D. Затем вывод OC1A, которому соответствует разряд 5 порта D, с помощью записи в регистр направления передачи данных DDRD значения $20 определяется как выход.

Выходной сигнал должен начинаться с паузы, поэтому регистр сравнения A инициализируется константой `Pause`, а разряды COM1A1 и COM1A0 в регистре TCCR1A устанавливаются в лог. 1, чтобы при сравнении содержимого счетного регистра и регистра сравнения A в момент окончания паузы на выходе OC1A был установлен высокий уровень сигнала (начало импульса).

В следующей части программы в качестве входного такта T/C1 с помощью мультиплексора выбирается такт системной синхронизации **Ф**, деленный на восемь. Для этого в регистр TCCR1B записывается значение $02 (см. табл. 4.2), и T/C1 начинает счет. Для того чтобы разрешить прерывание от T/C1 по совпадению с регистром сравнения A, должен быть установлен в лог. 1 разряд OCIE1A в регистре TIMSK, а также флаг общего разрешения прерываний I в регистре состояния SREG.

Сама главная программа представляет собой только бесконечный цикл, поскольку в этом примере нас интересует только подпрограмма обработки прерывания.

Эмуляция выполнения программы в AVR-Studio показана на рис. 16.16.

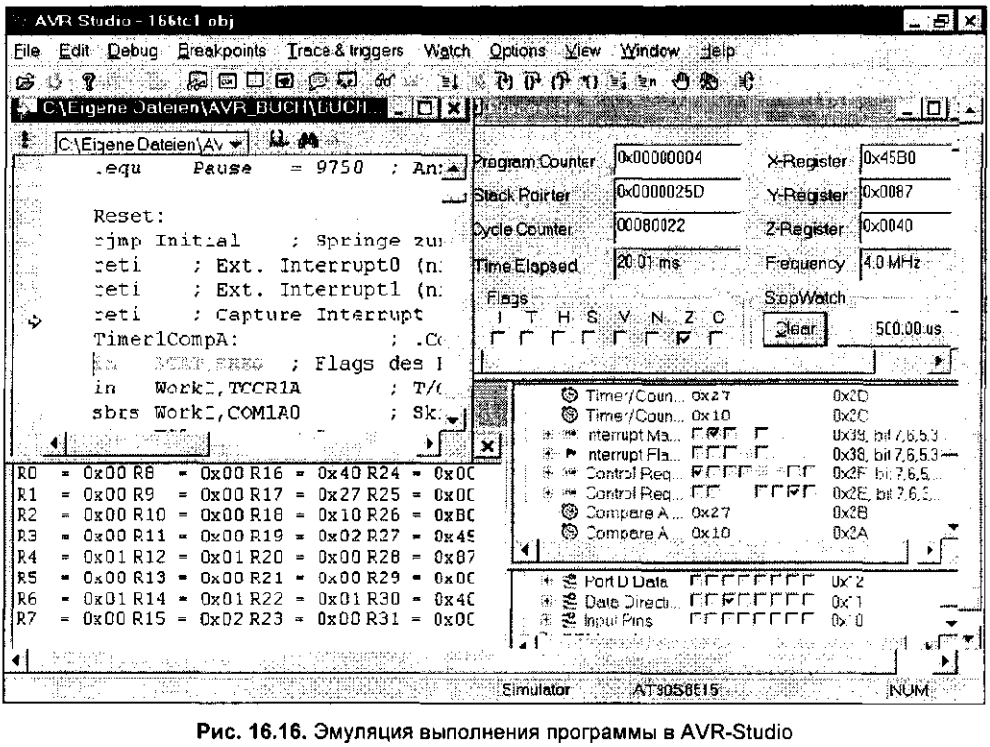


Рис. 16.16. Эмуляция выполнения программы в AVR-Studio

При первом входе в подпрограмму обработки прерывания выходной сигнал, в соответствии с инициализацией COM1A1 и COM1A0, переключается в лог. 1. В этот момент времени следует сброс секундомера в окне **Processor**. Рис. 16.16 соответствует второму входу в подпрограмму обработки прерывания. Импульс завершен (разряд PD5 опять установлен в лог. 0), и в поле **StopWatch** отображается его длительность: 500 мкс. В поле **Cycle Counter** отображается значение 80022, соответствующее количеству тактовых синхроимпульсов, прошедших с момента запуска программы. Поскольку 22 из 80022 импульсов ушло на инициализацию, период следования наших выходных сигналов составляет ровно 80000 тактов или 8000 / 4 МГц = 20 мс.

Рассмотренная программа была с помощью набора STK200 введена в микроконтроллер AT90S8515, и значения, наблюдаемые на выходе OC1A, в точности совпадали с полученными в AVR-Studio.

Трехканальный ЦАП с разрешением 10 разрядов

Рассмотрим альтернативный вариант формирования ШИМ-сигнала с помощью микроконтроллера AT90S1200, не оснащенного таймером/счетчиком T/C1 с возможностью ШИМ, а также с помощью остальных представителей семейства AVR, у которых T/C1 в том или ином случае применения используется для решения других задач. Если не требуется высокая скорость преобразования, то такой вариант прекрасно подходит для реализации простых, но точных ЦАП. С этой целью для формирования усредненного значения на ШИМ-выходе включают фильтр нижних частот (ФНЧ) Баттерворта (рис. 16.17) второго порядка с многопетлевой отрицательной обратной связью.

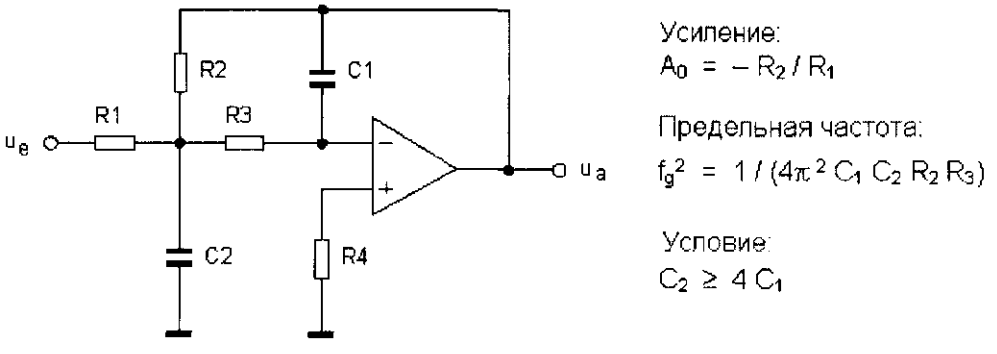


Рис. 16.17. Фильтр нижних частот второго порядка для формирования усредненного значения

Предельную частоту f_g ФНЧ выбирают значительно меньше часоты следования ШИМ-импульсов, чтобы подавить основную гармонику и гармоники высшего порядка, вносимые в прямоугольный сигнал, и получить только хорошо отфильтрованную составляющую постоянного напряжения. Для ФНЧ Баттерворта второго порядка входная частота f_e подавляется на $|A| / A_0 = -40$ дБ для $f_g = 0{,}1 \cdot f_e$ и на $|A| / A_0 = -80$ дБ для $f_g = 0{,}01 \cdot f_e$.

Однако выбирать слишком низкую предельную частоту также не следует, поскольку это излишне увеличит время установления колебаний в новое выходное значение. На практике выбирают $f_g = f_{ШИМ} / (10...1000)$.

Гармонический анализ сигнала $u_{(t)}$, представленного на рис. 16.18, дает следующий ряд Фурье:

$$u(t) = \hat{u} \cdot t_H / T + 2\,\hat{u} / \pi\,(\sin \omega\, t_H/2 \cdot \cos \omega t + {}^1/_2 \sin 2\omega\, t_H/2 \cdot \cos 2\omega t + {}^1/_3 \sin 3\omega\, t_H/2 \cdot \cos 3\omega t + {}^1/_4 \sin 4\omega\, t_H/2 \cdot \cos 4\omega t + \ldots)$$

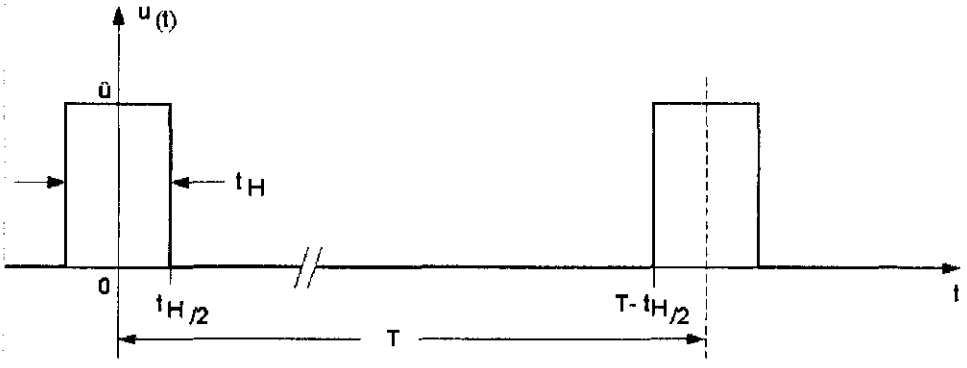


Рис. 16.18. ШИМ-сигнал при цифро-аналоговом преобразовании

Амплитуда основной и первой гармоники зависит от значения t_H (рис. 16.19). Максимальная амплитуда $a_{1\,max}$ основной гармоники достигается при $t_H = T/2$ и составляет около 0,64 û, а максимальная амплитуда $a_{2\,max}$ первой гармоники достигается при $t_H = T/4$ и $t_H = {}^3/_4\,T$ и составляет около 0,32 û.

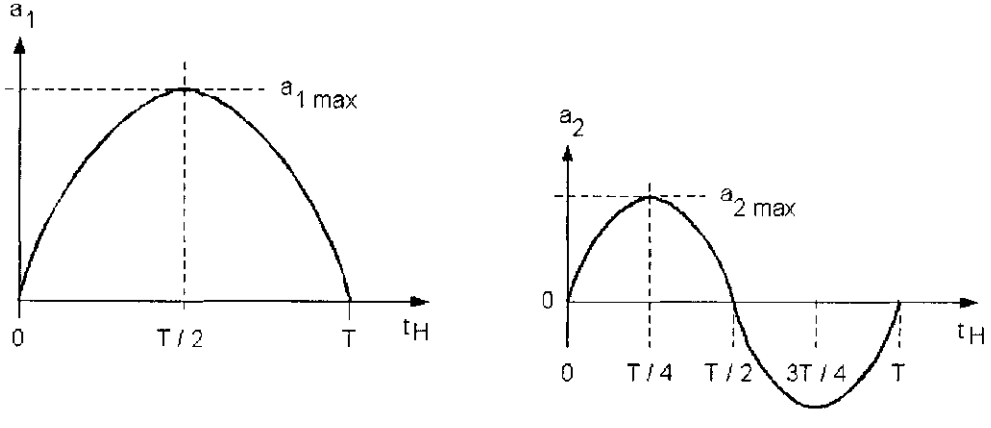


Рис. 16.19. Амплитуда a_1 основной гармоники (слева) и a_2 первой гармоники (справа) в зависимости от t_H

С помощью 10 бит можно представить диапазон чисел 0...1023 (000_h...$3FF_h$). Таким образом, младшему разряду в ШИМ-сигнале соответствует длительность импульса $t_H = T / 1024$.

При опорном напряжении –2,048 В на один младший разряд приходится 2 мВ, а диапазон выходных напряжений составляет 0...2,046 В. В результате вышеупомянутого разложения Фурье получаем для младшего разряда: составляющая постоянного напряжения $a_0 = \hat{u} / 1024 \approx 0{,}000977\ \hat{u}$.

Амплитуда первой гармонической составляющей (основной гармоники):

$a_1 = = (2\ \hat{u} / \pi) \cdot \sin \omega\ t_H/2 = (2\ \hat{u} / \pi) \cdot \sin 2\pi / T\ (T/2048) = (2\ \hat{u} / \pi) \cdot 0{,}00307 \approx \approx 0{,}00195\ \hat{u}$.

Амплитуда второй гармонической составляющей:

$a_2 = (2\ \hat{u} / \pi) \cdot ½ \sin 2\omega\ t_H/2 = (2\ \hat{u} / \pi) \cdot ½ \sin 4\pi / T\ (T/2048) = (2\ \hat{u} / \pi) \cdot 0{,}00307 \approx \approx 0{,}00195\ \hat{u}$;

...

Амплитуда двадцатой гармонической составляющей:

$a_{20} = (2\ \hat{u} / \pi) \cdot 1/20 \cdot \sin 20\omega\ t_H/2 = (2\ \hat{u} / \pi) \cdot 1/20 \cdot \sin 40\pi / T\ (T/2048) = = (2\ \hat{u} / \pi) \cdot 0{,}00307 \approx 0{,}00195\ \hat{u}$.

Таким образом, амплитуды отдельных гармонических составляющих, которые накладываются на составляющую постоянного напряжения a_0, для младшего разряда незначительны, но при этом практически не удаляются.

Амплитуда основной гармоники составляет $a_1 \approx 0{,}00195\ \hat{u}$ и, следовательно, почти в два раза больше младшего разряда. Однако, если частота ШИМ $f_{ШИМ}$ в десять раз больше предельной частоты f_g ФНЧ, то эта амплитуда подавляется на 40 дБ и в результате составит всего 1/50 часть младшего разряда.

Все остальные высшие гармоники младшего разряда подавляются еще сильнее, поэтому суммарная погрешность никогда не превышает ½ младшего разряда.

Иначе обстоит дело для более высоких амплитуд — особенно в случае максимальной амплитуды основной гармоники, возникающей при $t_H = T/2$ (как показано выше, она составляет 0,64 $\hat{u}$). При установленном ранее подавлении для младшего разряда 40 дБ она подавляется приблизительно на 0,0064 $\hat{u}$ и, следовательно, всегда в 6,5 раз больше младшего разряда. Таким образом, очевидно, что для достижения погрешности < ½ младшего разряда требуется меньшее значение предельной частоты ФНЧ.

С помощью использованного здесь метода широтно-импульсной модуляции формируются не фазы высокого и низкого уровня длительностью полпериода, как показано на рис. 16.20 внизу, а импульсы, распределенные равномерно на протяжении периода ШИМ-сигнала (рис. 16.20 вверху).

Оба ШИМ-сигнала, показанные на рис. 16.20, имеют одинаковую составляющую постоянного напряжения, однако частота следования импульсов для верхнего случая в четыре раза выше (в случае трехразрядной ШИМ). Естественно, для $t_H = T/2$ в случае 10-разрядной широтно-импульсной модуляции, рассматриваемой в нашем примере, она еще больше и в 512 раз превышает частоту ШИМ.

Еще один источник ошибок для ЦАП — цифровой выходной сигнал порта микроконтроллера AVR. Как было показано в главе 4, среднее арифметическое значение U_{AV} прямоугольного сигнала составляет $U_{AV} = (U_H \times t_H + U_L \times t_L) / T$, где U_H — напряжение сигнала высокого уровня, U_L — напряжение сигнала низкого уровня, а t_H и t_L — длительности соответствующих фаз (рис. 16.21).

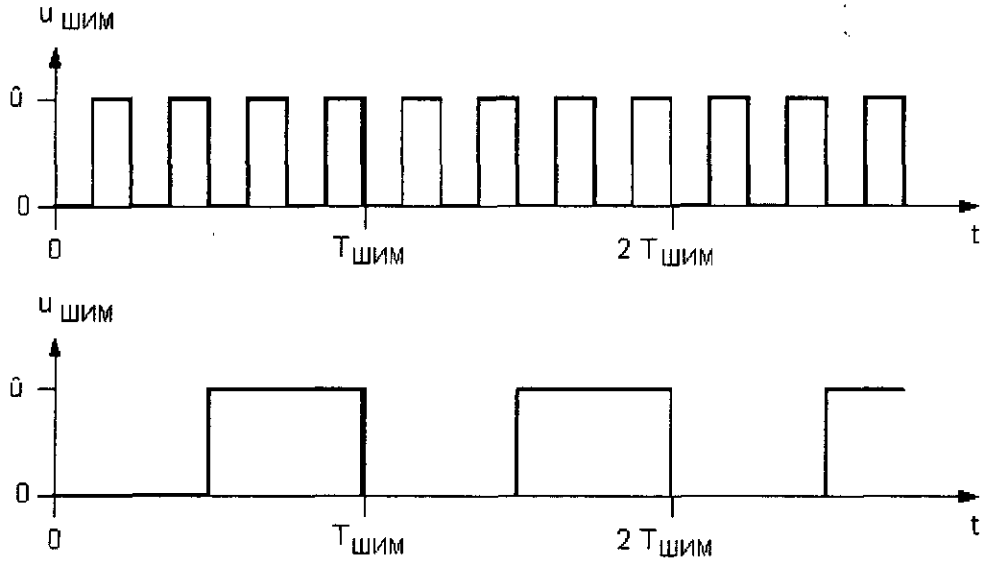


Рис. 16.20. Два различных выходных ШИМ-сигнала с одинаковой составляющей постоянного напряжения

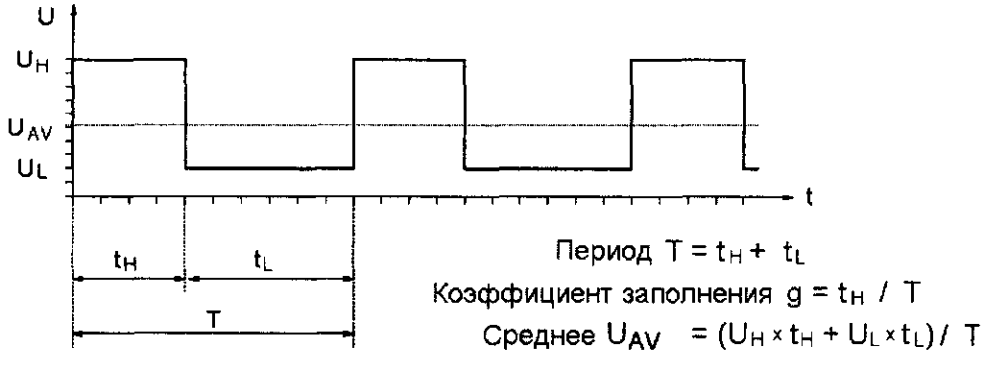


Рис. 16.21. Определение периода T, коэффициента заполнения g и среднего арифметического U_{AV} для прямоугольных импульсов напряжения U

Если бы в качестве входного напряжения ФНЧ использовался напрямую цифровой выходной сигнал, то по причине погрешности цифрового напряжения был бы получен ЦАП очень малой точности. На выходе микроконтроллера AVR не бывает ни полного значения рабочего напряжения U_H, ни значения $U_L = 0$ В. Поскольку выходное напряжение зависит от свойств материала, температуры и нагрузки, его значение можно сместить в относительно более высокую область. Для этого можно воспользоваться схемой, показанной на рис. 16.22.

Усиление этой схемы ФНЧ составляет $A_0 = - R_2 / R_1$. Оно отрицательное, поскольку основано на инвертирующей принципиальной схеме операционного усилителя. По этой причине, для того, чтобы получить положительное напряжение на выходе ЦАП, с помощью КМОП-ключа организовано переключение между уровнем 0 В и отрицательным опорным напряжением.

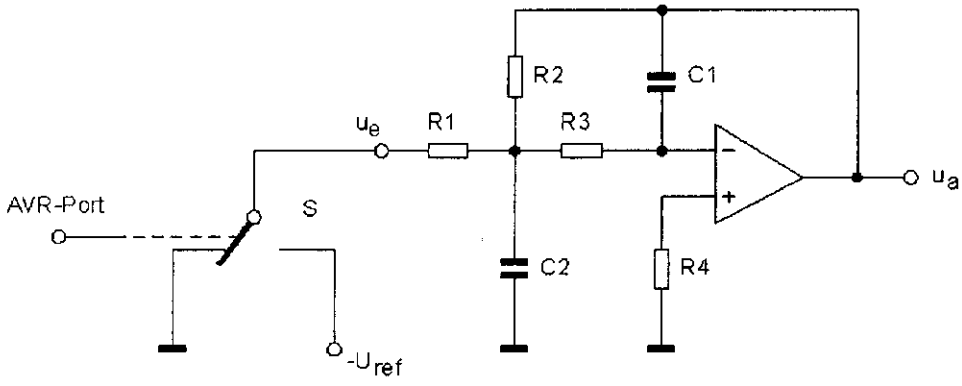


Рис. 16.22. ФНЧ второго порядка с операционным усилителем, используемый в качестве ЦАП в режиме ШИМ

Использование операционного усилителя делает схему совершенно независимой от нагрузки. Внутреннее сопротивление КМОП-ключа, которое обычно составляет 60...100 Ом, соединено последовательно с резистором R_1 и также принимает участие в усилении. Тем не менее, при данном соотношении сопротивлений R_1 и R_2 им можно пренебречь.

Общая схема трехканального ЦАП показана на рис. 16.23.

Опорное напряжение получают с помощью опорного диода LM336. С помощью триммера R_7 сопротивлением 2,7 кОм можно калибровать выходное напряжение источника опорного напряжения в некоторой области для компенсации сопротивления ключа и допуска опорного напряжения. Сопротивление R_9 служит для компенсации тока смещения и должно соответствовать значению параллельного включения R_6 и $(R_7 + R_8)$.

Все четыре операционных усилителя $V_{1a}...V_{1d}$ типа TL084 заключены в одном корпусе DIP14. Их положительное питающее напряжение должно составлять $V_{CC+} = +5$ В, а отрицательное — $V_{CC-} = -5$ В. Питающее напряжение шунтируется непосредственно на выводе корпуса V_{CC} (на рис. 16.23 не показан) обычным способом с помощью конденсатора емкостью 100 нФ.

КМОП-ключи $S_{1a}...S_{1c}$ типа 74HC4053 заключены в одном корпусе DIP16. Их положительное питающее напряжение должно составлять $V_{CC} = +5$ В, а отрицательное — $V_{EE} = -5$ В. Питающее напряжение шунтируется непосредственно на выводе корпуса V_{CC} (на рис. 16.23 не показан) обычным способом с помощью конденсатора емкостью 100 нФ.

Для трех ФНС должны использоваться высококачественные пленочные конденсаторы. Значения сопротивлений $R_1...R_3$ и емкостей $C_1...C_2$ для предельных частот $f_g = 2$ Гц и $f_g = 20$ Гц представлены в табл. 16.3.

Таблица 16.3. Сопротивления $R_1...R_3$ и емкости $C_1...C_2$ фильтра для предельных частот $f_g = 2$ Гц и $f_g = 20$ Гц при усилении $A_0 = -1$

f_g	R_1	R_2	R_3	C_1	C_2
2 Гц	130 кОм	130 кОм	510 кОм	100 нФ	1 мкФ
20 Гц	59 кОм	59 кОм	233 кОм	22 нФ	220 нФ

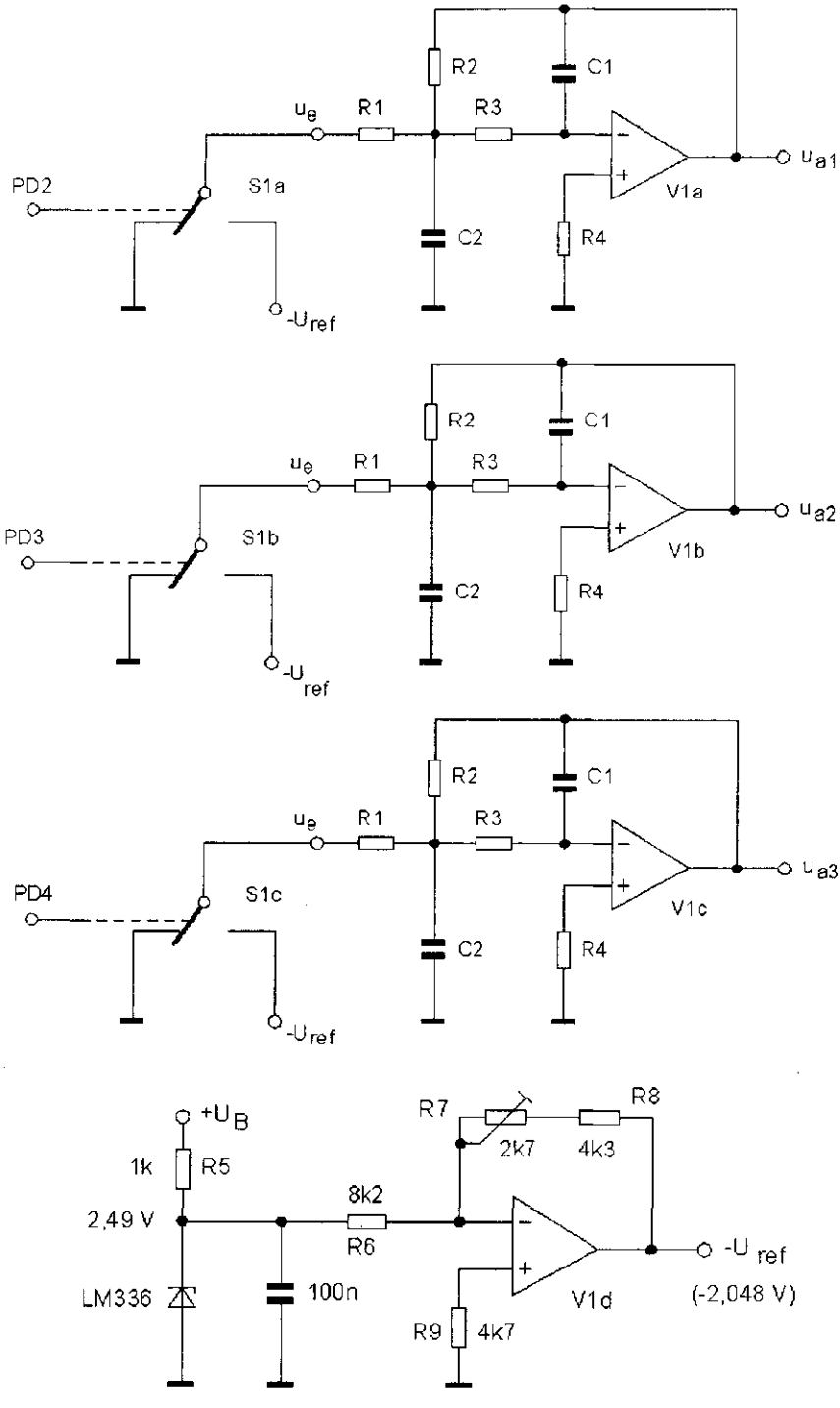


Рис. 16.23. Общая схема трехканального ЦАП

Для других предельных частот при заданном значении емкостей C_1 и C_2 сопротивления фильтра $R_1 \ldots R_3$ рассчитываются по следующим выражениям:

$$R_2 = \frac{a_1 C_2 - \sqrt{a_1^2 C_2^2 - 4 C_1 C_2 b_1 (1 - A_0)}}{4\pi f_g C_1 C_2}; \qquad R_1 = \frac{R_2}{-A_0}; \qquad R_3 = \frac{b_1}{4\pi^2 f_g^2 C_1 C_2 R_2}.$$

где A_0 — требуемое усиление (отрицательное вследствие инвертирующей принципиальной схемы усилителя); a_1 и b_1 — коэффициенты фильтра. В случае фильтра Баттерворта $a_1 = 1{,}4142$, $b_1 = 1$ (коэффициент a_1 не имеет никакого отношения к значению амплитуды a_1 из представленного выше разложения Фурье; совпадение обозначений — чисто случайное).

При выборе конденсаторов фильтра должно выполняться следующее условие:

$$\frac{C_2}{C_1} \geq \frac{4 b_1 (1 - A_0)}{a_1^2}.$$

Для упрощения схемы, представленной на рис. 16.23, в ней отказались от компенсации напряжения смещения операционного усилителя, тем не менее этим напряжением не следует пренебрегать, поскольку его порядок может соответствовать младшему разряду. Для решения этой проблемы можно воспользоваться усилителем в корпусе, снабженном выводами компенсации напряжения смещения. Еще одним интересным решением является программная калибровка ЦАП, при которой в виде числовых значений во внутренней памяти EEPROM сохраняется значение коррекции для нулевой точки и усиления. В этом случае, перед тем как будут выведены значения, с помощью программы корректируются ошибки усиления и нулевой точки, возникающие в схеме.

Рассматриваемая схема с целью тестирования была реализована на печатной плате и подключена к порту D модуля STK200. При тестировании по нажатию кнопки PD0 последовательно получают четыре различных значения напряжения на выходах трех ЦАП. О нажатии кнопки сигнализирует переключение светодиода PB7 модуля STK200.

Выходному напряжению U_{a1} соответствуют шестнадцатеричные значения \$0FF, \$1FF, \$2FF, \$3FF, \$0FF и т.д. Выходному напряжению U_{a2} соответствуют шестнадцатеричные значения \$001, \$301, \$201, \$101, \$001 и т.д. Выходному напряжению U_{a3} соответствуют шестнадцатеричные значения \$001, \$201, \$001, \$201, \$001 и т.д. Программа, реализующая эту функцию, представлена ниже.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\167DA.asm`

```
;******** 10-разрядный ЦАП ***********************************************
;*
;* Три ШИМ-счетчика в подпрограмме обработки прерывания каждые 5 мкс
;* увеличиваются на соответствующее выходное значение. Разряд переполнения
;* выдается как ШИМ-сигнал.
;*
;* Тактовая частота микроконтроллеров AVR: 8 МГц.
;*
;**************************************************************************
.nolist
.include "1200def.inc"
```

```
.list

.def    ac1L = r9       ; Младший байт текущего состояния счетчика канала 1
.def    ac2L = r10      ; Младший байт текущего состояния счетчика канала 2
.def    ac3L = r11      ; Младший байт текущего состояния счетчика канала 3
.def    rl1L = r12      ; Младший байт значения перезагрузки канала 1
.def    rl2L = r13      ; Младший байт значения перезагрузки канала 2
.def    rl3L = r14      ; Младший байт значения перезагрузки канала 3
.def    STAT = r15      ; Буфер для хранения флагов состояния главной программы
.def    WorkI = r16     ; Рабочий регистр для прерывания от таймера
.def    WorkH = r17     ; Рабочий регистр для главной программы
.def    ac1H = r18      ; Старший байт текущего состояния счетчика канала 1
.def    ac2H = r19      ; Старший байт текущего состояния счетчика канала 2
.def    ac3H = r20      ; Старший байт текущего состояния счетчика канала 3
.def    rl1H = r21      ; Старший байт значения перезагрузки канала 1
.def    rl2H = r22      ; Старший байт значения перезагрузки канала 2
.def    rl3H = r23      ; Старший байт значения перезагрузки канала 3
.def    tim1 = r24      ; Счетчик внутреннего цикла
.def    tim2 = r25      ; Счетчик внешнего цикла

.equ    OFL = 2         ; Переполнение при сложении в разряде 2
.equ    PWM1 = PD2      ; Разряд 2 порта D - выход для канала 1
.equ    PWM2 = PD3      ; Разряд 3 порта D - выход для канала 2
.equ    PWM3 = PD4      ; Разряд 4 порта D - выход для канала 3
.equ    Time = 40       ; Число тактовых импульсов до следующего прерывания
                        ;   40 импульсов при f = 8 МГц -> 5 мкс
Reset:
000000 c020    rjmp Initial          ; Переход к части инициализации
000001 9518    reti                  ; Внешнее прерывание 0 (не используется)
Timer0_Int:                          ; Прерывание от таймера 0 (для AT90S1200)
000002 b6ff    in STAT,SREG          ; Сохраняем флаги главной программы
000003 b702    in WorkI,TCNT0        ; Извлекаем текущее состояние счетчика
000004 5205    subi WorkI,Time-3     ; Устанавливаем таймер: длительность
                                     ;   интервала минус время выполнения
000005 bf02    out TCNT0,WorkI       ;   трех команд
000006 959a    dec tim2              ; Декрементируем счетчик внутреннего цикла
000007 b302    in WorkI,PortD        ; Извлекаем текущие ШИМ-состояния
000008 0c9c    add ac1L,rl1L         ; Прибавляем младший байт значения
                                     ; перезагрузки
000009 1f25    adc ac1H,rl1H         ; Прибавляем старший байт значения
                                     ; перезагрузки
00000a fb22    bst ac1H,OFL          ; Переполнение (разряд 2 ) - в флаг T
00000b f902    bld WorkI,PWM1        ; Переполнение на ШИМ-выходе канала 1
00000c 7023    andi ac1H,$03         ; Обнуляем разряды 2..7
00000d 0cad    add ac2L,rl2L         ; Прибавляем младший байт значения
                                     ; перезагрузки
00000e 1f36    adc ac2H,rl2H         ; Прибавляем старший байт значения
                                     ; перезагрузки
00000f fb32    bst ac2H,OFL          ; Переполнение (разряд 2 ) - в флаг T
000010 f903    bld WorkI,PWM2        ; Переполнение на ШИМ-выходе канала 2
000011 7033    andi ac2H,$03         ; Обнуляем разряды 2..7
000012 0cbe    add ac3L,rl3L         ; Прибавляем младший байт значения
                                     ; перезагрузки
000013 1f47    adc ac3H,rl3H         ; Прибавляем старший байт значения
                                     ; перезагрузки
000014 fb42    bst ac3H,OFL          ; Переполнение (разряд 2 ) - в флаг T
```

```
000015 f904    bld WorkI,PWM3          ; Переполнение на ШИМ-выходе канала 3
000016 7043    andi ac3H,$03           ; Обнуляем разряды 2..7
000017 bb02    out PortD,WorkI         ; Выводим новые ШИМ-состояния в порт D
000018 beff    out SREG,STAT           ; Восстанавливаем старые флаги
000019 9518    reti                    ; Выход из прерывания от таймера 0

Wait20ms:                              ; Каждые 5 мкс при прерывании от T/C0
00001a e180    ldi tim1,16             ; Загружаем счетчик внешнего цикла
L1:
00001b ef9a    ldi tim2,250            ; Загружаем счетчик внутреннего цикла
L2:
00001c 2399    tst tim2                ; Счетчик = 0 ?
00001d f7f1    brne L2                 ; Если нет, то ожидаем дальше
00001e 958a    dec tim1
00001f f7d9    brne L1
000020 9508    ret

Taste:                                 ; Ожидаем нажатия кнопки
000021 9980    sbic PinD,0             ; Пропускаем следующую команду, если
                                       ; нажата кнопка 0
000022 cffe    rjmp Taste              ; Ожидаем дальше
000023 dff6    rcall wait20ms          ; Пауза для учета дребезга контакта
000024 9980    sbic PinD,0             ; Пропускаем следующую команду, если
                                       ; кнопка все еще нажата
000025 cffb    rjmp Taste              ; Ожидаем дальше
T1:
000026 9b80    sbis PinD,0             ; Пропускаем следующую команду, если
                                       ; кнопка 0 отпущена
000027 cffe    rjmp T1                 ; Ожидаем дальше
000028 dff1    rcall wait20ms          ; Пауза для учета дребезга контакта
000029 9b80    sbis PinD,0             ; Пропускаем следующую команду, если
                                       ; кнопка все еще отпущена
00002a cffb    rjmp T1                 ; Ожидаем дальше
00002b 9bbf    sbis DDRB,7             ; Пропускаем следующую команду, если
                                       ; светодиод включен
00002c c002    rjmp T2
00002d 98bf    cbi DDRB,7              ; Включаем светодиод
00002e c001    rjmp T3
T2:
00002f 9abf    sbi DDRB,7              ; Выключаем светодиод
T3:
000030 9508    ret

Initial:                               ; Инициализация регистров ввода/вывода
000031 e010    ldi WorkH,$00           ; Устанавливаем все разряды в лог. 0
000032 bb18    out PortB,WorkH         ; Вывод в порт B
000033 bb17    out DDRB,WorkH          ; Входы - в высокоомное состояние
000034 bb12    out PortD,WorkH         ; Вывод в порт D
000035 e11c    ldi WorkH,$1C           ; Загружаем направление передачи данных
000036 bb11    ut DDRD,WorkH           ; Разряды 2..4 - выходы, остальные - входы
000037 2722    clr ac1H
000038 2499    clr ac1L
000039 2733    clr ac2H
00003a 24aa    clr ac2L
00003b 2744    clr ac3H
00003c 24bb    clr ac3L                ; Сбрасываем текущие состояния счетчиков
```

```
00003d 24cc    clr rl1L
00003e 2755    clr rl1H
00003f 24dd    clr rl2L
000040 2766    clr rl2H
000041 24ee    clr rl3L
000042 2777    clr rl3H              ; Сбрасываем значения перезагрузки
000043 94ca    dec rl1L              ; Декрементируем на 1 ($FF)
000044 94d3    inc rl2L              ; Инкрементируем на 1 ($01)
000045 94e3    inc rl3L              ; Инкрементируем на 1 ($01)
000046 ed18    ldi WorkH,-Time       ; Загружаем интервал (отрицательный,
                                     ;   потому что счетчик - суммирующий)
000047 bf12    out TCNT0,WorkH       ; Инициализируем таймер, T = 5 мкс (8 МГц)
000048 e011    ldi WorkH,1
000049 bf13    out TCCR0,WorkH       ; Запускаем таймер, такт = такт системной
                                     ; синхронизации
00004a e012    ldi WorkH,2
00004b bf19    out TIMSK,WorkH       ; Разрешаем прерывание от таймера
00004c 9478    sei                   ; Общее разрешение прерываний (разряд I)

Haupt:
00004d dfd3    rcall Taste           ; По нажатию кнопки выводим след. значение
00004e 94f8    cli                   ; Запрет прерываний
00004f 5f5f    subi rl1H,-$01        ; Значение перезагрузки 1 =
                                     ;   Значение перезагрузки 1 + $0100
000050 7053    andi rl1H,$03         ; Обнуляем разряды 2..7
000051 5061    subi rl2H,$01         ; Значение перезагрузки 2 =
                                     ;   Значение перезагрузки 2 - $0100
000052 7063    andi rl2H,$03         ; Обнуляем разряды 2..7
000053 5f7e    subi rl3H,-$02        ; Значение перезагрузки 3 =
                                     ;   Значение перезагрузки 3 + $0200
000054 7073    andi rl3H,$03         ; Обнуляем разряды 2..7
000055 9478    sei                   ; Опять разрешаем прерывание
000056 cff6    rjmp Haupt            ; Продолжаем цикл
```

Описание подпрограммы обработки прерывания от T/C0

Переполнение счетчика из $FF в $00 устанавливает флаг TOV0, и выполнение программы продолжается с метки `Timer_Int` по адресу $002 (подпрограмма обработки прерывания), после чего флаг TOV0 опять автоматически устанавливается. Первая команда подпрограммы сохраняет регистр состояния, который в данный момент содержит флаги главной программы.

Учитывая время ответа на запрос на прерывание (четыре тактовых импульса), после входа в подпрограмму обработки прерывания счетчик TCNT0 содержит значение от $04 до $06 в зависимости от того, какой длины команда выполнялась в момент возникновения переполнения: занимающая один, два или три (`ret`) тактовых цикла. Для того чтобы загрузить точное значение интервала, из текущего содержимого регистра TCNT0, которое прежде было записано в рабочий регистр `WorkI`, вычитается требуемое выходное значение длительности интервала `Time`.

Для того чтобы учесть время выполнения трех команд по адресам с $002 по $004, интервал, представленный константой `Time`, перед тем как он будет вычтен из текущего состояния счетчика `WorkI` и загружен в счетный регистр, дополни-

тельно уменьшается на 3. Счетчик цикла `tim2`, который используется в подпрограмме `Wait20ms`, при каждом прерывании уменьшается на 1.

Затем, прибавив к состоянию счетчиков соответствующие значения перезагрузки, получаем три новые ШИМ-состояния, и выдаем в порт D разряды переполнения. Перед возвратом в главную программу восстанавливаются флаги состояния, и по команде `reti` вновь устанавливается общее разрешение прерываний.

Время выполнения подпрограммы обработки прерывания от таймера 0

На выполнение 23 команд (без `reti`) уходит 23 тактовых цикла, к которым прибавляется время на переход к подпрограмме и время на выполнение команды `reti` (по четыре такта). Таким образом, в общей сложности для выполнения подпрограммы обработки прерывания требуется 31 тактовый импульс. При частоте системной синхронизации **Ф** = 8 МГц это соответствует времени 3,875 мкс.

Поскольку подпрограмма обработки прерывания вызывается только каждые 5 мкс, в распоряжении остальной части программы остается 22,5% системной пропускной способности. Если функция ЦАП должна использоваться в какой-либо другой программе, в которой на выполнение других функций требуется больше времени, можно увеличить период вызовов прерывания (разумеется, с учетом предельной частоты ШИМ).

Подпрограмма `Wait20ms` реализует задержку на 20 мс. Для этого используется счетчик цикла `tim2`, который декрементируется через каждые 5 мкс в подпрограмме обработки прерывания от T/C0. Внешний цикл выполняется 16 раз, за которые `tim2` при такте 5 мкс уменьшается от 250 до 0. Таким образом, общая длительность выполнения `Wait20ms` составляет 16 x 250 x 5 мкс = 20 мс.

Подпрограмма `Taste` ожидает нажатия кнопки PD0 для последовательного переключения выходных значений ЦАП. Когда на выводе PD0 распознается низкий уровень сигнала (нажатие кнопки), начинает выполняться подпрограмма `Wait20ms`. Если по истечении 20 мс на выводе PD0 все так же распознается сигнал низкого уровня, то это указывает на то, что кнопка действительно нажата. В противном случае, ситуация расценивается просто как дребезг контакта, и кнопка считается не нажатой (рис. 16.24).

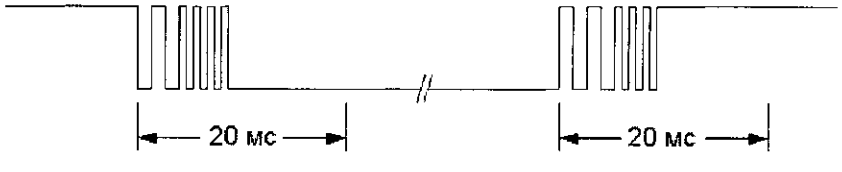


Рис. 16.24. Учет дребезга контакта кнопки на выводе PD0 с помощью программной задержки

В том случае, если кнопка нажата, ожидается момент, конда она будет отпущена. Как только на выводе PD0 появится высокий уровень сигнала (кнопка отпущена), начинает выполняться подпрограмма `Wait20ms`. Если по истечении 20 мс на выводе PD0 все так же распознается сигнал высокого уровня, то это ука-

зывает на то, что кнопка действительно отпущена. В противном случае, ситуация расценивается просто как дребезг контакта, и кнопка считается нажатой.

Для оптической сигнализации о нажатии кнопки по окончании подпрограммы `Taste` включается светодиод PB7.

Описание главной программы

Когда поступает сигнал сброса, регистр TCCR0 автоматически инициализируется значением $00, останавливая тем самым T/C0. По сбросу при включении питания выполняется переход по адресу $000 к части инициализации, обозначенной меткой `Initial`. Порт B требуется только для сигнализации о нажатии кнопки с помощью светодиода PB7, и потому конфигурируется как тристабильный вход, поскольку вывод светодиода PB7 реализован по технологии открытого коллектора через переключение регистра направления передачи данных DDRB (см. главу 10).

Все выходные разряды порта D устанавливаются в лог. 0, и разряды 2...4 определяются как выходы посредством записи лог. 1 в регистр направления передачи данных DDRD. Тем самым вывод PD0 определяется как вход для подключения кнопки. В модуле STK200 для работы светодиода по выводу PB7 и кнопки по выводу PD0 должна быть установлена соответствующая перемычка.

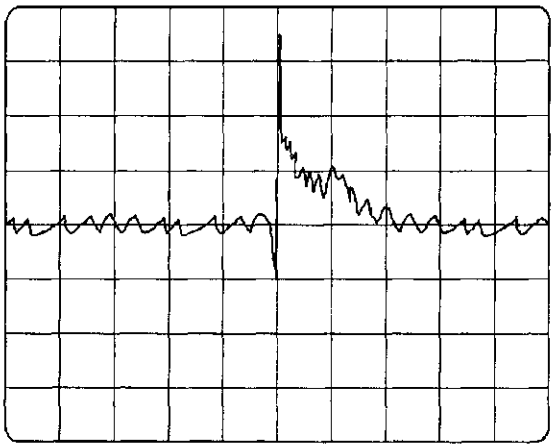
Текущее состояние счетчиков инициализируется с помощью $000, а значения перезагрузки — с помощью регистровых пар `rl1H:rl1L` = $00FF, `rl2H:rl2L` = $0001 и `rl3H:rl3L` = $0001. После этого в счетный регистр TCNT0 загружается отрицательное значение длительности интервала. Представленная контстантой `Time` длительность интервала должна быть отрицательной, поскольку T/C0 работает как суммирующий счетчик.

В следующей части программы с помощью мультиплексора в качестве входного такта для T/C0 выбирается непосредственно такт системной синхронизации **Ф**. Затем в регистр TCCR0 записывается значение $01 (см. табл. 4.2), и T/C0 начинает счет. Для того чтобы разрешить прерывание от T/C0, устанавливается разряд TOIE0 в регистре TIMSK, а также флаг общего разрешения прерываний I в регистре состояния SREG.

Сама главная программа представляет собой бесконечный цикл в ожидании нажатия кнопки на выводе PD0. Как только это происходит, вычисляются следующие значения перезагрузки в упомянутой выше последовательности. На время этих вычислений прерывание от T/C0 запрещено, поскольку в противном случае могут быть получены некорректные значения счетчика.

Измерения

В лабораторных условиях была замерена пульсация выходного напряжения данной схемы. Во всех случаях эффективное значение находилось в пределах одного милливольта. При этом при переключении на цифровых выходах были отмечены броски напряжения, показанные на рис. 16.25, однако их можно легко устранить с помощью компактного монтажа и дополнительных мер гашения.



По горизонтали: 1 мкс/деление. По вертикали: 2 мВ/деление

Рис. 16.25. Пульсация выходного напряжения

Четырехканальный АЦП с двойным интегрированием и разрешением 11 разрядов

Реализуем с помощью таймера/счетчика T/C1 АЦП с двойным интегрированием, показанный на рис. 16.26. Инвертирующий вход AIN1 интегрированного аналогового компаратора микроконтроллера AVR подключен на "землю", а на неинвертирующий вход AIN0 подано выходное напряжение интегратора V1. Диод D2 вместе с резистором R6 защищает вход AIN0 от слишком высоких уровней отрицательного входного напряжения.

Преобразование выполняется в три этапа. На первом этапе устанавливается в нуль выходное напряжение интегратора, состоящего из элементов V1, R3 и C1. Если текущее напряжение U_{AIN0} больше, чем U_{AIN1}, то на вход интегратора с помощью коммутатора S1 подается напряжение U_{ref+}. Если же U_{AIN0} меньше, чем U_{AIN1}, то на интегратор подается напряжение U_{ref-}. В это время интегратор в нулевой точке постоянно сравнивает напряжения U_{AIN0} и U_{AIN1}, и в тот момент, когда выход компаратора меняет состояние, достигается нулевая точка интегрирования, и начинается второй этап аналого-цифрового преобразования: измеряемое напряжение U_{x1} или U_{x2} (U_{R1} или U_{R2} — зависит от перемычки J1) с помощью коммутатора S1 подается на вход интегратора на время T_X. Если в качестве входного такта для T/C1 выбран деленный на 64 такт системной синхронизации Φ = 4 МГц, то подсчету 4096 тактовых импульсов соответствует время T_X = 65,536 мс.

На рис. 16.27 проиллюстрирован следующий процесс. В начале выходное напряжение интегратора U_I имеете некоторое произвольное значение. В данном примере оно отрицательное, поэтому на этапе S1 интегрируется опорное напряжение U_{ref-} до тех пор, пока напряжение U_I не достигнет порогового уровня 0 В. В этот момент начинается этап S2, на протяжении которого в течение времени T_X интегрируется положительное напряжение U_{x0}.

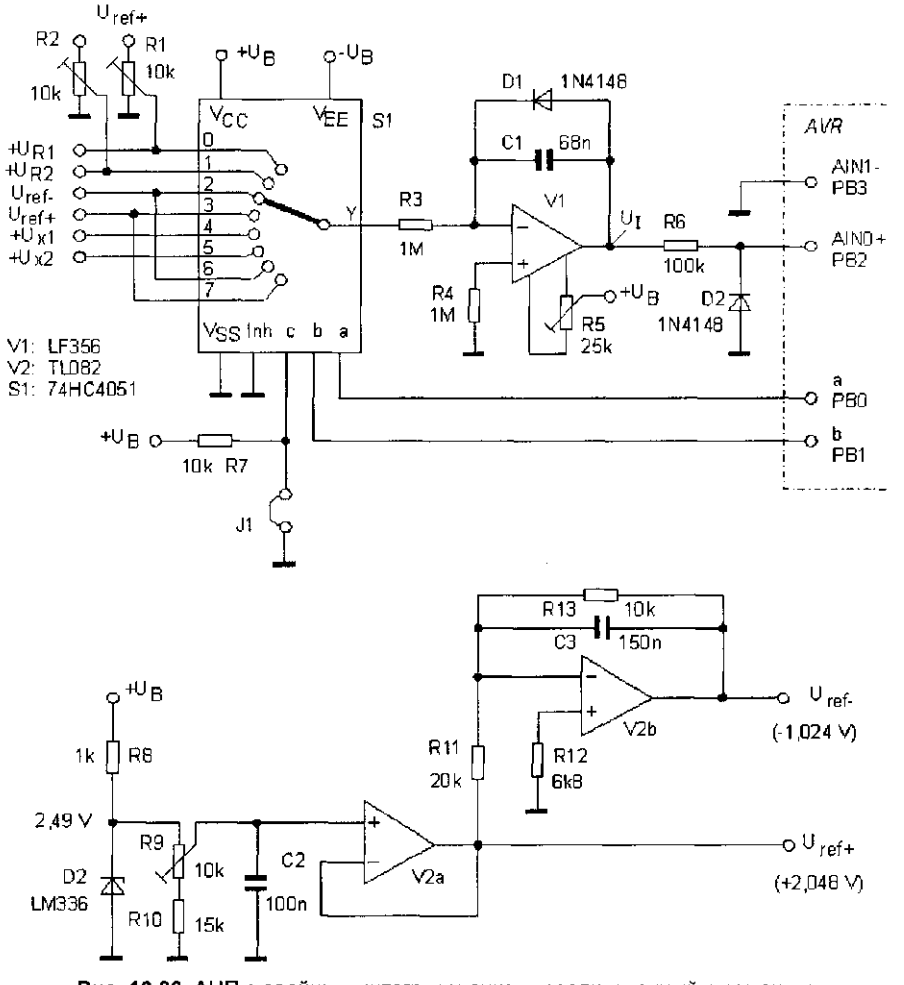


Рис. 16.26. АЦП с двойным интегрированием, реализованный с помощью аналогового компаратора и таймера/счетчика T/C1

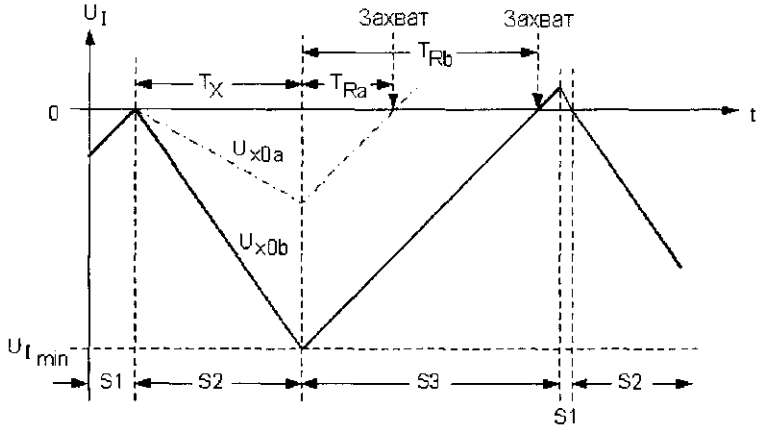


Рис. 16.27. Выходное напряжение U_I интегратора для двух различных значений (a и b) входного напряжения U_{x0}

На рис. 16.27 линия U_I показана для двух различных значений: a и b. По истечении времени интегрирования T_X начинается этап S3, в течение которого содержимое счетчика T/C1 увеличивается, начиная с исходного значения 0, и интегрируется опорное напряжение U_{ref-} до тех пор, пока вновь не будет пересечена линия нуля для напряжения U_I.

На этапе S3 компаратор микроконтроллера AVR сравнивает напряжение на неинвертирующем входе AIN0 с напряжением на инвертирующем входе AIN1. Как только U_{AIN0} становится больше U_{AIN1}, состояние на выходе компаратора меняется с лог. 0 на лог. 1. По этому нарастающему фронту текущее содержимое регистра TCNT1 переносится в регистр завата на входе.

После возникновения захвата процесс интегрирования продолжается еще некоторое время, однако, благодаря диоду D1, уровень выходного напряжения V1 ограничивается. Затем вновь следует этап S1, на протяжении которого достигается корректное значение напряжения $U_I = 0$ В для начала очередного измерения.

За время интегрирования T_X достигается следующее значение выходного напряжения интегратора U_I:

$$U_{I(Tx)} = -U_x \cdot T_X / \tau, \text{ где } U_x \geq 0 \text{ В}; \tau = R_3C_1; T_X = 65{,}536 \text{ мс}.$$

Отрицательное значение $U_{I(Tx)}$, достигаемое при $U_{x\ max} = 2{,}047$ В, составляет $U_{I(Tx)\ min} = -1{,}973$ В.

Время интегрирования T_R, необходимое для достижения порога захвата и переноса состояния счетчика в регистр захвата, прямо пропорционально измеряемому входному напряжению U_x:

$$T_R = t \cdot (-U_{I(Tx)}) / (-U_{ref-}) \text{ при } \tau = R_3C_1; U_{ref-} = -1{,}024 \text{ В и } U_{I(Tx)} = -U_x \cdot T_X / \tau;$$

$$T_R = T_X \cdot U_x / |U_{ref-}| \text{ при } U_x \geq 0 \text{ и } U_{ref-} < 0.$$

Отсюда получаем для напряжения U_x:

$$U_x = |U_{ref-}| \cdot T_R / T_X.$$

Таким образом, точность результата зависит только от допуска для U_{ref-} и соотношения T_R / T_X, и никоим образом — от пассивных элементов R_3 и C_1. Поскольку T_X и T_R получают с помощью счетчика T/C1 от одной и той же стабилизированной тактовой частоты, их соотношение в продолжение относительно малого времени измерения можно рассматривать как константу, и потому решающее значение для точности измерений имеет исключительно точность U_{ref-}.

Требуемые опорные напряжения U_{ref-} и U_{ref+} получают с помощью опорного диода LM336 с температурной компенсацией и двух операционных усилителей V2a и V2b. Отрицательное опорное напряжение U_{ref-} устанавливается в значение -1,024 В с помощью триммера R_9 сопротивлением 10 кОм. Резистор R_{12} предназначен для компенсации тока смещения и имеет сопротивление параллельного включения резисторов R_{11} и R_{13}.

Операционный усилитель V1 имеет такие же высокоомные входы на полевых транзисторах, что и операционные усилители V2a и V2b. Его напряжение смещения можно установить с помощью триммера R_5. Усилители V2a и V2b типа TL082

заключены в общий корпус DIP8. Их положительное питающее напряжение V_{CC+} = +5 В, а отрицательное — V_{CC-} = –5 В.

Используется КМОП-коммутатор S1 типа 74HC4051 в корпуске DIP16. Его положительное питающее напряжение V_{CC} = + 5 В, а отрицательное — V_{EE} = –5 В.

Питающие напряжения интегральной схемы обычно шунтированы на вывод корпуса V_{CC} с помощью конденсатора емкостью 100 нФ (с целью упрощения на рис. 16.16 не показано).

В качестве тактовой частоты для микроконтроллера AT90S8515 выбираем значение 4 МГц, которое для получения входного такта T/C1 делится на 64. Содержимое счетчика T/C1 инкрементируется каждые 16 мкс, и, таким образом, времени интегрирования T_X = 65,536 мс соответствует 4096 тактовых импульсов.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\168AD.asm`

```
;**** Реализация АЦП с двойным интегрированием с помощью T/C1***********
;*
;* Микроконтроллер AT90S8515 преобразует аналоговое входное напряжение
;* в шестнадцатеричное число.
;* Разрешение - 11 разрядов ($000..$7FF).
;* Флаг T определяет, какое напряжение измеряется: Ux0 (T=0) или Ux1 (T=1).
;*
;* Тактовая частота микроконтроллера AVR: 4 МГц (стандарт STK200)
;*
;***********************************************************************
.nolist
.include "8515def.inc"
.list

.def    tmp1 = r16      ; Рабочий регистр 1
.def    tmp2 = r17      ; Рабочий регистр 2
.def    prm1 = r18      ; Передаваемый параметр 1
.def    tim1 = r19      ; Счетчик цикла 1
.def    tim2 = r20      ; Счетчик цикла 2
.def    Cnt = r21       ; Счетчик строк
.def    ErgL = r22      ; Рабочий регистр
.def    ErgH = r23      ; Рабочий регистр
.def    WorkL = r24     ; Рабочий регистр
.def    WorkH = r25     ; Рабочий регистр

.equ    Tx = $1000      ; 4096 тактов таймера для Tx=65,536 мс
.equ    Max = $2000     ; Макс. продолжительность интегрирования Uref-
.equ    a = PB0         ; Разряд 0 порта B, линия a на этапе S1
.equ    b = PB1         ; Разряд 1 порта B, линия b на этапе S1

Reset:
rjmp Initial            ; Переход к части инициализации

;***********************************************************************
;* Подпрограмма и объявления для управления ЖК-модулем идентичны
;* рассмотренным в разделе "Подключение ЖК-модулей" и потому
;* здесь повторно не рассматриваются.
;***********************************************************************

HexAscii:                       ; Преобразование шестнадцатеричного в Ascii
```

```
000050 302a    cpi prm1,$0A          ; Больше 9 ?
000051 f410    brsh HA1              ; Переход, если > 9
000052 5d20    subi prm1,-$30        ; Прибавляем $30
000053 9508    ret
HA1:
000054 5c29    subi prm1,-$37        ; Прибавляем $37
000055 9508    ret
SetAdr:                              ; Установка адреса индикатора для Ux
000056 f02e    brts Ux1              ; Переход, если флаг T = 1 (Ux1)
000057 e421    ldi prm1,64+1         ; Адрес = 2-й символ 2-й строки
000058 6820    sbr prm1,1<<7         ; Устанавливаем разряд 7 -> строка DD-RAM
000059 dfe6    rcall WaitBusy        ; Ожидаем готовности ЖК-модуля к приему
00005a dfcf    rcall SendCom         ; Устанавливаем указатель адреса в DD-RAM
00005b 9508    ret
Ux1:                                 ; Измеряемое значение Ux1
00005c e429    ldi prm1,64+9         ; Адрес = 10-й символ 2-й строки
00005d 6820    sbr prm1,1<<7         ; Устанавливаем разряд 7 -> строка DD-RAM
00005e dfe1    rcall WaitBusy        ; Ожидаем готовности ЖК-модуля к приему
00005f dfca    rcall SendCom         ; Устанавливаем указатель адреса в DD-RAM
000060 9508    ret

DispErg:                             ; Вывод результата на табло
000061 2f27    mov prm1,ErgH         ; Старший байт результата
000062 dfed    rcall HexAscii        ; Шестнадцатеричное значение в младшем
                                     ; полубайте -> Ascii
000063 dfdc    rcall WaitBusy        ; Ожидаем готовности ЖК-модуля к приему
000064 dfc8    rcall SendDat         ; Передаем символ, автоинкремент адреса
000065 2f26    mov prm1,ErgL         ; Младший байт результата
000066 9522    swap prm1             ; Меняем местами старш. и млад. полубайты
000067 702f    andi prm1,$0F         ; Обнуляем старший полубайт
000068 dfe7    rcall HexAscii        ; Шестнадцатеричное значение в младшем
                                     ; полубайте -> Ascii
000069 dfd6    rcall WaitBusy        ; Ожидаем готовности ЖК-модуля к приему
00006a dfc2    rcall SendDat         ; Передаем символ, автоинкремент адреса
00006b 2f26    mov prm1,ErgL         ; Младший байт результата
00006c 702f    andi prm1,$0F         ; Обнуляем старший полубайт
00006d dfe2    rcall HexAscii        ; Шестнадцатеричное значение в младшем
                                     ; полубайте -> Ascii
00006e dfd1    rcall WaitBusy        ; Ожидаем готовности ЖК-модуля к приему
00006f dfbd    rcall SendDat         ; Передаем символ, автоинкремент адреса
000070 9508    ret

DispOFL:                             ; Выдача сообщения о переполнении
000071 e42f    ldi prm1,'O'
000072 dfcd    rcall WaitBusy        ; Ожидаем готовности ЖК-модуля к приему
000073 dfb9    rcall SendDat         ; Передаем символ, автоинкремент адреса
000074 e426    ldi prm1,'F'
000075 dfca    rcall WaitBusy        ; Ожидаем готовности ЖК-модуля к приему
000076 dfb6    rcall SendDat         ; Передаем символ, автоинкремент адреса
000077 e42c    ldi prm1,'L'          ; "OFL" = "OverFLow" (переполнение)
000078 dfc7    rcall WaitBusy        ; Ожидаем готовности ЖК-модуля к приему
000079 dfb3    rcall SendDat         ; Передаем символ, автоинкремент адреса
00007a 9508    ret

Initial:                             ; Инициализация регистров ввода/вывода
000092 e092    ldi WorkH,High(RamEnd)
```

```
000093 bf9e    out sph,WorkH
000094 e59f    ldi WorkH,Low(RamEnd)
000095 bf9d    out spl,WorkH          ; Инициализация стека
000096 e30f    ldi tmp1,$3F           ; Разряды 7..6 = 0, остальные = 1
000097 bb02    out PortD,tmp1         ; /RD,/WR = 0, ост. разряды порта D = 1
000098 bb05    out PortC,tmp1         ; Ena, RS = 0, ост. разряды порта C = 1
000099 ec00    ldi tmp1,$C0           ; Разряд 7 = 1, разряд 6 = 1
00009a bb01    out DDRD,tmp1          ; Разряды 7..6 - выходы, остальные - входы
00009b bb04    out DDRC,tmp1          ; Разряды 7..6 - выходы, остальные - входы
00009c ef0f    ser tmp1               ; Инициализация tmp1
00009d bb0a    out DDRA,tmp1          ; Порт A - выход
00009e e092    ldi WorkH,$02          ; b (разряд 1) - в 1, остальные - в 0
00009f bb98    out PortB,WorkH        ; Выбираем Uref-
0000a0 e093    ldi WorkH,$03          ; Разряды 0..1,4..7 - в 1, остальные - в 0
0000a1 bb97    out DDRB,WorkH         ; b (PB1) и a (PB0) - выход
0000a2 e097    ldi WorkH,$07          ; Захват по нар. фронту на выходе компар.
0000a3 b998    out ACSR,WorkH         ; Регистр управления компаратора
0000a4 df68    rcall InitLCD          ; Инициализируем 8-разрядный ЖК-модуль
0000a5 e02c    ldi prm1,$0C           ; Табло - вкл., курсор, мерцание - откл.
0000a6 df99    rcall WaitBusy         ; Ожидаем готовности ЖК-модуля к приему
0000a7 df82    rcall SendCom          ; Передаем команду
0000a8 e021    ldi prm1,$01           ; Гасим табло, курсор - в начало
0000a9 df96    rcall WaitBusy         ; Ожидаем готовности ЖК-модуля к приему
0000aa df7f    rcall SendCom          ; Передаем команду
0000ab e0f4    ldi ZH,high(Text1)
0000ac e0e0    ldi ZL,low(Text1)      ; Указатель Z указывает на начало текста
0000ad e150    ldi cnt,16             ; Текст - из 16-ти символов
0000ae df87    rcall OutText          ; Выводим Text1
0000af e420    ldi prm1,64            ; Адрес = начало второй строки
0000b0 6820    sbr Prm1,1<<7          ; Устанавливаем разряд 7 -> строка DD-RAM
0000b1 df8e    rcall WaitBusy         ; Ожидаем готовности ЖК-модуля к приему
0000b2 df77    rcall SendCom          ; Устанавливаем указатель адреса в DD-RAM
0000b3 e0f4    ldi ZH,high(Text2)
0000b4 e0e8    ldi ZL,low(Text2)      ; Указатель Z указывает на начало текста
0000b5 e150    ldi cnt,16             ; Текст - из 16-ти символов
0000b6 df7f    rcall OutText          ; Выводим Text2

Messung:
0000b7 ef90    ldi WorkH,high(-Tx)    ; Старший байт времени интегрирования Tx -
0000b8 bd9d    out TCNT1H,WorkH       ;   в счетный регистр
0000b9 e080    ldi WorkL,low(-Tx)     ; Младший байт времени интегрирования Tx -
0000ba bd8c    out TCNT1L,WorkL       ;   в счетный регистр
Schritt1:                             ; Установка нулевой точки интегрирования
0000bb 9ac1    sbi PortB,b            ; b = 1
0000bc 9945    sbic acsr,aco          ; Пропускаем следующую команду, если
                                      ; AIN0 < AIN1 (aco=0)
0000bd c004    rjmp S12               ; Переход, если AIN0 > AIN1 (aco=1)
0000be 98c0    cbi PortB,a            ; a = 0 -> выбираем Uref-
S11:
0000bf 9b45    sbis acsr,aco          ; Пропускаем следующую команду, если
                                      ; AIN0 > AIN1 (aco=1)
0000c0 cffe    rjmp S11
0000c1 c003    rjmp Schritt2          ; Достигаем нулевой точки
S12:
0000c2 9ac0    sbi PortB,a            ; a = 1 -> выбираем Uref+
S13:
```

```
0000c3 9945    sbic acsr,aco          ; Пропускаем следующую команду, если
                                      ; AIN0 < AIN1 (aco=0)
0000c4 cffe    rjmp S13
Schritt2:                             ; Начало Tx при интегрировании Ux
0000c5 b398    in WorkH,PortB
0000c6 f990    bld WorkH,a            ; T-Flag -> a (Ux0 или Ux1)
0000c7 7f9d    cbr WorkH,1<<b         ; b = 0
0000c8 bb98    out PortB,WorkH        ; Переключаем Ux0 или Ux1
0000c9 e493    ldi WorkH,$43          ; ICNC1=0, ICES1=1, CTC1=0
0000ca bd9e    out TCCR1B,WorkH       ; Запускаем T/C1, такт сист. синхр. /64
S21:
0000cb b58c    in WorkL,TCNT1L        ; Альтернатива для OCR1A-Int.
0000cc b59d    in WorkH,TCNT1H        ; Считываем содержимое счетчика
0000cd 5f8f    subi WorkL,$FF         ; Ожидаем, когда счетчик перейдет
0000ce 4f9f    sbci WorkH,$FF         ;   в состояние $FFFF
0000cf f3d8    brlo S21               ; Переход, если еще не достигнуто
0000d0 b398    in WorkH,PortB         ; Готовим подключение Uref-
0000d1 7f9e    cbr WorkH,1<<a         ; a = 0,
0000d2 6092    sbr WorkH,1<<b         ; b = 1 -> выбрано Uref-
S22:
0000d3 b788    in WorkL,TIFR          ; Извлекаем флаги T/C1
0000d4 ff87    sbrs WorkL,TOV1        ; Пропускаем следующую команду, если
                                      ; переполнение в $0000
0000d5 cffd    rjmp S22               ; Переход, если Tx еще не завершено
Schritt3:                             ; Определение TR при интегрировании Uref-
0000d6 bb98    out PortB,WorkH        ; Подано Uref-
0000d7 6888    sbr WorkL,1<<TOV1 | 1<<ICF1 ; ICF1,TOV1 = 1
0000d8 bf88    out TIFR,WorkL         ; Сбрасываем ICF1 и TOV1
WaitCapt:                             ; Ожидание захвата
0000d9 b788    in WorkL,TIFR          ; Извлекаем флаги T/C1
0000da ff83    sbrs WorkL,ICF1        ; Пропускаем следующую команду, если
                                      ; имеет место захват
0000db c012    rjmp CheckOFL          ; Переполнение счетчика?
MessEnde:
0000dc b564    in ErgL,ICR1L          ; Результат - из регистра захвата
0000dd b575    in ErgH,ICR1H
WaitTime:                             ; Ожидание окончания измерения
0000de b58c    in WorkL,TCNT1L
0000df b59d    in WorkH,TCNT1H        ; Считываем содержимое счетчика
0000e0 5080    subi WorkL,low(Max)    ; Достигнуто ли
0000e1 4290    sbci WorkH,high(Max)   ;   максимальное состояние счетчика?
0000e2 f3d8    brlo WaitTime          ; Переход, если еще нет
0000e3 9576    lsr ErgH
0000e4 9567    ror ErgL
0000e5 9576    lsr ErgH
0000e6 9567    ror ErgL               ; Делим результат на 4
Display:
0000e7 df6e    rcall SetAdr           ; Установка адреса индикатора для Ux
0000e8 fd73    sbrc ErgH,3            ; Если разряд 11 = 0, то нет переполнения
0000e9 c002    rjmp OFlow             ; Переход, если переполнение
0000ea df76    rcall DispErg          ; Выводим результат на табло
0000eb c009    rjmp Next
OFlow:
0000ec df84    rcall DispOFL          ; Выводим сообщение о переполнении
0000ed c007    rjmp Next
```

```
CheckOFL:
0000ee b58c    in WorkL,TCNT1L
0000ef b59d    in WorkH,TCNT1H          ; Считываем содержимое счетчика
0000f0 5080    subi WorkL,low(Max)      ; Превышено ли
0000f1 4290    sbci WorkH,high(Max)     ;   максимальное состояние счетчика?
0000f2 f330    brlo WaitCapt            ; Переход, если еще нет
0000f3 df62    rcall SetAdr             ; Установка адреса индикатора для Ux
0000f4 cff7    rjmp OFlow               ; Возникло переполнение

Next:
0000f5 e490    ldi WorkH,$40            ; ICNC1=0, ICES1=1, CTC1=0
0000f6 bd9e    out TCCR1B,WorkH         ; Останавливаем T/C1
0000f7 f016    brts N1                  ; Переход, если флаг T=1
0000f8 9468    set                      ; Устанавливаем флаг T -> Ux1
0000f9 cfbd    rjmp Messung             ; Следующее измерение
N1:
0000fa 94e8    clt                      ; Сбрасываем флаг T -> Ux0
0000fb cfbb    rjmp Messung             ; Следующее измерение

.org $400
Text1:
.db "Ux0:    Ux1:    "
Text2:
.db "$---    $---    "
```

Описание подпрограмм

Подпрограмма `HexAscii` преобразовывает шестнадцатеричный полубайт в диапазоне 0...9 и полубайт в диапазоне A...F в ASCII-эквивалент с помощью прибавления 30_h и 37_h соответственно.

Подпрограмма `SetAdr` устанавливает адрес индикатора во вторую позицию второй строки для вывода значения U_{X1} и в 10-ю позицию той же строки для вывода значения U_{X2}.

Подпрограмма `DispErg` выводит три цифры измеренного значения по адресу индикатора, предварительно установленному подпрограммой `SetAdr`.

Подпрограмма `DispOFL` выводит три ASCII-символа "OFL", сигнализирующие о переполнении измеренного значения, по адресу индикатора, предварительно установленному подпрограммой `SetAdr`.

Описание главной программы

Когда поступает сигнал сброса, регистр TCCR1B автоматически инициализируется значением $00, останавливая тем самым T/C1. По сбросу при включении питания происходит переход по адресу $000, которому соответствует часть инициализации, обозначенная меткой `Initial`. После инициализации указателя стека должны быть инициализированы как выходы разряды портов, используемые модулем STK200 при управлении табло: A15 и A14 — для порта C; /RD, /WR — для порта D, а также шина данных порта A. Разряды 0...1 порта B управляют аналоговым мультиплексором S1 (см. рис. 16.26), и также определены как выходы. Входы AIN0 и AIN1 компаратора внутренне соединены с разрядами 2...3 порта B, и объявлены как входы.

Интегрированный аналоговый компаратор с помощью своего регистра ACSR настроен таким образом, чтобы нарастающий фронт на его выходе приводил к захвату (то есть, к переносу текущего состояния счетчика в регистр захвата).

После инициализации ЖК-модуля и вывода в первой строке заголовка "Ux0: Ux1: ", а во второй — шаблона "$--- $--- ", начинается процесс измерения, который в бесконечном цикле выполняет примерно четыре измерения в секунду. Измерения происходят попеременно для U_{X1} и U_{X2}, поэтому каждое из значений выводится на табло два раза в секунду.

В начале измерения счетчик T/C1 инициализируется отрицательной константой `Tx`, которой соответствует время интегрирования 65,536 мс. Поскольку T/C1 работает как суммирующий счетчик, он выполняет счет за 4096 шагов от $(-T_X)$ до $0000.

После описанного ранее первого этапа установки нулевой точки следует второй этап интегрирования для измеренного напряжения U_{X1} (или U_{X2}). Поскольку U_{X1} и U_{X2} измеряются попеременно, после каждого измерения в части программы `Next` переключается флаг T, определяющий напряжение для следующего измерения.

В начале второго этапа с помощью коммутатора S1 на вход интегратора подается входное напряжение, соответствующее флагу T. Затем счетчик T/C1 с помощью регистра управления TCCR1B конфигурируется так, чтобы было отключено подавление помех для функции захвата, нарастающий фронт на выходе аналогового компаратора вызывал срабатывание функции захвата, а запуск осуществляется по каждому 64-му такту системной синхронизации на тактовом входе.

Незадолго до завершения времени измерения (при состоянии счетчика $FFFF) на порт B выдаются управляющие сигналы a и b аналогового коммутатора S1 для подготовки к быстрому переключению на интегрирование U_{ref-}. Наступает третий этап, когда состояние счетчика T/C1 переходит в $0000.

После сброса флага переполнения счетчика TOV1 и флага захвата ICF1 в части программы, обозначенной меткой `WaitCapt`, реализовано ожидание наступления захвата или возможного превышения диапазона измерения. Флаг ICF1 указывает на то, что содержимое счетчика было перенесено в регистр захвата, и происходит ветвление к части программы `MessEnde`. В этой части ожидается окончание полного времени измерения, чтобы продолжительность всех циклов измерения была примерно одинакова, несмотря на различия во времени, затраченном на управление в нулевой точке интегрирования на первом этапе.

Когда официальное время измерения истекло, результат счета делится на четыре, поскольку 4096 импульсам времени интегрирования T_X соответствует напряжение 1,024 В, и полученное таким образом состояние счетчика дает значение измеренного напряжения U_X.

Перед выводом измеренных значений в части программы `Display` выполняется проверка на предмет выхода за пределы диапазона измерений, и, если такого выхода не обнаружено, результат отображается на табло. Если произошел выход за пределы измерений (как в случае превышения верхнего состояния счетчика в части программы `WaitCapt` еще до возникновения захвата), о переполнении извещает надпись "OFL", выведенная на табло.

В последней части программы `Next` счетчик T/C1 останавливается, и с помощью переключения флага T готовится измерение другого входного напряжения, после чего бесконечный цикл продолжается переходом к метке `Messung`.

С помощью перемычки J1 (см. рис. 16.26) пользователь выбирает, какое напряжение он будет измерять в качестве U_{X1} и U_{X2}: используемые на плате напряжения U_{R1} и U_{R2} (перемычка помещена) или внешние входные напряжения U_{X1} и U_{X2} (перемычка извлечена). Само собой разумеется, при другом применении схемы вместо перемычки J1 на вход с коммутатора S1 может быть подан цифровой управляющий сигнал. В этом случае пользователь может управлять выбором между U_{R1} и U_{R2} (U_{X1} и U_{X2}) прямо из программы. Если из схемы удалить элементы R1 и R2 и освободить выводы 0 и 1 коммутатора, то можно реализовать измерение сразу четырех внешних напряжений.

Разрешение 11 разрядов не означает точности 11 разрядов. При тестировании рассмотренной схемы в лабораторных условиях с помощью модуля STK200 при постоянном входном напряжении была показана эффективная точность 8...9 разрядов при погрешности результатов измерения ±5...8.

Это, в первую очередь, обусловлено использованием интегрированного аналогового компаратора микроконтроллеров AVR, который специфицирован для входного диапазона 0...U_B и при сравнении работает у нижней границы 0 В своего диапазона. Ухудшение характеристики с приближением к нулевой точке можно четко проследить с помощью графика, представленного на рис. 16.28.

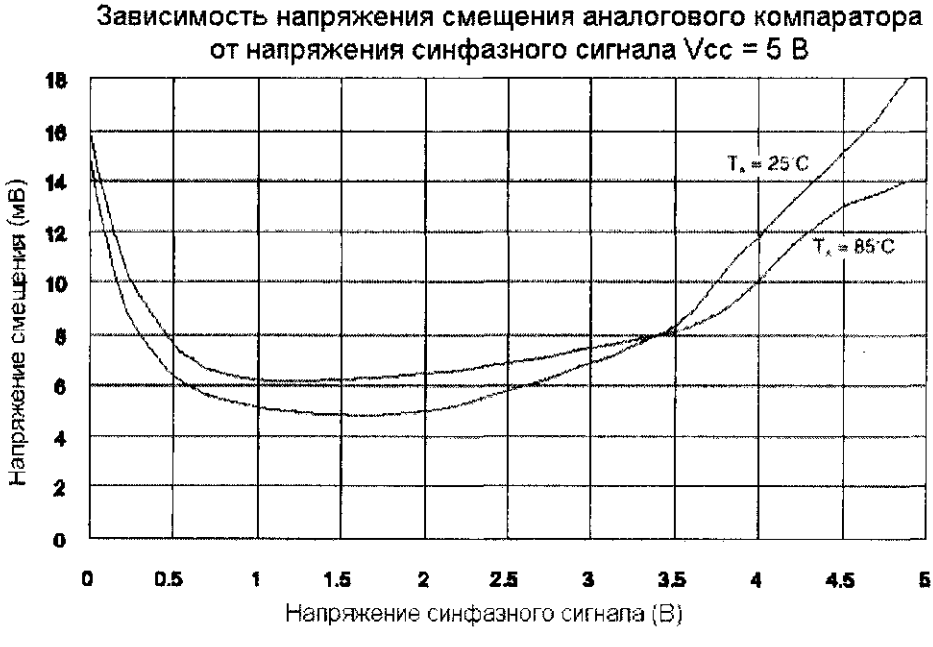


Рис. 16.28. Напряжение смещения аналогового компаратора

Использование вместо встроенного компаратора микроконтроллеров AVR внешнего компаратора LM311 (рис. 16.29) позволяет уменьшить погрешность результата измерения до ±2...3. Если же заменить лабораторную сборку печатной

платой с разделением аналоговых и цифровых соединений с "землей", то можно достичь точности в диапазоне разрешения.

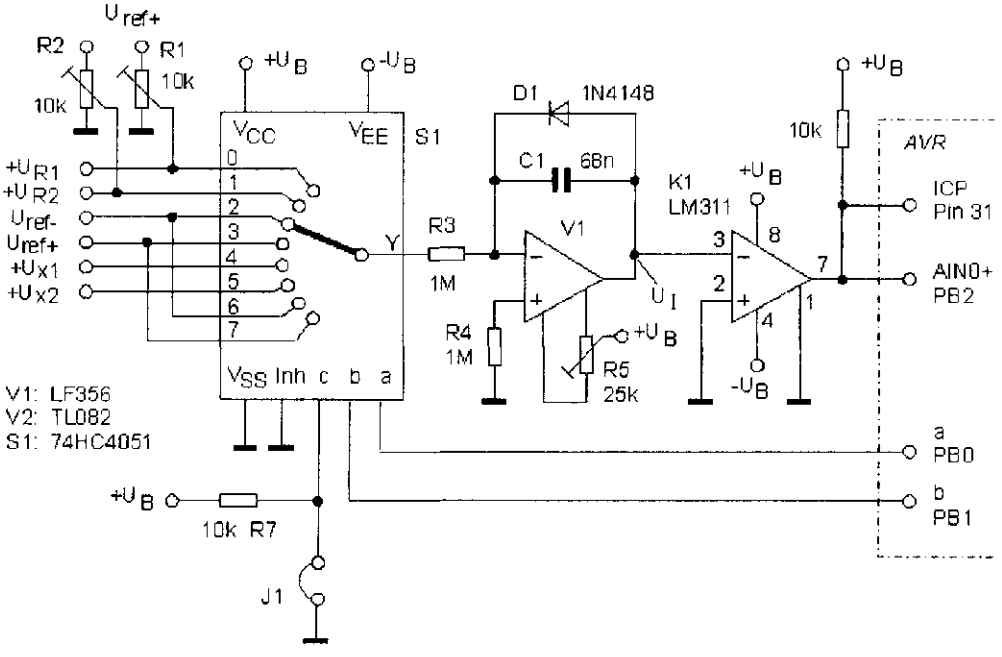


Рис. 16.29. Реализация функции захвата с помощью внешнего компаратора

На схеме, показанной на рис. 16.29, ниспадающий фронт сигнала на выходе K1 активизирует функцию захвата на выводе ICP в тот момент, когда выходное напряжение интегратора U_I имеет положительное значение. Кроме того, состояние выхода компаратора может быть опрошено через вывод PB2 (AIN0), который теперь расценивается как обычный вход, поскольку аналоговый компаратор больше не используется.

В этом случае необходимо внести незначительные изменения в программу (измененная версия программы находится на прилагаемом к книге компакт-диске в файле `\Programm\168LM311.asm`):

- регистр ACSR в части `Initial` должен быть инициализирован значением $00 (теперь захват срабатывает по ниспадающему фронту на входе ICP);
- при опросе компаратора вместо ACO в ACSR теперь опрашивается разряд PB2;
- в счетчик T/C1 в части `Schritt2` теперь загружается стартовое значение $03 вместо $43 (захват по ниспадающему фронту на входе ICP);
- счетчик T/C1 теперь останавливается при достижении значения $00, а не $40 (часть `Next`).

Программная реализация приемопередатчика UART для микроконтроллера AT90S1200

В отличие от всех остальных представителей базовой серии семейства AVR, микроконтроллер AT90S1200 не оснащен встроенным аппаратным приемопередатчиком UART. Тем не менее, было бы неплохо получить возможность обращаться к нему через последовательный интерфейс ПК, если в некотором приложении модель AT90S2313 менее предпочтительна с точки зрения цены.

В различных изданиях встречается несколько разных примеров "программного" UART для микроконтроллера AT90S1200, однако все известные реализации предусматривают полный прием байта за раз, без остановки. При скорости передачи 9600 бод это блокирует работу микроконтроллера примерно на одну миллисекунду — промежуток времени, за который AT90S1200 не может отрегаировать на несколько важных событий, а при тактовой частоте 12 МГц мог бы выполнить до 12000 команд.

По этой причине предлагаем вниманию читателя реализацию интерфейса RS232 для полнодуплексной передачи без контроля по четности с постоянной скоростью 9600 бод — приемопередатчик UART в форме "конечного автомата", прерывающий выполнение программы каждые 20,833 мкс. При этом основные события главной программы могут по-прежнему обслуживаться.

Через каждые 20,833 мкс подпрограмма UART опрашивает состояние линий приема/передачи. Если обе линии при предыдущем прерывании от таймера находились в ждущем режиме и до сих пор из него не вышли, то сразу же происходит выход из подпрограммы, и продолжает выполняться главная программа. Однако, если подпрограмма обнаружит старт-бит в линии приема или установленный флаг передачи, то это указывает на то, что главная программа должна записать отправляемый байт в буфер передачи TBuf, изменив тем самым прежнее состояние ждущего режима на "Начало приема" или "Передача бита".

Период повторения прерываний от таймера выбираем равным 20,833 мкс, поскольку это — ровно пятая часть длительности передачи бита при скорости 9600 бод, которую легко получить, разделив частоту системной синхронизации 12 МГц на 250. Таким образом, линия приема за время приема одного бита опрашивается пять раз, что гарантирует доставку этого бита. Для того чтобы определить, *что* принимается: 0 или 1 — используется метод "мажорирования". При таком способе также устраняется влияние импульсных помех, поскольку значение принимаемого бита проверяется в пять раз чаще (рис. 16.30).

Результат пяти опросов бита суммируется. При этом низкий уровень сигнала принимают за +1, а высокий — за –1. На логический уровень принимаемого бита указывает старший разряд полученной суммы.

Отправитель начинает передачу байта асинхронно к интервалам опроса по прерываниям от таймера, передавая старт-бит. На рис. 16.30 видно, что старт-бит опознается по низкому уровню сигнала в линии приема после того как при предыдущем опросе был прочитан высокий уровень (ждущий режим). Такая смена уровня расценивается как начало старт-бита, и сумма опроса (RSum) сбрасывается в 0.

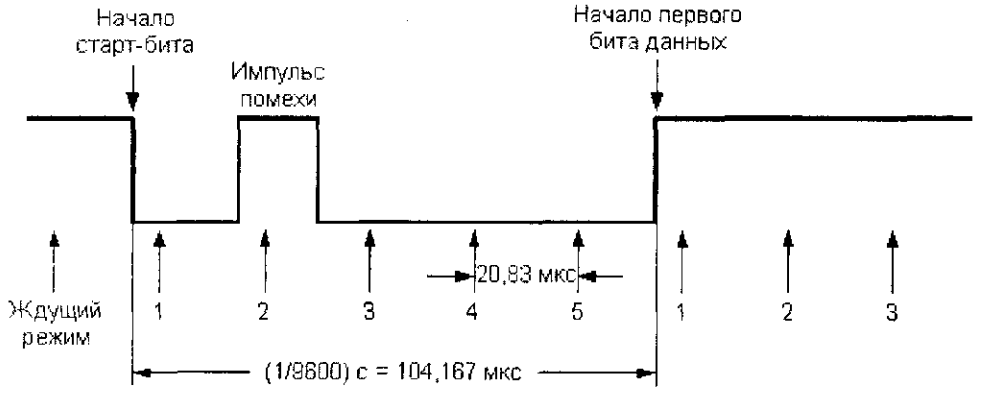


Рис. 16.30. Опрос бита на линии приема

При следующем прерывании, которое возникает через 20,833 мкс, в линии приема появляется импульс помехи, и потому считывается 1, а сумма декрементируется в $FF. Третий опрос опять дает 0, и сумма инкрементируется в $00. При следующих двух опросах опять распознается 0, и сумма в конце концов принимает значение $02. Старший разряд суммы, несмотря на то, что в момент передачи возникла помеха, даст корректное значение старт-бита 0.

После прохождения подобной схемы выполняется опрос восьми битов данных, последовательно сохраняемых во временном буфере приема RTMP. Как только прочитаны все восемь битов, содержимое буфера RTMP переписывается в постоянный буфер приема RBuf, о чем главная программа извещается установкой флага RRdy. После этого буфер RTMP сразу же освобождается для приема следующих символов. Считывание символа главной программой определяется по сбросу флага RRdy.

Когда главная программа собирается передать байт по последовательному интерфейсу, она записывает его в буфер TBuf и устанавливает флаг передачи TBusy, чтобы подать сигнал подпрограмме обработки прерывания UART. После передачи байта, корректно снабженного старт- и стоп-битами, подпрограмма обработки прерывания опять сбрасывает флаг TBusy, чтобы известить главную программу о начале новой передачи.

Когда передается байт, в линию TxD сначала выдается старт-бит, занимающий пять интервалов передачи. После подсчета пяти прерываний в линию TxD выдается первый бит данных (начиная с младшего), затем следует пять циклов передачи последующего бита и т.д. Эта процедура выполняется до тех пор, пока не будут переданы все восемь битов данных, и завершается передачей стоп-бита, также занимающего по длительности пять прерываний.

Блок-схема программной реализации UART показана на рис. 16.31.

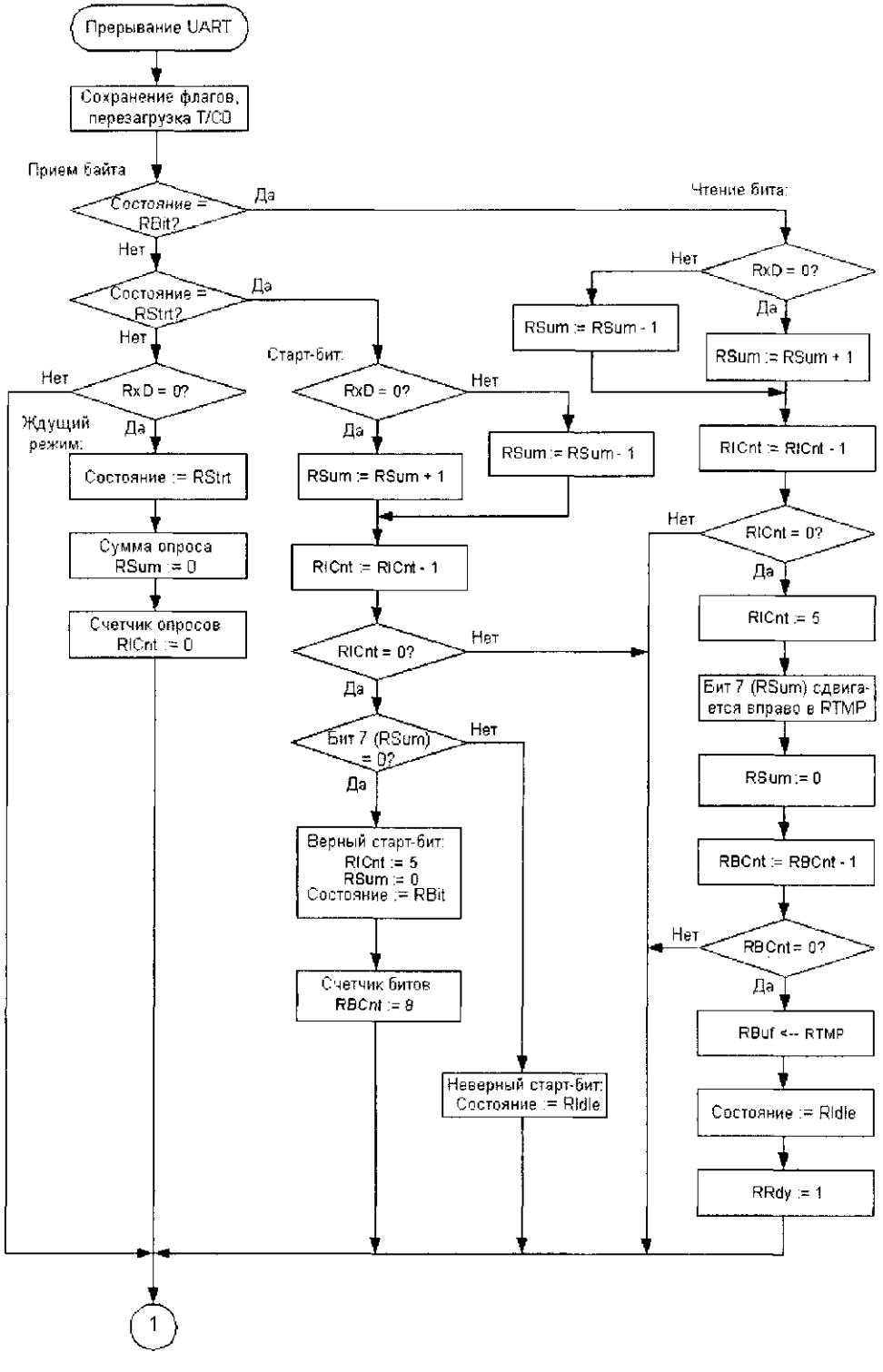


Рис. 16.31 (начало). Блок-схема программной реализации UART для микроконтроллера AT90S1200

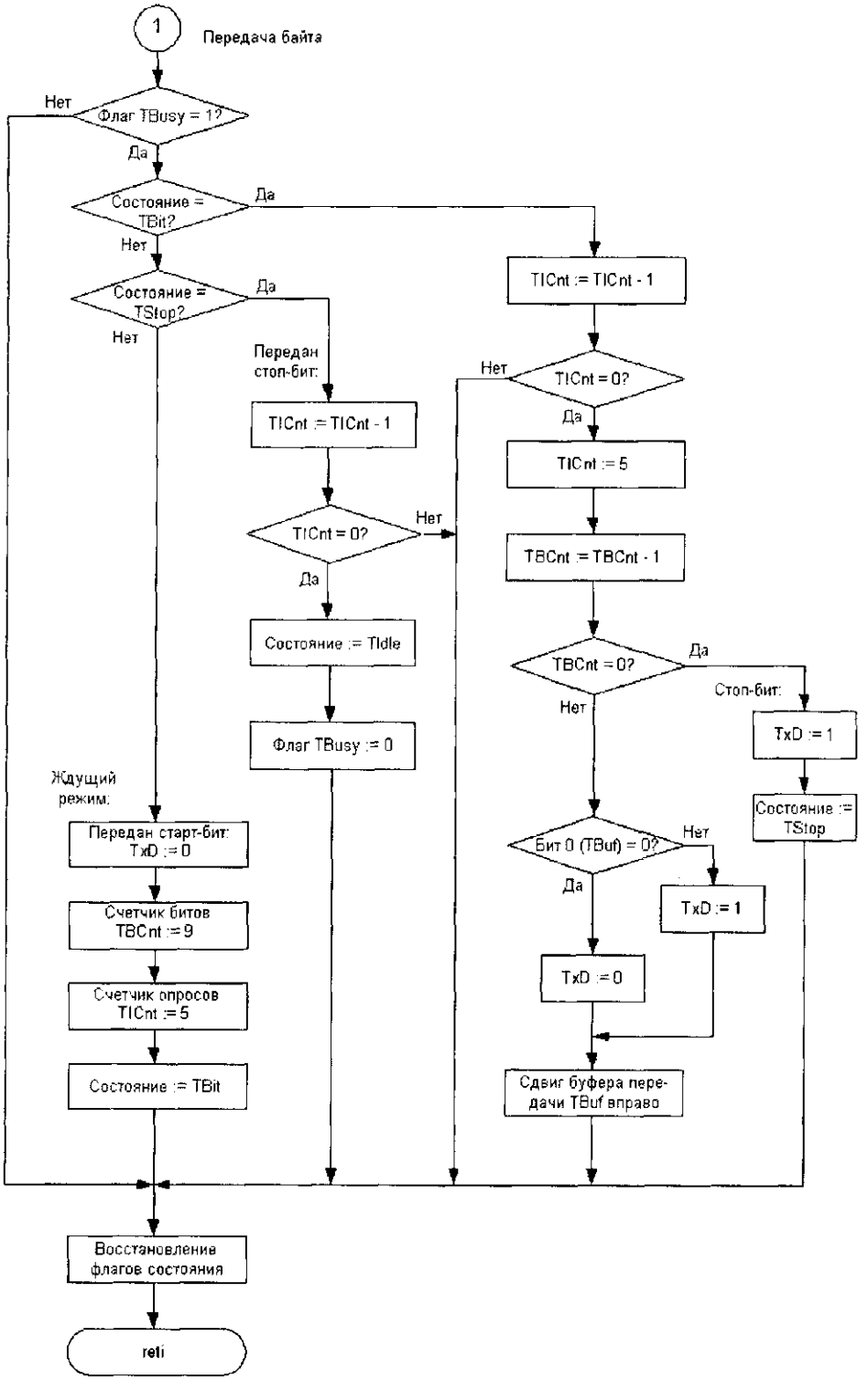


Рис. 16.31 (окончание)

Интерфейс RS232 и аппаратный приемопередатчик UART были подробно рассмотрены в главе 6. По сути, программный UART построен по той же схеме. Поскольку отсутствует флаг FE, который устанавливается при обнаружении ошибок кадрирования, и флаг OR, который устанавливается при обнаружении переполнения буфера, при приеме байта нет ожидания стоп-бита, так как в этом отношении прием всегда корректен. Если же пользователь предъявляет повышенные требования к надежности передачи, то можно легко дополнительно реализовать опрос стоп-бита, флагов и бита четности.

Если для какого-либо случая применения скорость передачи 9600 бит/с слишком высока, ее можно легко понизить при совсем незначительных изменениях в программе.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\169UART.asm`

```
;**** Программная реализация UART для микроконтроллера AT90S1200 *********
;*
;* Входной сигнал RxD принимается со скоростью 9600 бод.
;* Выходной сигнал TxD передается с такой же скоростью.
;* В главной программе в качестве примера принятый байт передается обратно.
;* Тактовая частота микроконтроллера AVR: 12 МГц
;*
;***************************************************************************
.nolist
.include "1200def.inc"
.list

;***** Определения регистров:
.def    Save = r12      ; Буфер для временного хранения содержимого SREG
.def    RTmp = r13      ; Временный буфер приема
.def    RBuf = r14      ; Буфер приема
.def    TBuf = r15      ; Буфер передачи
.def    Stat = r16      ; Состояние UART (T_BUSY, T_Stop, T_Bit, T_Idle,
                        ; R_RDY, R_Bit, R_Start, R_Idle)
.def    tmp1 = r17      ; Общий рабочий регистр 1
.def    tmp2 = r18      ; Общий рабочий регистр 2
.def    TBCnt = r19     ; Счетчик битов при передаче
.def    TICnt = r20     ; Счетчик интервалов опроса при передаче
.def    RBCnt = r21     ; Счетчик битов при приеме
.def    RICnt = r22     ; Счетчик интервалов опроса при приеме
.def    RSum = r23      ; Сумма приема

; Описания:
.equ    Time = 250      ; Количество тактовых импульсов до следующего прерывания
.equ    RxD = PD0       ; Разряд 0 порта D = RxD
.equ    TxD = PD1       ; Разряд 1 порта D = TxD
.equ    RIdle = 0       ; Разряд 0 = ждущий режим при приеме
.equ    RStrt = 1       ; Разряд 1 = начало приема
.equ    RBit = 2        ; Разряд 2 = опрос битов при приеме
.equ    RRdy = 3        ; Разряд 3 = в буфер приема получен новый байт
.equ    TIdle = 4       ; Разряд 4 = ждущий режим при передаче
.equ    TBit = 5        ; Разряд 5 = начало передачи
.equ    TStop = 6       ; Разряд 6 = опрос битов при передаче
.equ    TBusy = 7       ; Разряд 7 = в данный момент выполняется передача
```

```
RESTART:
000000 c050    rjmp Initial            ; Переход к части инициализации
000001 9518    reti                    ; Внешнее прерывание (не используется)

;***************************************************************************
;* Прерывание от таймера 0
;* Эта подпрограмма вызывается каждые 20,833 мкs (250 тактовых импульса
;* таймера при частоте кварца 12 МГц).
;* В начале обработки прерывания сохраняются флаги главной программы,
;* а таймер после его переполнения инициализируется по-новому.
;* Далее реализуется UART.
;* В конце обработки прерывания восстанавливаются флаги главной программы,
;* и с помощью команды reti опять разрешаются прерывания.
;***************************************************************************

Timer0_Int:          ; Прерывание от таймера 0, обслуживание: 13 тактов
000002 b6cf    in Save,SREG            ; Сохраняем флаги главной программы
000003 b712    in tmp1,TCNT0           ; Текущее содержимое таймера - в tmp1
000004 5f17    subi tmp1,Time-3        ; Интервал минус время выполнения
000005 bf12    out TCNT0,tmp1          ;   трех команд инициализации
UARTIn:                                ; USTartIn: 20 тактов, UBitIn: 19 тактов
000006 fd02    sbrc Stat,RBit;         ; Следующий переход выполняется, если
000007 c019    rjmp UBitIn             ;   UART - в режиме опроса битов
000008 fd01    sbrc Stat,RStrt         ; Следующий переход выполняется, если
000009 c007    rjmp UStrtIn            ;   UART - в режиме начала передачи
UIdleIn:                               ; UART - в ждущем режиме при приеме
00000a 9980    sbic PinD,RxD           ; Пропускаем следующую команду, если
                                       ;   RxD=0 (старт-бит)
00000b c023    rjmp UARTOut            ; Не старт-бит, UART - еще в ждущем режиме
00000c 7f08    cbr Stat,$07            ; Сбрасываем флаги состояния UART 0..2
00000d 6002    sbr Stat,1<<RStrt       ; Входное состояние UART = R_Strt
00000e e065    ldi RICnt,5             ; Инициализируем счетчик интервалов опроса
00000f 2777    clr RSum                ; Обнуляем сумму приема
000010 c01e    rjmp UARTOut
UStrtIn:
000011 9573    inc RSum                ; Увеличиваем сумму приема на 1
000012 9980    sbic PinD,RxD           ; Пропускаем следующую команду, если RxD=0
000013 5072    subi RSum,$02           ; Уменьшаем сумму приема на 2
000014 956a    dec RICnt               ; Уменьшаем счетчик опросов на 1
000015 f4c9    brne UARTOut            ; Переход, если старт-бит не завершен
000016 ff77    sbrs RSum,7             ; Если разряд 7 = 1, то неверный старт-бит
000017 c003    rjmp UIS3               ; Переход, если корректный старт-бит
000018 7f08    cbr Stat,$07            ; Сбрасываем флаги состояния UART 0..2
000019 6001    sbr Stat,1<<RIdle       ; Входное состояние UART = R_Idle
00001a c014    rjmp UARTOut
UIS3:
00001b e065    ldi RICnt,5             ; 5 интервалов опроса на один бит
00001c e058    ldi RBCnt,8             ; Считываем 8 битов
00001d 2777    clr RSum                ; Обнуляем сумму приема
00001e 7f08    cbr Stat,$07            ; Сбрасываем биты состояния UART 0..2
00001f 6004    sbr Stat,1<<RBit        ; Входное состояние UART = R_Bit
000020 c00e    rjmp UARTOut
UBitIn:
000021 9573    inc RSum                ; Увеличиваем сумму приема на 1
000022 9980    sbic PinD,RxD           ; Пропускаем следующую команду, если RxD=0
```

```
000023 5072    subi RSum,$02          ; Уменьшаем сумму приема на 2
000024 956a    dec RICnt              ; Уменьшаем счетчик опросов на 1
000025 f449    brne UARTOut           ; Переход, если бит еще не завершен
000026 e065    ldi RICnt,5            ; Инициализируем счетчик интервалов опроса
000027 0f77    lsl RSum               ; Разряд 7 - в флаг переноса
000028 94d7    ror RTmp               ; Флаг переноса вправо во временный буфер
000029 2777    clr RSum               ; Обнуляем сумму приема
00002a 955a    dec RBCnt              ; Уменьшаем счетчик битов на 1
00002b f419    brne UARTOut           ; Переход, если еще не все 8 битов
00002c 2ced    mov RBuf,RTmp          ; Копируем считанный байт в буфер приема
00002d 7f08    cbr Stat,$07           ; Сбрасываем флаги состояния UART 0..2
00002e 6009    sbr Stat,1<<RIdle | 1<<RRdy ; Входное состояние UART = R_Idle
                                      ; R_RDY=1: байт считан
UARTOut:
00002f ff07    sbrs Stat,TBusy        ; Следующий переход выполняется, если
000030 c01e    rjmp IntEnde           ;   UART - не в режиме передачи
000031 fd05    sbrc Stat,TBit         ; Следующий переход выполняется, если
000032 c008    rjmp UBitOut           ;   UART в режиме опроса бита
000033 fd06    sbrc Stat,TStop        ; Следующий переход выполняется, если
000034 c016    rjmp UStopOut          ;   UART в режиме прекращения передачи
UIdleOut:                             ; UART - в ждущем режиме при передаче
000035 9891    cbi PortD,TxD          ; Выдаем старт-бит (TxD=0)
000036 e045    ldi TICnt,5            ; 5 опросов на бит
000037 e039    ldi TBCnt,9            ; Выдаем старт-бит и 8 битов данных
000038 780f    cbr Stat,$70           ; Сбрасываем флаги состояния UART 4..6
000039 6200    sbr Stat,1<<TBit       ; Выходное состояние UART = T_Bit
00003a c014    rjmp IntEnde
UBitOut:
00003b 954a    dec TICnt              ; Уменьшаем счетчик опросов на 1
00003c f491    brne IntEnde           ; Переход, если вывод бита не завершен
00003d e045    ldi TICnt,5            ; Инициализируем счетчик опросов
00003e 953a    dec TBCnt              ; Уменьшаем счетчик битов на 1
00003f f039    breq StopBit           ; Переход, если переданы все биты
000040 fcf0    sbrc TBuf,0            ; Пропускаем следующую команду, если
                                      ; младший разряд = 0
000041 c002    rjmp UO1
000042 9891    cbi PortD,TxD          ; Выход TxD=0
000043 c001    rjmp UO2
UO1:
000044 9a91    sbi PortD,TxD          ; Выход TxD=1
UO2:
000045 94f6    lsr TBuf               ; Сдвигаем временный буфер на 1 вправо
000046 c008    rjmp IntEnde           ; Переход, если не все биты переданы
StopBit:
000047 9a91    sbi PortD,TxD          ; Передаем стоп-бит (TxD=1)
000048 780f    cbr Stat,$70           ; Сбрасываем флаги состояния UART 4..6
000049 6400    sbr Stat,1<<TStop      ; Выходное состояние UART = T_Stop
00004a c004    rjmp IntEnde
UStopOut:
00004b 954a    dec TICnt              ; Уменьшаем счетчик опросов на 1
00004c f411    brne IntEnde           ; Переход, если стоп-бит не завершен
00004d 700f    cbr Stat,$F0           ; Сбрасываем флаги состояния UART 4..7
00004e 6100    sbr Stat,1<<TIdle      ; Выходное состояние UART = T_Idle
                                      ; Флаг T_Busy = 0
IntEnde:
00004f becf    out SReg,Save          ; Устанавливаем старые флаги
```

```
000050 9518    reti                    ; Выход из подпрогр. обработки прерывания

Initial:
000051 e012    ldi tmp1,2
000052 bb12    out PortD,tmp1          ; PD1= 1
000053 bb11    out DDRD,tmp1           ; Разряд 1 = выход, остальные - входы
000054 e101    ldi Stat,$11            ; Входное и выходное состояние UART = Idle
000055 ee1c    ldi tmp1,-20            ; Устанавливаем фиктивный интервал
000056 bf12    out TCNT0,tmp1          ; Инициализируем таймер
000057 e011    ldi tmp1,1
000058 bf13    out TCCR0,tmp1          ; Запускаем таймер, делитель = 1
000059 e012    ldi tmp1,2
00005a bf19    out TIMSK,tmp1          ; Разрешаем прерывание от таймера
00005b 9478    sei                     ; Общее разрешение прерываний

Haupt:
00005c ff03    sbrs Stat,RRdy          ; Пропускаем следующую команду, если
                                       ; байт принят
00005d cffe    rjmp Haupt
H1:
00005e fd07    sbrc Stat,TBusy         ; Пропускаем следующую команду, если
                                       ; буфер передачи пуст
00005f cffe    rjmp H1                 ; Ожидаем окончания передачи
000060 2cfe    mov TBuf,RBuf           ; Копируем байт в буфер передачи
000061 6800    sbr Stat,1<<TBusy       ; Флаг указывает на передачу байта
000062 7f07    cbr Stat,1<<RRdy        ; Сбрасывает флаг приема
000063 cff8    rjmp Haupt              ; Ожидаем следующего байта
```

Главная программа выполняет инициализацию порта и таймера в бесконечном цикле, в котором только что принятый байт сразу же передается обратно отправителю (эхо).

При тестировании этой программы с помощью набора STK200 в служебной программе Hyper-Terminal (в Windows 95) никаких проблем не возникло.

Подключение к микроконтроллеру AT90S8515 микросхемы ЦАП MAX5154 через интерфейс SPI

Пусть к микроконтроллеру AT90S8515 требуется подключить микросхему ЦАП (цифро-аналоговый преобразователь) MAX5154 от компании Maxim. Эта микросхема содержит два ЦАП с разрешением 12 бит, а также имеет встроенный интерфейс SPI. Микроконтроллер AT90S8515 конфигурируется как ведущее устройство, микросхема MAX5154 — ведомое устройство изначально, по своей сути.

Каждый из двух преобразователей имеет структуру типа R-2R, разрешение 12 разрядов и вывод с внутренним операционным усилителем для подачи напряжения (усиление напряжения 2 В/В). Подключение MAX5154 к микроконтроллеру AT90S8515 показано на рис. 16.32.

Время установления выходного напряжения нового уровня с точностью ±LSB после изменения на 4 В обычно составляет 15 мкс при скорости нарастания выходного напряжения усилителя 0,75 В/мкс. Уровень опорных напряжений **RefA** и **RefB** может находиться в диапазоне 0...(V_{CC}–1,4) В, а выходных аналоговых напряжений **OutA** и **OutB** — в диапазоне Gnd...V_{CC}. Питающее напряжение V_{CC}

может колебаться в диапазоне 4,5...5,5 В. Максимальное потребление тока составляет 650 мкА (типичное — 500 мкА). Встроенная схема сброса по включению питания, не требующая никакого внешнего монтажа, при подаче рабочего напряжения обнуляет все внутренние регистры.

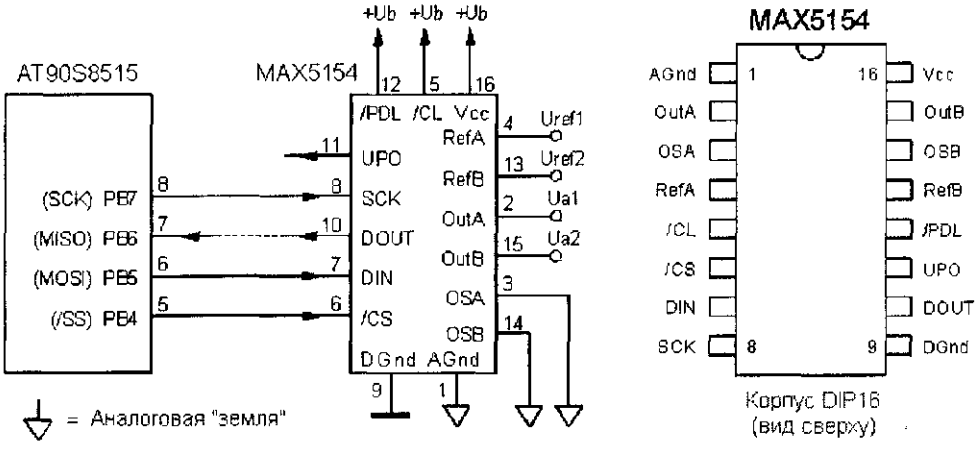


Рис. 16.32. Подключение микросхемы ЦАП MAX5154 к микроконтроллеру AT90S8515 через SPI

Как показано на функциональной схеме MAX5154 (рис. 16.33), каждый преобразователь содержит сдвоенный буферизованный вход, организованный в виде 12-разрядного входного регистра, за которым следует 12-разрядный регистр ЦАП и, наконец, схема R-2R. Содержимое всех регистров может обновляться (как по отдельности, так и одновременно) с помощью одной из 16-разрядных команд, перечисленных в табл. 16.4.

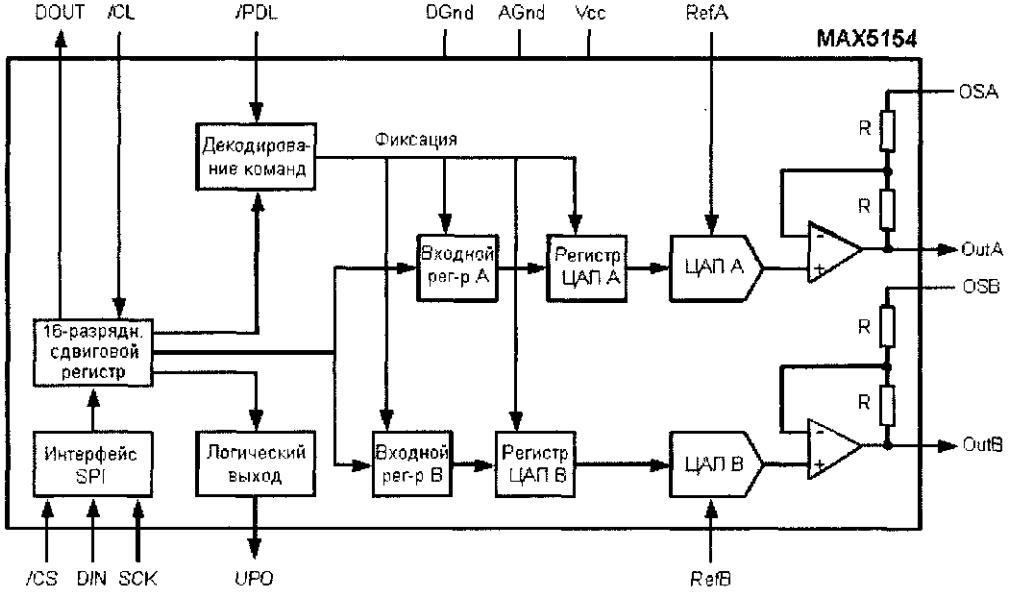


Рис. 16.33. Функциональная схема микросхемы ЦАМ MAX5154

В левой части рис. 16.34 показана стандартная конфигурация микросхемы MAX5154 в униполярном режиме работы. Если требуется смещение выходного напряжения, то между выводом **OSX** и аналоговой "землей" **AGnd** можно включить источник напряжения U_O (как показано в правой части рис. 16.34).

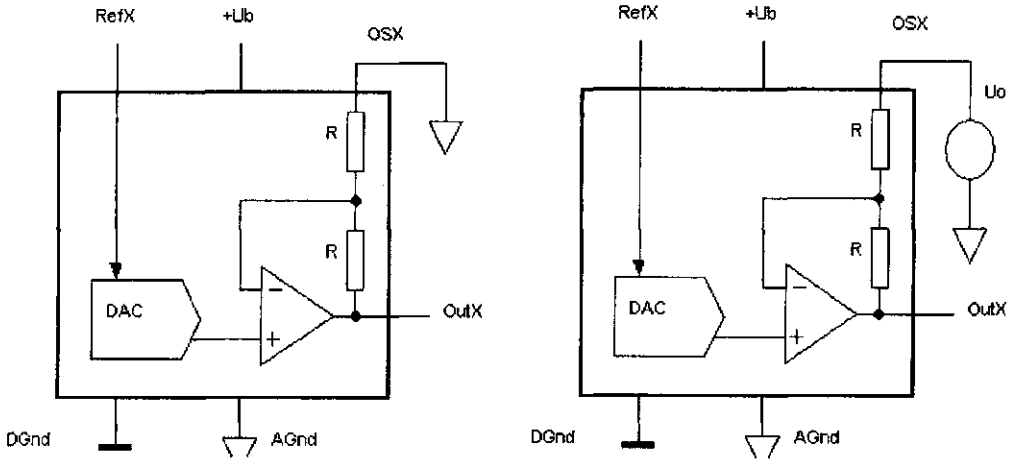


Рис. 16.34. Схема выхода микросхемы MAX5415 со смещением напряжения и без него

Обычно вывод OSX выходного усилителя каждого из преобразователей соединен с выводом AGnd. В этом случае выходное напряжение OutX преобразователя X (X = A или B):

$$OutX = 2 \cdot RefX \cdot Z / 1000_h = 2 \cdot RefX \cdot Z / 4096_d,$$

$$1\ LSB = 2 \cdot RefX / 1000_h = 2 \cdot RefX / 4096_d,$$

где Z — выражаемое в напряжении число, а RefX — опорное напряжение, поданное на соответствующий преобразователь.

Максимальный возможный уровень выходного напряжения для наибольшего шестнадцатеричного Z_{max}, которое можно представить с помощью двенадцати двоичных разрядов, составляет:

$$OutX_{max} = 2 \cdot RefX \cdot FFF_h / 1000_h = 2 \cdot RefX \cdot 4095_d / 4096_d,$$

Для того чтобы получить смещение выходного напряжения, вывод OSX (X = A или B) может быть соединен не только с выводом AGnd, но и со внешним источником напряжения U_O. В этом случае уровень выходного напряжения будет находиться в диапазоне $-U_O \ldots 2 \cdot RefX \cdot Z / 4096_d - U_O$.

Пусть, к примеру, опорное напряжение RefX составляет +2,048 В, а на вывод OSX подано напряжение U_O = –0,5 В. В этом случае диапазон значений для выходного напряжения составит +0,5... 4,595 В. При этом, само собой разумеется, выходное напряжение, которое зависит от ограничений рабочего напряжения, может лежать только в пределах $0 \ldots +U_b$!

С помощью соответствующих команд из табл. 16.4 оба ЦАП могут как по отдельности, так и одновременно быть переведены в режим останова, если на выводе **/PDL** (**P**ower **D**own **L**ockout — блокировка отключения питания) установлен

уровень лог. 1. Уровень лог. 0 на этом выводе запрещает аппаратное отключение, и соответствующие команды остаются безрезультатными.

В режиме останова потребляемый ток составляет в среднем 2 мкА (максимум 10 мкА), а выводы опорного напряжения и выходных усилителей переходят в высокоомное состояние. Интерфейс SPI остается активным, а данные хранятся во входных регистрах. Благодаря этому, микросхема MAX5154 при возвращении в обычный активный режим работы может принимать то же состояние на выходах, что и до останова. При выходе из режима останова ЦАП может возобновить работу или с помощью команды, использующей значение во входном регистре, или с помощью команды, загружающей новые данные (см. табл. 16.4).

Для возвращения в активный режим ЦАП требуется около 20 мкс, чтобы стабилизировалось напряжение на выходе. Кроме того, ЦАП может выйти из режима останова и вернуться к активному состоянию в результате изменения уровня на выводе /PDL с лог. 1 на лог. 0.

Последовательный интерфейс микросхемы MAX5154 совместим с первым вариантом стандарта SPI (CPHA = лог. 0, CPOL = лог. 0). Соответствующая временная диаграмма представлена в главе 7 на рис. 7.4. Частота тактовых сигналов SCK интерфейса SPI может составлять до 10 МГц.

Командное слово длиной 16 бит состоит из адресного разряда (A0) для выбора между ЦАП A и B, двух разрядов управления (C1, C0), двенадцати битов данных (D11...D0) и дополнительного разряда S0, который всегда содержит лог. 0. Формат данных SPI показан на рис. 16.35.

MSB ..... LSB														
16 последовательных бит данных														
Адрес	2 разряда управления		12 бит данных											Доп. бит
A0	C1	C0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	S0

Рис. 16.35. Формат последовательных данных микросхемы MAX5154

Если разряды A0, C1 и C0 содержат лог. 0, то биты D11...D8 становятся разрядами управления (табл. 16.4). Управляющее слово длиной 16 бит может передаваться целиком или разбиваться на два пакета по 8 бит каждый. На протяжении всей передачи на линии /CS должен быть уровень лог. 0. Временная диаграмма передачи данных в микросхему MAX5154 по интерфейсу SPI представлена на рис. 16.36.

Таблица 16.4. Обзор команд микросхемы MAX5154

A0	*C1*	*C0*	*D11...D0*	*S0*	*Описание*
0	0	1	12 бит данных ЦАП	0	Загрузка во входной регистр A, содержимое регистра ЦАП не изменяется
1	0	1	12 бит данных ЦАП	0	Загрузка во входной регистр B, содержимое регистра ЦАП не изменяется
0	1	0	12 бит данных ЦАП	0	Загрузка во входной регистр A, оба регистра ЦАП обновляются
1	1	0	12 бит данных ЦАП	0	Загрузка во входной регистр B, оба регистра ЦАП обновляются

Таблица 16.4. Окончание

A0	C1	C0	D11...D0	S0	Описание
0	1	1	12 бит данных ЦАП	0	В оба регистра ЦАП загружается содержимое сдвигового регистра (оба ЦАП одновременно начинают работать с новыми данными)
1	0	0	xxxx xxxx xxxx	0	В оба регистра ЦАП загружается содержимое соответствующих входных регистров (оба ЦАП одновременно начинают работать с хранимыми данными)
1	1	1	xxxx xxxx xxxx	0	Оба ЦАП переходят в режим останова (условие: /RDL = лог. 1)
0	0	0	001x xxxx xxxx	0	В регистр ЦАП А загружается содержимое входного регистра А (ЦАП А начинает работать с ранее записанными данными)
0	0	0	101x xxxx xxxx	0	В регистр ЦАП В загружается содержимое входного регистра В (ЦАП В начинает работать с ранее записанными данными)
0	0	0	110x xxxx xxxx	0	ЦАП А переходит в режим останова (условие: /RDL = лог. 1)
0	0	0	111x xxxx xxxx	0	ЦАП В переходит в режим останова (условие: /RDL = лог. 1)
0	0	0	010x xxxx xxxx	0	Выход UPO устанавливается в лог. 0 (по умолчанию)
0	0	0	011x xxxx xxxx	0	Выход UPO устанавливается в лог. 1
0	0	0	1000 xxxx xxxx	0	Режим 0: биты на выходе DOUT сдвигаются дальше по ниспадающему фронту сигнала SCK (по умолчанию)
0	0	0	1001 xxxx xxxx	0	Режим 1: биты на выходе DOUT сдвигаются дальше по нарастающему фронту сигнала SCK
0	0	0	000x xxxx xxxx	0	Без функции (NOP)

x = любое значение (лог. 0 или лог. 1)

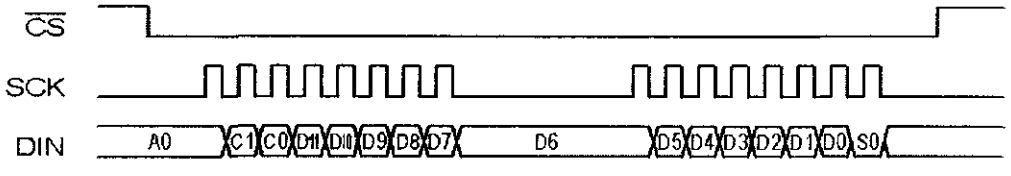


Рис. 16.36. Временная диаграмма передачи данных по интерфейсу SPI с паузой после первого байта

Ниспадающий фронт сигнала на выводе /CS переводит микросхему в режим приема данных. По нарастающему фронту импульсов SCK биты данных принимаются, а по ниспадающему — сдвигаются. По нарастающему фронту сигнала

/CS данные, в зависимости от значения переданных разрядов управления, записываются во входной регистр и/или в регистр ЦАП.

Благодаря двойной буферизации цифровых входов MAX5154, возможно загружать новые данные только во входной регистр без обновления регистра ЦАП или обновлять регистр ЦАП содержимым входного регистра, или одновременно обновлять входной регистр и регистр ЦАП.

Выход внутреннего 16-разрядного сдвигового регистра соединен с выводом **DOUT** — выходом с открытым стоком и внутренним подтягивающим сопротивлением. Последовательно соединяя выводы DOUT микросхем MAX5154 со входом DIN следующей микросхемы, можно получить так называемую гирляндную цепь (рис. 16.37).

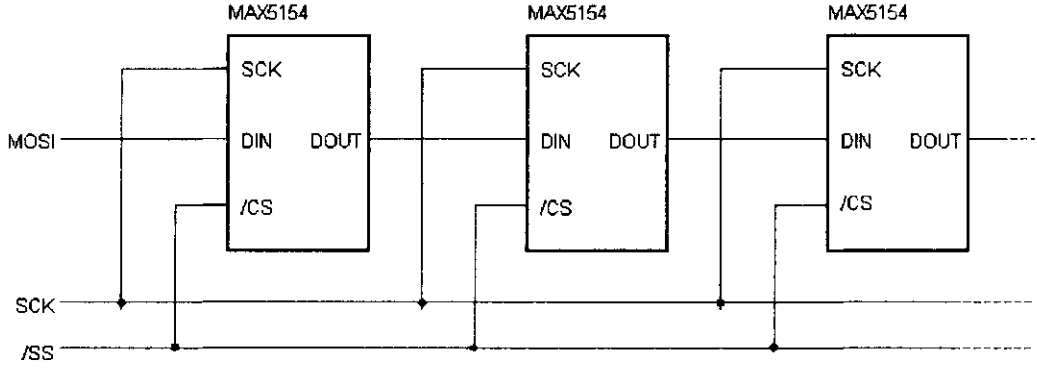


Рис. 16.37. Гирляндная цепь нескольких микросхем MAX5154

В противоположность способу подключения, показанному на рис. 7.2, где несколько микросхем подключаются к интерфейсу SPI параллельно, здесь для N устройств требуется только одна линия выбора, что приводит к экономии (N–1) линий /SS. При этом, однако, придется смириться с потерями в быстродействии, поскольку последовательный пакет данных всегда должен проходить через все N микросхем, даже если он предназначен только для одной из них.

Для добавления микросхем в гирляндную цепь, работающую согласно второму варианту протокола SPI, MAX5154 можно запрограммировать таким образом, чтобы биты передавались на выход DOUT по нарастающему фронту сигнала SCK (табл. 16.4, режим 1). После подачи питания автоматически устанавливается режим 0 (режим по умолчанию).

В случае с выводом **UPO** речь идет о цифровом выходе, который предоставляется в свободное распоряжение пользователя и может с помощью соответствующих команд из табл. 16.4 устанавливаться в лог. 0 или лог. 1. Значением по умолчанию, устанавливаемым при включении питания, является лог. 0.

Подав лог. 0 на вход **/CL** можно аппаратно обнулить все регистры микросхемы MAX5154. В этом случае выходное напряжение OutA и OutB принимает значение 0 В.

Для бесперебойной работы MAX5154 рекомендуется ограничить рабочее напряжение непосредственно на выводе V_{CC} с помощью конденсатора емкостью 4,7 мкФ и параллельно включенного с ним керамического конденсатора емкостью

0,1 мкФ. Для минимизации паразитных индуктивностей проводящий путь всегда следует выбирать как можно более коротким. Аналоговая "земля" всегда должна быть соединена с наиболее качественной точкой соединения с "землей" схемы. Оптимально использовать печатную плату, выполненную по многослойной технологии с непрерывной поверхностью "земли".

Входное сопротивление входов опорного напряжения зависит от изменений шестнадцатеричных значений в регистре ЦАП, потому выходное сопротивление источника опорного напряжения должно быть небольшим.

Представленный ниже пример программы демонстрирует управление микросхемой MAX5154 с помощью микроконтроллера AT90S8515. В цикле увеличивается и выводится в ЦАП А значение от $000 до $FFF. Сигнал выходного напряжения ЦАП А имеет пилообразную форму с линейно нарастающим передним фронтом в диапазоне 0 В ... (4095/4096) RefA. Аналогичным образом, уменьшается и выводится в ЦАП В значение от $FFF до $000. В результате сигнал выходного напряжения ЦАП В имеет пилообразную форму с линейно ниспадающим задним фронтом в диапазоне (4095/4096) RefA ... 0 В (рис. 16.38).

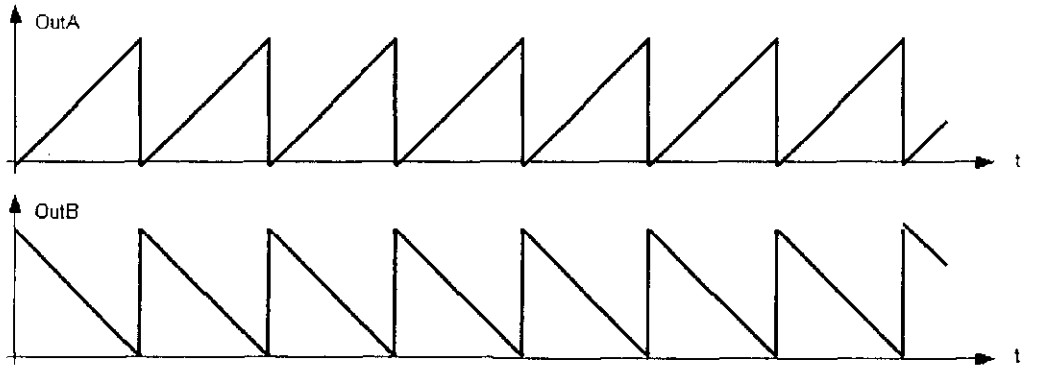


Рис. 16.38. Изменение выходного напряжения OutA и OutB в рассматриваемом примере

При каждом проходе цикла прежде всего выполняется загрузка во входной регистр ЦАП А без выдачи нового значения на выход.

- Формат команды = 0 0 1 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0 S0.

Затем выполняется загрузка во входной регистр ЦАП В.

- Формат команды = 1 1 0 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0 S0.

Наконец, оба ЦАП одновременно обновляют новое выходное значение.

Поскольку младший бит (S0) 16-разрядного командного слова всегда содержит лог. 0, при инкрементировании/декрементировании будем прибавлять/ вычитать константу 2.

В нашем распоряжении нет команды типа "Сложение с непосредственным значением", поэтому будем просто вычитать отрицательное значение константы $0002.

Переполнение счетчика из $FFF в $000 и наоборот уничтожает содержимое разрядов команды A0, C1, C0 (биты 13...15), поэтому после каждого прохода их содержимое следует восстанавливать.

В подпрограмме Output_SPI микросхема MAX5154 активизируется с помощью низкого уровня на линии /SS. Затем в регистр данных SPI, как только он освободится, должны быть загружены исключительно четыре выводимых байта. О завершении передачи сигнализирует флаг SPIF, который автоматически сбрасывается при следующем доступе к SPDR. Перед выходом из подпрограммы линия /SS опять деактивируется.

На выполнение этой подпрограммы при такте системной синхронизации 8 МГц требуется как раз 20 мкс, поэтому выходные значения обоих ЦАП наверняка достигают стабильного уровня при каждом проходе цикла.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\16105154.asm`

```
;**** Управление микросхемой MAX5154 через интерфейс SPI ******************
;*
;* К микроконтроллеру AT90S8515 по интерфейсу SPI подключена микросхема
;* MAX5154. На выходе ЦАП A получаем пилообразное напряжение с нарастающим
;* передним фронтом, а на выходе ЦАП B - пилообразное напряжение с
;* ниспадающим задним фронтом.
;* Тактовая частота микроконтроллера AVR: 4 МГц (Стандарт STK200).
;*
;*****************************************************************************
.nolist
.include "8515def.inc"
.list

.equ    DACa = $20      ; Загрузка входного регистра A, регистр ЦАП A неизменен
.equ    DACb = $C0      ; Загрузка входного регистра B, оба регистра ЦАП обновл.
.equ    nss = PB4       ; /SS = PB4

.def    OutL = r12      ; Младший байт для вывода через SPI
.def    OutH = r13      ; Старший байт для вывода через SPI
.def    InL = r14       ; Младший байт для ввода через SPI
.def    InH = r15       ; Старший байт для ввода через SPI
.def    temp = r16      ; Рабочий регистр
.def    A_lo = r24      ; Младший байт ЦАП A
.def    A_hi = r25      ; Старший байт ЦАП A
.def    B_lo = r26      ; Младший байт ЦАП B
.def    B_hi = r27      ; Старший байт ЦАП B

Reset:
000000 c00b    rjmp Initial

Output_SPI:                             ; 16 бит через SPI, CPHA=0,CPOL=0
000001 98c4    cbi PortB,nss            ; /SS = активный (лог. 0)
000002 b8df    out SPDR,OutH            ; Выводим старший байт
spi1:
000003 9b77    sbis SPSR,SPIF           ; Пропускаем следующую команду, если
                                        ; передача завершена
000004 cffe    rjmp spi1
000005 b0ff    in InH,SPDR              ; Запоминаем прочитанный байт
000006 b8cf    out SPDR,OutL            ; Выводим младший байт
spi2:
000007 9b77    sbis SPSR,SPIF           ; Пропускаем следующую команду, если
                                        ; передача завершена
000008 cffe    rjmp spi2
```

```
000009 b0ef    in InL,SPDR             ; Запоминаем прочитанный байт
00000a 9ac4    sbi PortB,nss           ; /SS = неактивный (лог. 1)
00000b 9508    ret

Initial:                               ; Часть инициализации
00000c e50f    ldi temp, Low(RAMEND)
00000d bf0d    out SPL, Temp
00000e e002    ldi temp, High(RAMEND)
00000f bf0e    out SPH, Temp           ; Инициализируем указатель стека
000010 e700    ldi temp,$70            ; SCK=0,MISO=PullUp,MOSI=1,/SS=1,ост.=HiZ
000011 bb08    out PortB,temp          ; Выводим состояние простоя
000012 eb00    ldi temp,$B0            ; SCK,MOSI,/SS = выход, остальные = входы
000013 bb07    out DDRB,temp           ; Конфигурируем направление передачи
                                       ; данных через порт B
000014 e500    ldi temp,$50            ; SPIE=0,SPE=1,DORD=0, MSTR=1,CPHA=0,
000015 b90d    out SPCR,temp           ; CPOL=0, f(SCK)=f(Сист.) / 4
000016 e080    ldi A_lo,$00
000017 e090    ldi A_hi,$00            ; Инициализируем счетчик A нулем
000018 e0a0    ldi B_lo,$00
000019 e0b0    ldi B_hi,$00            ; Инициализируем счетчик B нулем
Loop:
00001a 9602    adiw A_lo,2             ; Инкрементируем на 1 регистр двойной
                                       ; длины A (2 - потому что S0=0)
00001b 719f    cbr A_hi,$E0            ; Обнуляем разряды 15..13 (переполнение)
00001c 6290    sbr A_hi,DACa           ; Устанавливаем разряды управл-я для ЦАП A
00001d 2ec8    mov OutL,A_lo
00001e 2ed9    mov OutH,A_hi
00001f dfe1    rcall Output_SPI        ; Выводим в ЦАП A 16 бит через SPI
000020 9712    sbiw B_lo,2             ; Декрементируем на 1 регистр двойной
                                       ; длины B (2 - потому что S0=0)
000021 71bf    cbr B_hi,$E0            ; Обнуляем разряды 15..13 (переполнение)
000022 6cb0    sbr B_hi,DACb           ; Устанавливаем разряды управл-я для ЦАП B
000023 2eca    mov OutL,B_lo
000024 2edb    mov OutH,B_hi
000025 dfdb    rcall Output_SPI        ; Выводим в ЦАП B 16 бит через SPI
000026 cff3    rjmp Loop
```

Расширение портов ввода/вывода микроконтроллера AT90S4414 с помощью интерфейса SPI

Хотя микроконтроллеры AVR, оснащенные интерфейсом SPI, и имеют в своем распоряжении сразу четыре восьмиразрядных порта, через которые они могут напрямую обмениваться данными со внешним миром, в некоторых случаях применения даже их может не хватить. Эта глава демонстрирует, как без лишних затрат увеличить число вводов/выводов. Принцип расширения портов проиллюстрирован рис. 16.39.

В качестве входов можно воспользоваться сдвиговыми регистрами In1 ... InN типа 74HC165, в которые по короткому импульсу, сигнализирующему о начале передачи, параллельно загружается информация для временного хранения. Затем их содержимое сдвигается для последовательной передачи в микроконтроллер AVR по интерфейсу SPI.

В качестве выходов используются сдвиговые регистры Out1 ... OutN типа 74HC595, в которые по нарастающему фронту сигнала /SS на выводе PB4 последовательно заносятся данные, полученные от микроконтроллера по интерфейсу SPI.

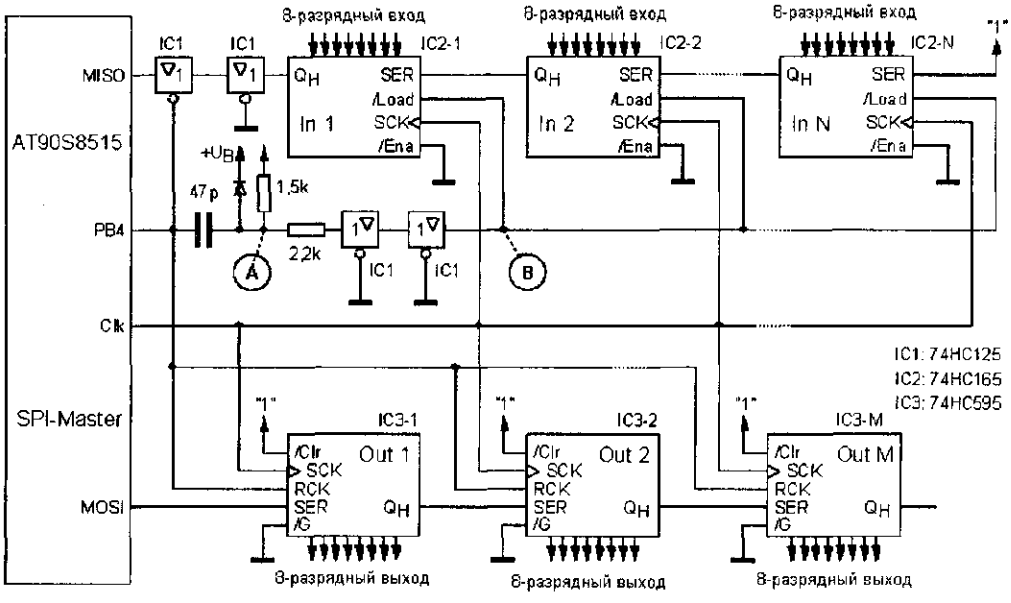


Рис. 16.39. Принцип расширения портов с помощью интерфейса SPI

Временная диаграмма передачи данных показана на рис. 16.40.

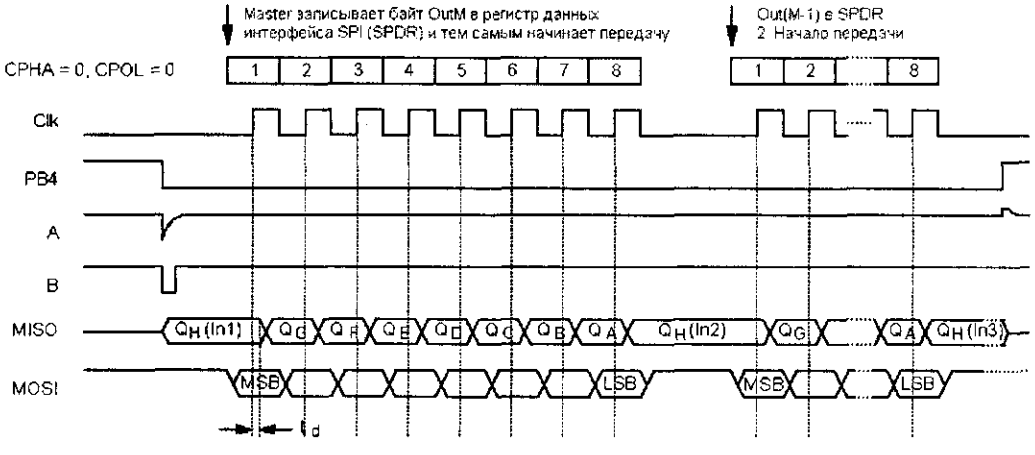


Рис. 16.40. Временная диаграмма передачи данных через интерфейс SPI

Сигнал /SS, связанный с выходом MISO микроконтроллера AVR, переключает выход драйвера 74HC125 из тристабильного в активное состояние. В результате на вход MISO поступает старший разряд байта, хранимого в этот момент в регистре In1. Если выбрать протокол SPI вида CPHA=0, CPOL=0, то биты будут сдвигаться по ниспадающему, а приниматься — по нарастающему фронту тактового

сигнала. Микросхема 74HC165 сдвигает свое содержимое по нарастающему фронту тактового сигнала. Задержка t_d до поступления очередного бита, обусловленная включением между выходом In1 и входом MISO микроконтроллера AVR двух драйверов 74HC125, достаточна для обеспечения надежного приема по нарастающему фронту.

С целью тестирования была разработана и подключена через порт B к модулю STK200 плата, соответствующая схеме, показанной на рис. 16.41.

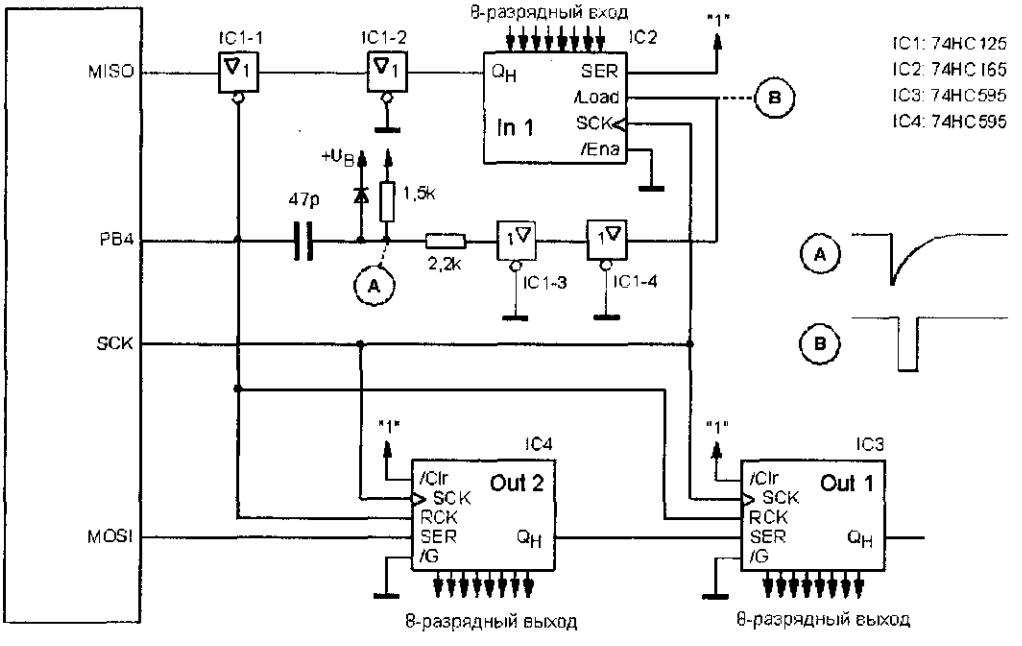


Рис. 16.41. Тестовая схема для проверки взаимодействия с модулем STK200

Для того чтобы можно было осуществлять функциональный контроль с помощью представленной ниже программы, выводы Out2 в тестовой схеме были напрямую соединены со входами In1. Таким образом, байт, выводимый в некотором цикле через Out2, в следующем цикле опять считывался через In1.

Количество входных сдвиговых регистров не обязательно должно совпадать с количеством выходных сдвиговых регистров. Единственное требование заключается только в том, что за один цикл должно передаваться количество байтов, заданое максимальным количеством регистров того или иного вида.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\1611shft.asm`

```
;**** Расширение портов AVR с помощью интерфейса SPI *********************
;*
;* Ширина портов микроконтроллера AT90S4414 увеличивается на
;* 16 выходов и 8 входов.
;* Тактовая частота микроконтроллера AVR: 4 МГц (Стандарт STK200).
;*
;***************************************************************************
.nolist
.def    Last = r15              ; Промежуточный буфер
```

```
.def    tmp1 = r16              ; Рабочий регистр 1
.def    Out1 = r17              ; Выходное значение для выхода 1
.def    Out2 = r18              ; Выходное значение для выхода 2
.def    In1 = r19               ; Входное значение для входа 1
.equ    nss = PB4               ; /SS = PB4
.equ    Latch = PB3             ; Импульс фиксации = PB3
.equ    okay = PB1              ; LED1 для порта B в случае корректной передачи
.equ    LEDK = PB0              ; LED0 для порта B в случае нажатия кнопки 0
.equ    time = 117              ; T = время * 1024 / 4 МГц = 30 мс

Reset:
000000 c025    rjmp Initial

WaitTime:                               ; Задержка на 30 мс
000001 e80b    ldi tmp1,-time           ; Загружаем интервал таймера
000002 bf02    out TCNT0,tmp1           ; Инициализируем счетчик
000003 e002    ldi tmp1,1<<TOV0         ; Загружаем битовую позицию флага TOV0
000004 bf08    out TIFR,tmp1            ; Сбрасываем флаг TOV0
WT1:
000005 b708    in tmp1,TIFR             ; Извлекаем содержимое регистра TIFR
000006 ff01    sbrs tmp1,TOV0           ; Пропускаем следующую команду, если
                                        ; возникает переполнение T/C0
000007 cffd    rjmp WT1                 ; Дальше ожидаем переполнения
000008 9508    ret

WaitKey:                                ; Ожидаем нажатия кнопки 0
000009 9980    sbic PinD,0              ; Пропускаем следующую команду, если
                                        ; PD0=0 (нажата кнопка)
00000a cffe    rjmp WaitKey             ; Ожидаем дальше
00000b dff5    rcall WaitTime           ; Задержка на 30 мс (учет дребезга конт.)
00000c 9980    sbic PinD,0              ; Пропускаем следующую команду, если
                                        ; кнопка еще нажата
00000d cffb    rjmp WaitKey             ; Ожидаем дальше
WK2:
00000e 9b80    sbis PinD,0              ; Пропускаем следующую команду, если
                                        ; PD0=1 (кнопка отпущена)
00000f cffe    rjmp WK2                 ; Ожидаем дальше
000010 dff0    rcall WaitTime           ; Задержка на 30 мс (учет дребезга конт.)
000011 9b80    sbis PinD,0              ; Пропускаем следующую команду, если
                                        ; кнопка еще отпущена
000012 cffb    rjmp WK2                 ; Ожидаем дальше
000013 9508    ret

ToggleLED:                              ; Переключение светодиода LED0
000014 9bb8    sbis DDRB,LEDK           ; Пропускаем следующую команду, если
                                        ; светодиод, соотв. кнопке, включен
000015 c002    rjmp TL1
000016 98b8    cbi DDRB,LEDK            ; Отключаем светодиод
000017 9508    ret
TL1:
000018 9ab8    sbi DDRB,LEDK            ; Включаем светодиод
000019 9508    ret

SPITransf:                              ; 16 бит через SPI, CPHA=0,CPOL=0
00001a 98c4    cbi PortB,nss            ; /SS = активный (лог. 0)
00001b 98c3    cbi PortB,Latch          ; Импульс фиксации (лог. 0)
```

```
00001c 9ac3    sbi PortB,Latch          ; Выводим
00001d b92f    out SPDR,Out2            ; Выводим Out2
spi1:
00001e 9b77    sbis SPSR,SPIF           ; Пропускаем следующую команду, если
                                        ; передача завершена
00001f cffe    rjmp spi1
000020 b13f    in In1,SPDR              ; Считываем вход 1
000021 b91f    out SPDR,Out1            ; Выдаем Out1
spi2:
000022 9b77    sbis SPSR,SPIF           ; Пропускаем следующую команду, если
                                        ; передача завершена
000023 cffe    rjmp spi2
000024 9ac4    sbi PortB,nss            ; /SS = неактивный (лог. 1)
000025 9508    ret

Initial:
000026 e50f    ldi tmp1,Low(RAMEND)
000027 bf0d    out SPL,tmp1
000028 e001    ldi tmp1,High(RAMEND)
000029 bf0e    out SPH,tmp1             ; Инициализация указателя стека
00002a e70c    ldi tmp1,$7C             ; SCK=0,MISO=нагруз. сопр.,MOSI=1,/SS=1,
00002b bb08    out PortB,tmp1           ; Фикс.=1,Бит2=нагруз. сопр.,LED1,LED0=0
00002c eb08    ldi tmp1,$B8             ; SCK,MOSI,/SS,Фиксация
00002d bb07    out DDRB,tmp1            ; LED1,LED0 = выход,MISO, разряд 2 = вход
00002e e000    ldi tmp1,$00             ; Обнуляем все разряды
00002f bb02    out PortD,tmp1           ; Все разряды = HiZ (кнопки)
000030 bb01    out DDRD,tmp1            ; Направление передачи для порта D = вход
000031 e500    ldi tmp1,$50             ; SPIE=0,SPE=1,DORD=0,MSTR=1,
000032 b90d    out SPCR,tmp1            ; CPHA=0,CPOL=0, f(SCK)=f(Сист.) / 4
000033 e005    ldi tmp1,5               ; Настройка делителя частоты
000034 bf03    out TCCR0,tmp1           ; Запускаем таймер, коэф. деления = 1024
000035 2722    clr Out2                 ; Инициализируем выходное значение

Haupt:
000036 dfd2    rcall WaitKey            ; Ожидаем нажатия кнопки 0
000037 dfdc    rcall ToggleLED          ; переключаем светодиод 0
000038 9523    inc Out2                 ; Инкрементируем выходное значение
000039 2f12    mov Out1,Out2            ; Копируем в Out1
00003a 9510    com Out1                 ; Дополнение до единицы
00003b dfde    rcall SPITransf          ; Передача через SPI: Out1, Out2 и In1
00003c 9ab9    sbi DDRB,okay            ; В случае корректной передачи включаем
                                        ; светодиод 1
00003d 113f    cpse In1,Last            ; Сравниваем входной байт с байтом,
                                        ; полученным ранее на выходе
00003e 98b9    cbi DDRB,okay            ; Если неравно, включаем светодиод 1
00003f 2ef2    mov Last,Out2            ; Сохраняем текущее значение Out2
000040 cff5    rjmp Haupt
```

Описание подпрограммы

Часть `WaitTime` реализует задержку на 30 мс. За это время в таймер T/C0 загружается соответствующее значение, и сбрасывается флаг переполнения. Как только этот флаг опять устанавливается, требуемый промежуток времени истекает.

В части `WaitKey` реализовано ожидание нажатия кнопки на выводе PD0. Как только кнопка будет нажата и опять отпущена, последует возврат к вызывающей программе.

Часть `ToggleLED` при каждом обращении переключает светодиод LEDK на выводе PB0, чтобы сигнализировать о том, что было распознано нажатие кнопки.

Часть `SPITransf` передает через интерфейс SPI 16 бит данных. После того как по выводу PB4 активируется линия /SS, на вывод PB3 выдается импульс фиксации, который можно было бы подат на сдвиговой регистр In1 в качестве строб-импульса альтернативно дифференцированному фронту сигнала /SS (пункт A на рис. 16.41). В этом случае конденсатор, диод и два резистора выпадают. В тестовой схеме такие импульсы фиксации не используются.

Далее выдается первый байт Out2, и ожидается появление флага SPIF, сигнализирующего о завершении передачи. Байт для записи в In1, считываемый параллельно при пересылке Out2, может быть считан из буфера приема SPDR и записан в одноименную переменную In1. За этим следует пересылка Out1. Опять параллельно считывается сдвиговой регистр In1, однако, поскольку на последовательный вход SER постоянно подается лог. 1, в качестве второго байта всегда будет получено значение $FF. В завершение передачи по интерфейсу SPI линия /SS переходит в состояние покоя (лог. 1). Линия /SS соединяется со входом RCK (**R**egis-ter **C**lock — синхронизация регистра) сдвиговых регистров IC3 и IC4. По нарастающему фронту байты, принятые непосредственно в сдвиговые регистры, передаются в выходной регистр.

Описание главной программы

После сброса по включению питания происходит ветвление по адресу $000 — к части инициализации, обозначенной меткой `Initial`. После установки указателя стека разряды SCK, MOSI, /SS и Latch интерфейса SPI должны быть определены как выходы, а MISO — как вход. Порты PB0 и PB1, соединенные с двумя светодиодами, объявляются как входы, поскольку в процессе выполнения программы они с помощью регистра направления передачи данных DDRB настроены как входы с подтягивающим сопротивлением и открытым коллектором.

Инициализация порта D выполняется только для полноты примера, поскольку этот порт и регистр DDRD в результате внутренних процессов, имевших место при поступлении сигнала сброса по включению питания, уже инициализированы значением $00, и, следовательно, порт D объявлен как вход.

В регистре управления SPI обнуляются разряды SPIE, DORD, CPOL, CPHA, SPR1 и SPR0, а также устанавливаются в лог. 1 разряды SPE и MSTR. В результате интерфейс сконфигурирован как ведущее устройство SPI , и ему разрешено передавать свои данные с помощью четверти такта системной синхронизации в соответствии с первым случаем протокола передачи, описанным в главе 7. Прерывание от SPI не требуется, и потому запрещено.

Когда поступает сигнал сброса, в регистр TCCR0 автоматически загружается значение $00, тем самым останавливая таймер. Для того чтобы его запустить, с помощью мультиплексора в качестве входного такта для T/C0 выбирается коэффициент деления такта системной синхронизации 1024. Для этого в регистр

TCCR0 записывается значение $05 (см. табл. 4.2), и T/C0 начинает счет. В завершение части инициализации выходное значение `Out2` инициализируется нулем.

В главной части программы (метка `Haupt`) ожидается появление импульса от кнопки на выводе PD0. После срабатывания кнопки зажегшийся светодиод `LEDK` извещает пользователя о том, что нажатие было распознано. Затем выходное значение `Out2` инкрементируется, копируется в `Out1` и там инвертируется. Теперь `Out1` содержит дополнение до единицы значения `Out2`. Далее оба значения выводятся с помощью подпрограммы `SPITransf`. Об успешном завершении передачи сигнализирует светодиод `okay`. Это реализовано с помощью сравнения переменной `Last`, хранящей последнее выведенное значение `Out2`, с переменной `In1`. Если передача была выполнена корректно, то светодиод `okay` продолжает гореть, в противном случае — гаснет.

Первое сравнение `In1` и `Last` будет неопределенным, поскольку по первому импульсу фиксации, поданному на вход /Load, в In1 записывается некоторое случайное значение, однако, уже начиная со второго нажатия кнопки, это сравнение будет информативным. Перед тем как бесконечный цикл будет завершен переходом к метке `Haupt`, текущее значение `Out2` сохраняется в переменной `Last` для последующего прохода цикла.

При тестировании программы с помощью набора STK200 было выбрано изменение выходных значений по нажатию кнопки, поскольку таким образом в состоянии покоя можно контролировать уровень передаваемых битов на выводах IC3 и IC4.

Программная реализация интерфейса SPI для модели AT90S1200 при подключении АЦП

Модели AT90S1200 и AT90S2313 не оснащены аппаратной реализацией интерфейса SPI, тем не менее их также можно использовать в качестве ведущих устройств периферийных модулей с помощью программной реализации SPI, которая и будет рассмотрена в данной главе.

В качестве примера рассмотрим подключение аналого-цифрового преобразователя (АЦП) TLV1572 от компании Texas Instruments разрешением 10 бит, оснащенного встроенным интерфейсом SPI.

Этот преобразователь работает по методу последовательных приближений, содержит интегрированную схему выборки и хранения, а также интерфейс SPI, соответствующий третьему случаю протокола передачи (CPHA = лог. 1, CPOL = лог. 0). Частота следования тактов SPI на выводе SCK может достигать 20 МГц при питающем напряжении $+U_b$ = 5 В или 10 МГц при $+U_b$ = 3 В. Опорное напряжение U_{ref} может находиться в диапазоне +3 В ... +Ub, а оцифровываемое аналоговое напряжение на входе U_x — в диапазоне Gnd...Uref.

Подключение микросхемы TLV1572 к микроконтроллеру AT90S1200 показано на рис. 16.42.

Входам/выходам SCK, MOSI, MISO и /SS соответствуют обычные порты ввода/вывода микроконтроллера AT90S1200, а функция SPI этих выводов будет смоделирована программно (набор PD2...PD5 выбран случайным образом).

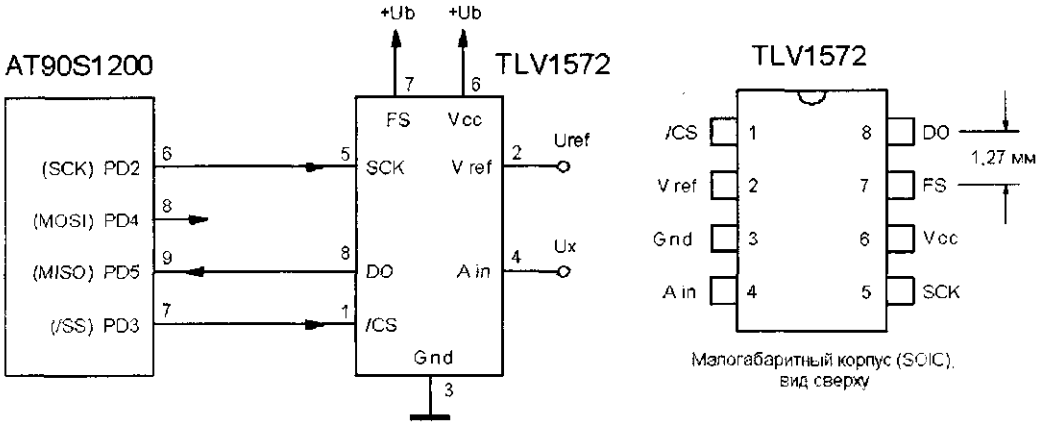


Рис. 16.42. Подключение АЦП TLV1572 к микроконтроллеру AT90S1200

Шестнадцатеричное число Z, которое будет получено на выходе TLV1572 в результате преобразования, находится в следующей зависимости от U_{ref} и U_x (при разрешении 10 бит):

$$Z = 400_h \cdot U_x / U_{ref} = 1024_d \cdot U_x / U_{ref},$$

$$1\ LSB = U_{ref} / 400_h = U_{ref} / 1024_d.$$

Таким образом, переполнение не возникает, пока $U_x \le (U_{ref} - 1\ LSB)$ или $U_x \le (1023/1024)\ U_{ref}$. Так, к примеру, величинам U_{ref} = 5,12 В, 1 LSB = 5 мВ, и наибольшему напряжению на входе U_x = 5,115 В, соответствует числовое значение $Z = 3FF_h = 1023_d$.

Вход FS микросхемы TLV1572 задает протокол передачи: SPI или DSP. Если по ниспадающему фронту сигнала /CS на входе FS считывается высокий уровень, то передача выполняется по протоколу SPI.

Цикл преобразования в обычном случае занимает 16 тактов SCK и начинается по нарастающему фронту первого тактового импульса, поступившего после ниспадающего фронта сигнала /CS или /SS (рис. 16.43). С этого первого тактового импульса начинается этап выборки (дискретизации), во время которого интегрированный конденсатор внутренней схемы выборки и хранения заряжается от аналогового напряжения U_x.

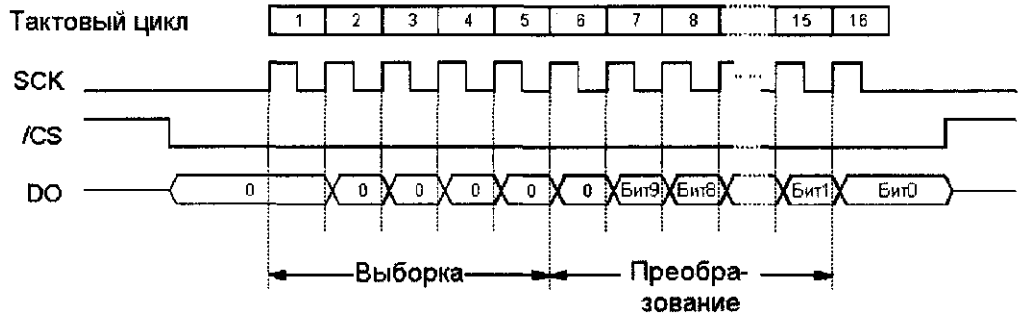


Рис. 16.43. Принцип действия АЦП TLV1572

Первые шесть битов на выходе DO, который соответствует выходу подчиненного устройства MISO, всегда содержат лог. 0, а собственно информация содержится в разрядах 7...16. Биты на выход DO выдаются (начиная со старшего) по нарастающему фронту сигнала SCK, а по ниспадающему фронту принимаются ведущим устройством.

В том случае, если требуется разрешение меньше 10 бит, ведущее устройство во время передачи может выставить сигнал высокого уровня на линию /CS. Например, для восьмиразрядного АЦП передача завершается после 14 тактовых импульсов появлением на входе /CS лог. 1. Следующий ниспадающий фронт сигнала /CS переводит микросхему TLV1572 в исходное состояние, и инициализирует ее для следующего преобразования.

Тактовый сигнал сдвига SCK интерфейса SPI одновременно выполняет роль тактового сигнала для внутреннего регистра SAR (**S**uccessive **A**pproximation **R**egister — регистр последовательных приближений). Решение о том, какой уровень назначить биту в результате приближения: лог. 0 или лог. 1 — принимается по нарастающему фронту сигнала SCK. Во избежание влияния помех в тактовом импульсе, бит данных появляется на выходе DO микросхемы TLV1572 с задержкой относительно нарастающего фронта сигнала SCK.

В обычном случае преобразование занимает 16 тактовых сигналов SCK. Ведущее устройство SPI, соединенное с АЦП TLV1572, может за раз передавать только восемь бит, поэтому вначале передаются первые восемь разрядов, а затем — следующие восемь. В этом случае в линии /CS во время передачи всех 16 битов должен быть низкий уровень сигнала (рис. 16.44).

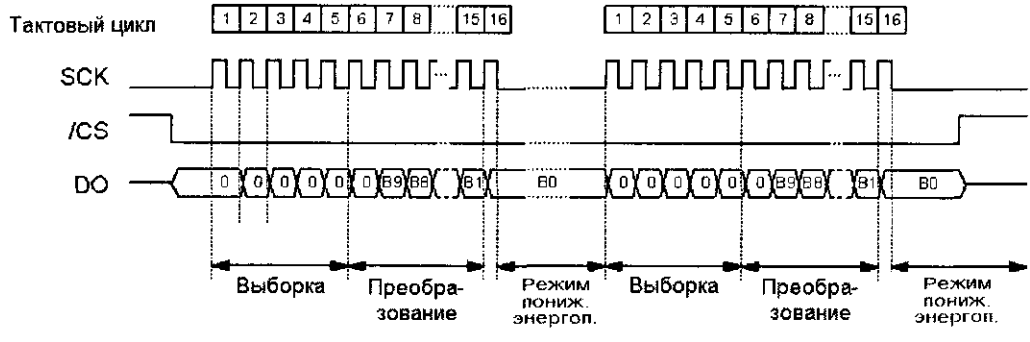


Рис. 16.44. Временная диаграмма двух последовательных аналого-цифровых преобразований с автоматическим переходом в режим пониженного энергопотребления после передачи LSB

Перерыв между двумя восьмибитными пакетами не должен превышать 100 мкс, иначе уровень напряжения на конденсаторе схемы выборки и хранения станет слишком низким.

После того как по ниспадающему фронту тактового импульса передан младший разряд, микросхема автоматически переходит в режим пониженного энергопотребления, из которого она выходит по нарастающему фронту следующего тактового импульса. В режиме пониженного энергопотребления микросхема потребляет ток не более 10 мкА (в активном режиме — максимум 8,5 мА, обычно — 5,5 мА). Когда в линии /CS появляется лог. 1, а микросхема находится в режиме

пониженного энергопотребления, вывод DO переходит в тристабильное состояние.

Для того чтобы представленный ниже программный пример можно было использовать и для других случаев применения, был реализован выход MOSI ведущего устройства, хотя для подключения микросхемы TLV1572 этого не требуется (данные только передаются ведущему устройству, а обратно ничего не принимается).

Имя файла на прилагаемом к книге компакт-диске: `\Programm\16121572.asm`

```
;**** Программная реализация SPI для микроконтроллера AT90S1200 **********
;*
;* Интерфейс SPI реализуется программно для подключения АЦП TLV1572.
;*
;* Тактовая частота микроконтроллеров AVR: 4 МГц (стандарт STK200)
;*
;*****************************************************************************
.nolist
.include "1200def.inc"
.list

.equ    sck = PD2               ; Shift-Clock = порт D / разряд 2
.equ    nss = PD3               ; /SS = порт D / разряд 3
.equ    mosi = PD4              ; MOSI = порт D / разряд 4
.equ    miso = PD5              ; MISO = порт D / разряд 5

.def    spi_lo = r14            ; Младший байт результата АЦ преобразования
.def    spi_hi = r15            ; Старший байт результата АЦ преобразования
.def    temp = r16              ; Рабочий регистр

Reset:
000000 c023    rjmp Init_SPI
Read_Write_SPI:                         ; Запись/чтение 16 бит через SPI
000001 9893    cbi PortD,nss            ; /CS АЦП 1572 = активный (лог. 0)
000002 e100    ldi temp,16              ; Инициализируем счетчик цикла числом 16
spi_loop:
000003 0cee    lsl spi_lo               ; Сдвиг влево млад. байта, D0=лог.0, D7->C
000004 1cff    rol spi_hi               ; Сдвиг влево старш. байта, C->D8, D15->C
000005 f410    brcc low_mosi            ; Переход, если выводимый бит = лог. 0
000006 9a94    sbi PortD,mosi           ; MOSI = лог. 1
000007 c002    rjmp sck_high            ; На этом завершается высокий уровень SCK
low_mosi:
000008 9894    cbi PortD,mosi           ; MOSI = лог. 0
000009 0000    nop                      ; Задержка, пока бит не станет стабильным
sck_high:
00000a 9a92    sbi PortD,sck            ; Нарастающий фронт SCK
00000b 0000    nop                      ; Задержка
sck_low:
00000c 9892    cbi PortD,sck            ; Ниспадающий фронт SCK
00000d 0000    nop                      ; Теперь бит данных на входе MISO стабилен
00000e 9985    sbic PinD,miso           ; Пропускаем следующую команду, если
                                        ; бит MISO = лог. 0
00000f 94e3    inc spi_lo               ; Когда D0=0 (lsl), устан. "inc spi_lo" D0
000010 950a    dec temp                 ; Счетчик цикла - 1
000011 f789    brne spi_loop            ; Следующий бит, если temp не равно 0
```

```
000012 9a94    sbi PortD,mosi          ; MOSI переходит в состояние покоя (лог.1)
000013 9a93    sbi PortD,nss           ; /SS = неактивный (лог. 1)
000014 9508    ret

Auswertung:                            ; Вывод результата
000015 94f6    lsr spi_hi              ; Сдвиг вправо старшего байта,
                                       ; D15 = лог.0, D8->C
000016 94e7    ror spi_lo              ; Сдвиг вправо младш. байта, C->D7, D0->C
000017 9896    cbi PortD,6             ; Младший разряд результата - в порт D,
                                       ; разряд 6 = 0
000018 f408    brcc Auswert1           ; Переход, если младший разряд
                                       ; действительно = лог. 0
000019 9a96    sbi PortD,6             ; Младший разряд результата - в порт D,
                                       ; разряд 6 = 1
Auswert1:
00001a bae7    out DDRB,spi_lo         ; Разряды 1...8 результата - в порт B
00001b 94f6    lsr spi_hi              ; Сдвиг вправо старшего байта,
                                       ; D15 = лог. 0, MSB->C
00001c 9890    cbi PortD,0             ; Старший разряд результата - в порт D,
                                       ; разряд 0 = 0
00001d f408    brcc Auswert2           ; Переход, если старший разряд
                                       ; действительно = лог. 0
00001e 9a90    sbi PortD,0             ; Старший разряд результата - в порт D,
                                       ; разряд 0 = 1
Auswert2:
00001f 9508    ret
Init_SPI:                              ; CPHA = 1, CPOL = 0
000020 e000    ldi temp,$00            ; Все разряды = 0
000021 bb08    out PortB,temp          ; Все выводы порта B - низкий уровень
000022 bb07    out DDRB,temp           ; Выводы драйвера B - все высокоомные
000023 e308    ldi temp,$38            ; Разряды 3...5 = 1, остальные = 0
000024 bb02    out PortD,temp          ; /SS,MOSI,MISO = лог.1, SCK,ост. = лог.0
000025 ed0f    ldi temp,$DF            ; Разряд 5 = 0, остальные = 1
000026 bb01    out DDRD,temp           ; PD5 (MISO) = вход, остальные = выходы
Haupt:
000027 e605    ldi temp,$65            ; Условно для выводимого слова
000028 2ee0    mov spi_lo,temp         ; Инициализируем млад. байт вых. регистра
000029 2ef0    mov spi_hi,temp         ; Инициализируем старш. байт вых. регистра
00002a dfd6    rcall Read_Write_SPI    ; Ввод/вывод 16 бит через SPI
00002b dfe9    rcall Auswertung        ; Обрабатываем результат АЦ преобразования
00002c cffa    rjmp Haupt
```

Поскольку микросхема TLV1572 при питающем напряжении U_b = +5 В может работать на максимальной тактовой частоте 20 МГц, в представленной выше программе не требуется учитывать длительность сигналов высокого и низкого уровня на входе SCK как в том случае, когда в роли ведущего устройства выступает микроконтроллер AT90S1200 с максимальной тактовой частотой 12 МГц. Если к рассмотренному программному интерфейсу необходимо подключить более медленнодействующее периферийное устройство, то можно увеличить фазу высокого и низкого уровней сигнала SCK с помощью дополнительных команд nop или вставки цикла ожидания.

Описание программы

После сброса по включению питания следует часть инициализации, в которой задается направление передачи данных и инициализируются необходимые порты.

В главной программе в бесконечном цикле через интерфейс SPI выдается условное значение $6565, и одновременно с этим считывается 10-разрядное значение из АЦП TLV1572 (разряды 0...7 — в `spi_lo`, разряды 8...9 — в `spi_hi`), которое затем с целью проверки выводится через порты микроконтроллера AT90S1200. При этом младший разряд выдается на вывод 6 порта D, разряды 1...8 — на выводы 0...7 порта B, а старший разряд — на вывод 0 порта D.

Поскольку разрядам порта B соответствуют выводы с открытым коллектором (см. рис. 10.3), то для подсоединения светодиодов перемычка "JP-PortB" модуля STK200 должна быть установлена в положение "Quasi – Pull Ups". В этом случае светодиоды будут одновременно служить для оптического отображения значений разрядов 1...8. Подключение светодиодов к портам PD0 и PD6 в модуле STK200 не предусмотрено, поэтому логические уровни разрядов 0 и 9 результирующей величины можно проверить только с помощью какого-либо измерительного прибора.

Подпрограмма `Read_Write_SPI` выводит 16 бит через интерфейс SPI, а затем подобным же образом считывает их. Принятые данные после выхода из подпрограммы остаются в регистрах `spi_lo` и `spi_hi`, в которые они были записаны при пересылке. Если в некотором случае применения требуется передать только восемь бит, достаточно одного регистра `spi_lo` (в этом случае счетчик цикла `temp` должен быть инициализирован значением 8).

В начале активируется линия /SS (лог. 0). При каждом проходе цикла один разряд регистровой пары `spi_hi` / `spi_lo` (начиная со старшего) сдвигается влево, сохраняясь в виде флага переноса, а затем выдается на выход MOSI. После этого следует нарастающий фронт сигнала SCK, по которому ведомая микросхема TLV1572 выдает текущий бит на выход DO. По окончанию фазы высокого уровня сигнала SCK (ниспадающий фронт) и выполнения фиктивной команды задержки ведущий микроконтроллер AT9S1200 принимает этот бит на своем входе MISO.

Поскольку в результате операции сдвига младший разряд регистровой пары `spi_lo` /`spi_hi` (разряд 0 `spi_lo`) заполняется лог. 0, то лог. 1, считанную по ниспадающему фронту сигнала SCK, можно легко записать в младший разряд с помощью команды `inc`. В том случае, если был считан лог. 0, младший разряд регистровой пары остается без изменений.

После шестнадцати проходов цикла оба байта будут выведены или прочитаны, линия MOSI перейдет в состояние покоя (лог. 1), а линия /SS переключится в неактивное состояние и произойдет возврат в вызывающую программу.

Эта программа не может быть проверена в какой-либо реальной схеме, а только с помощью оболочки AVR-Studio, так как на момент издания книги микросхема TLV1572 уже была снята с производства. Тем не менее, автор решил использовать этот пример, поскольку чисто программная реализация интерфейса SPI, несомненно, будет интересна многим читателям.

Подключение к шине I²C датчика температуры LM75

LM75 — это датчик температуры, который в течение 100 мкс с помощью интегрированного сигма-дельта АЦП преобразовывает измеренную температуру в цифровое девятиразрядное представление, и затем выводит его через встроенный интерфейс I²C.

Кроме того, LM75 имеет в своем распоряжении блок контроля за температурой, который через вывод OS (**O**vertemperature **S**hutdown — отключение по перегреву) с помощью прерывания может известить ведущее устройство I²C о превышении некоторого запрограммированного температурного порога T_{OS}.

Ведущее устройство может установить как значение T_{OS} для извещения о перегреве, так и порог гистерезиса T_{HYST}, при достижении которого система температурного контроля сбрасывается в исходное состояние. В случае сброса по включению питания для LM75 устанавливаются следующие значения по умолчанию: $T_{OS} = 80°C$; $T_{HYST} = 75°C$. Ведущее устройство может в любой момент считать пороговые величины T_{OS} и T_{HYST} через интерфейс I²C точно так же как и текущее значение измеренной температуры.

Диапазон допустимых температур для LM75 составляет –25...+100°C при максимальном отклонении ±2°C или –55...+125°C при максимальном отклонении ±3°C.

Датчик LM75 выпускается в двух вариантах: LM75CIM-3 с рабочим напряжением 3,3 В и LM75CIM-5 с рабочим напряжением 5 В. Мы не будем здесь рассматривать всех технических данных, параметров и временных диаграмм микросхемы LM75, поскольку оригинальный паспорт на это устройство компании National Semiconductor находится на прилагаемом к книге компакт-диске в файле `Daten\LM75.pdf`.

Размещение выводов микросхемы LM75 показано на рис. 16.45.

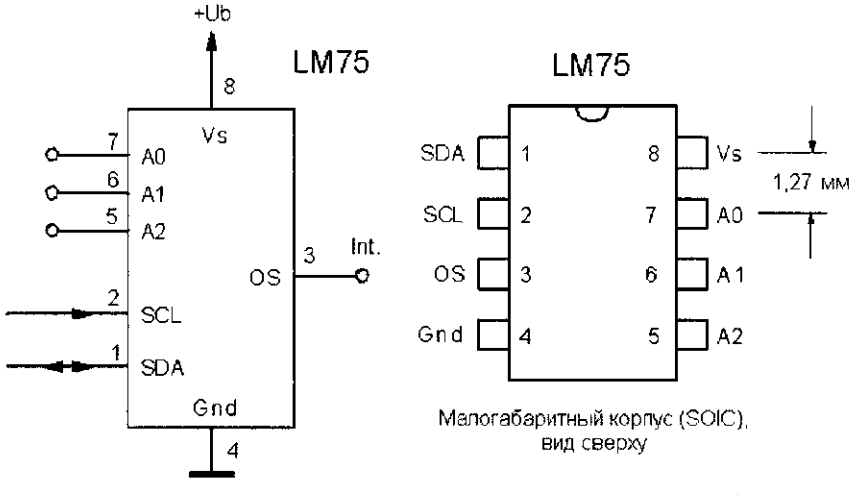


Рис. 16.45. Подключение датчика температуры LM75 к шине I²C

Выводу OS (отключение по перегреву) соответствует выход с открытым коллектором без собственного подтягивающего сопротивления, и он может быть под-

ключен к линии прерывания ведущего устройства I^2C. Внешнее подтягивающее сопротивление следует выбирать как можно большим (максимум 30 кОм), чтобы минимизировать ошибку измерений, обусловленную нагревом кристалла вследствие незначительного выходного тока.

Через выводы $A_0 \ldots A_2$ получают три младших разряда адреса I^2C микросхемы, а четыре старших разряда жестко заданы внутренне как 1 0 0 1. Полностью формат семиразрядного адреса ведомого устройства I^2C представлен на рис. 16.46.

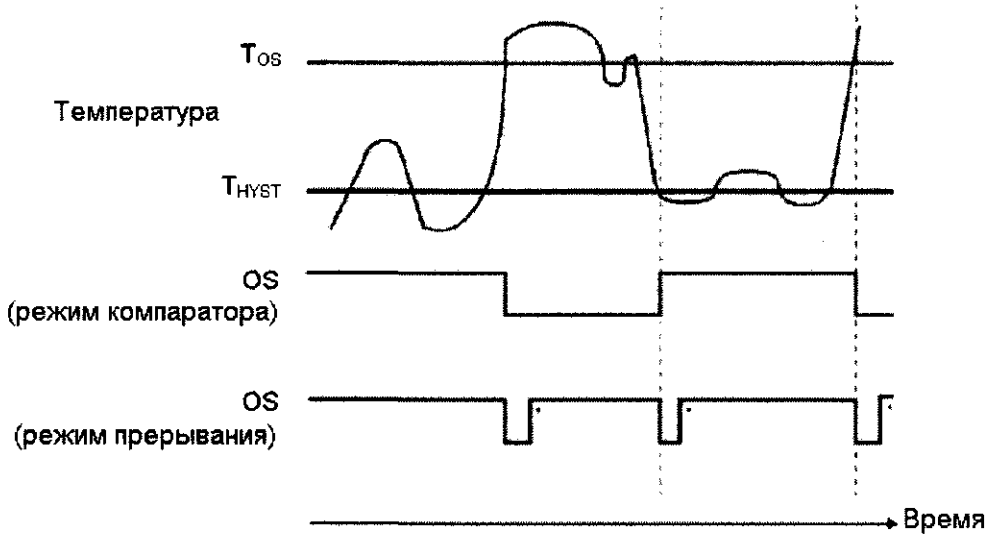
MSB						LSB
1	0	0	1	A2	A1	A0

Рис. 16.46. Адрес датчика температуры LM75

Пользователь может задать режим работы LM75 с помощью регистра конфигурации. К примеру, можно определить работу в режиме компаратора или прерывания.

В режиме компаратора микросхема LM75 выступает в качестве термостата: в случае превышения порога T_{OS} выход OS переходит в активное состояние, а при достижении порога T_{HYST} — в неактивное. Какой уровень считается активным: лог. 0 или лог. 1 — также программируется с помощью регистра конфигурации. В режиме компаратора датчик LM75 можно использовть для управления вентиляторами охлаждения или аварийными сигнализаторами.

Если в режиме прерывания будет превышен порог T_{OS}, то активность выхода OS сохраняется только до чтения регистра LM75 ведущим устройством I^2C. После подобного сброса выход OS опять может перейти в активное состояние в результате достижения нижнего порога T_{HYST} (рис. 16.47).



*) Здесь выход OS переходит в неактивное состояние только в результате чтения регистра LM75 или останова

Рис. 16.47. Поведение выхода OS в режимах компаратора и прерывания, если активному состоянию соответствует низкий уровень сигнала

И опять таки, вывод OS остается в активном состоянии до тех пор, пока не будет прочитан регистр LM75 ведущим устройством I^2C или датчик не перейдет в состояние останова.

Параметры, выбранные по умолчанию при включении питания LM75, рассчитаны на то, чтобы при подаче рабочего напряжения микросхема оказалась в определенном состоянии:

- режим компаратора;
- $T_{OS} = 80°C$, $T_{HYST} = 75°C$;
- активному состоянию выхода OS соответствует низкий уровень сигнала;
- внутренний указатель адреса установлен в $00.

При таких параметрах датчик LM75 может работать даже в автономном режиме (то есть, без какого-либо соединения по шине I^2C с ведущим устройством).

Во избежание погрешностей срабатывания, в регистре конфигурации можно запрограммировать количество последовательных замеров, при которых превышается порог T_{OS}, достаточное для перевода в активное состояние выхода OS: от 1 (значение по умолчанию) до 6.

Режим останова снижает уровень потребляемого микросхемой тока от типичного значения 250 мкА (максимум 1 мА) в активном состоянии до менее, чем 1 мкА. Для перехода в этот режим должен быть установлен в лог. 1 соответствующий разряд регистра конфигурации. При останове в режиме прерывания выход OS переходит в неактивное состояние, а в режиме компаратора его состояние не определено. На время останова интерфейс I^2C остается активным, а регистры "T_{OS}", "T_{HYST}" и конфигурации доступны для чтения и записи.

Структура регистра конфигурации LM75 показана на рис. 16.48.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	Число превышений порога		Полярн. OS	Комп./ прерыв.	Останов

Рис. 16.48. Структура регистра конфигурации LM75

При включении питания все разряды этого регистра обнуляются. Рассмотрим назначение отдельных разрядов:

- D7...D5 — зарезервированы, всегда содержат лог. 0;
- D4, D3 — количество последовательных замеров, при которых превышается порог T_{OS}, переводящее выход OS в активное состояние (табл. 16.5);
- D2 — если D2 = 0, то активному состоянию выхода OS соответствует низкий уровень сигнала, если же D2 = 1 — высокий уровень;
- D1 — если D1 = 0 — режим компаратора, если же D1 = 1 — режим прерывания;
- D0 — если D0 = 0 — нормальный режим работы. если же D0 = 1 — режим останова.

Таблица 16.5. Сочетания разрядов D4 и D3 регистра конфигурации LM75

D4	D3	Количество замеров, соответствующих превышению порога T_{OS}
0	0	1 (по умолчанию)
0	1	2
1	0	4
1	1	6

Четыре регистра датчика LM75: "T_{HYST}", "T_{OS}", "Конфигурация" и "Температура" — адресуются с помощью регистра-указателя, в который записывается соответствующий адрес (рис. 16.49).

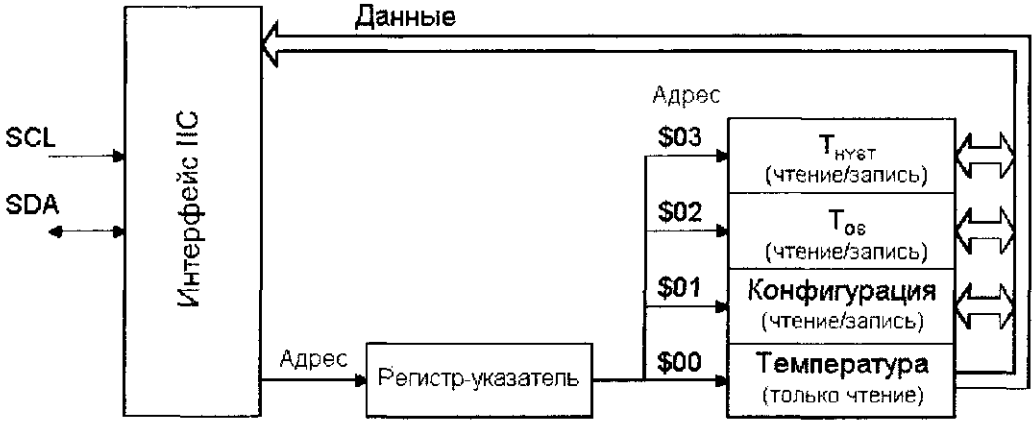


Рис. 16.49. Внутренняя регистровая структура датчика LM75

При сбросе по включению питания регистр-указатель принимает значение $00 и указывает на регистр температуры. Все регистры данных, за исключением "Температура", доступны для чтения и записи.

Регистр указатель (рис. 16.50) содержит непосредственный шестнадцатеричный адрес регистра ($00...$03), в который должна быть выполнена запись или из которого должно быть считано значение.

P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	0	Адрес регистра	

Рис. 16.50. Структура регистра-указателя

Формат остальных регистров показан на рис. 16.51.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	X	X	X	X	X	X	X

Рис. 16.51. Формат регистров "T_{HYST}", "T_{OS}" и "Температура"

Девять разрядов данных DB8...DB0 представлены в форме дополнения до двух; один младший разряд соответствует температуре 0,5°C; разряды D6...D0 младшего полубайта не определены (табл. 16.6).

Таблица 16.6. Соответствие температур и выходных значений в регистре

Температура	*Двоичное значение DB8...DB0*	*Шестнадцатеричное значение*
+ 125°C	0 1111 1010	0FA
+ 25°C	0 0011 0010	032
+ 1°C	0 0000 0010	002
+ 0,5°C	0 0000 0001	001
0°C	0 0000 0000	000
– 0,5°C	1 1111 1111	1FF
– 1°C	1 1111 1110	1FE
– 25°C	1 1100 1110	1CE
– 55°C	1 1001 0010	192

Обмен данными через шину I^2C подробно рассматривается в главе 8, и потому здесь мы затронем только свойства, имеющие отношения к датчику LM75.

Данные из LM75 могут быть считаны двумя способами. Если регистр-указатель уже содержит корректный адрес, в операции чтения участвует только байт адреса, по которому должен быть установлен в лог. 1 разряд, с последующим байтом данных для записи/чтения. Если же указатель должен быть инициализирован заранее, то последовательность состоит из байта адреса, байта указателя, условия ReSTART, еще раз байта адреса и байтов данных. Первым поступает старший байт данных (D15...D8), начиная со старшего разряда.

Считывание байта ведущим устройством I^2C подтверждается положительным (лог. 0) или отрицательным (лог. 1) квитированием. Сигнал отрицательного квитирования извещает ведущее устройство о том, что оно только что прочитало последний байт.

Если из 16-разрядного регистра будет по ошибке считано только восемь бит, а разряд D7 последующего байта содержит лог. 0, то датчик LM75 может оказаться в таком состоянии, в котором на линии SDA надолго задержится лог. 0. Это может заблокировать передачу по шине до тех пор, пока по линии SCK не будет принято по меньшей мере девять тактовых импульсов. Кроме того, ведущее устройство может вырабатывать тактовые импульсы и по каждому из них проверять линию SDA до тех пор, пока в ней опять не появится высокий уровень сигнала. Последующее условие STOP переводит датчик LM75 в исходное состояние.

Возможные варианты обмена данными с LM75 по шине I^2C показаны на рис. 16.52 – рис. 16.54 (S — условие START; A — положительное квитирование (лог. 0); /A — отрицательное квитирование (лог. 1); P — условие STOP).

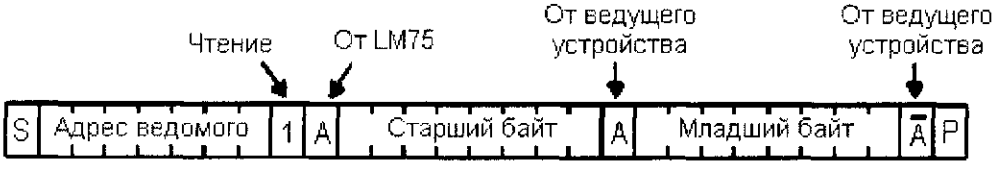


Рис. 16.52. Типичное чтение 16-разрядного регистра, когда указатель уже содержит корректный адрес

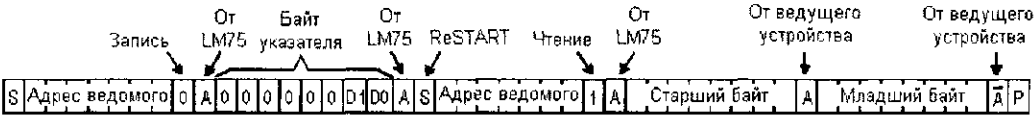


Рис. 16.53. Типичный случай установки указателя, за которой следует доступ на чтение 16-ти разрядов (например, температуры, значения T_{OS} или T_{HYST})

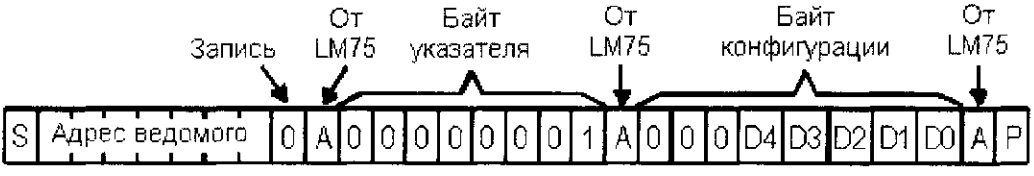


Рис. 16.54. Типичная инициализация байта конфигурации

Пример использования датчика LM75 в качестве ведомого устройства I^2C, когда в качестве ведущего устройства выступает микроконтроллер семейства AVR, рассматривается ниже в разделе "Использование микроконтроллера AVR в качестве ведущего устройства I^2C".

Подключение к шине I^2C схемы SAA1064 управления четырехразрядным светодиодным табло

SAA1064 — это микросхема, предназначенная для управления четырьмя семисегментными индикаторами с десятичной точкой и общим анодом в мультиплексном режиме или же двумя индикаторами в статическом режиме. В ее состав входит интегрированный интерфейс I^2C, позволяющий обращаться по четырем адресам ведущих устройств. Сила выходного тока для отдельных сегментов подсоединенного светодиодного индикатора достигает 21 мА и программируется с шагом 3 мА. Благодаря встроенной регулировке нагрузки по току в секции выводов, отпадает необходимость во внешнем сопротивлении для ограничения токов через сегменты.

Допустимый диапазон питающего напряжения микросхемы SAA1064 составляет +4,5...+18 В при типичном значении потребляемого тока 8,6 мА, если подключены все сегменты. Мы не будем здесь рассматривать всех технических данных, параметров и временных диаграмм микросхемы SAA1064, поскольку оригинальный паспорт на это устройство компании Philips находится на прилагаемом к книге компакт-диске в файле `Daten\SAA1064.pdf`.

Размещение выводов микросхемы SAA1064 показано на рис. 16.55.

Младшие два разряда A1 и A0 ведомого устройства I^2C задаются с помощью входа ADR (вывод 1 микросхемы SAA1064), а старшие пять жестко определены как 0 1 1 1 0. Формат семиразрядного адреса ведомого устройства SAA1064 показан на рис. 16.56.

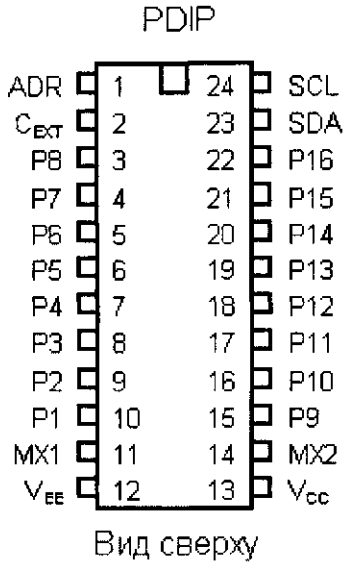


Рис. 16.55. Конструкция корпуса и размещение выводов микросхемы SAA1064

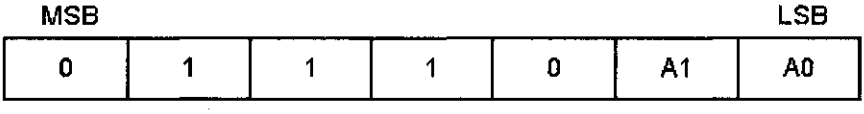


Рис. 16.56. Формат адреса, используемого микросхемой SAA1064

Четыре различных адреса определяются уровнем аналогового напряжения U_{ADR}, поданного на вывод ADR (табл. 16.7).

Таблица 16.7. Выбор одного из четырех возможных адресов SAA1064 по уровню напряжения U_{ADR} с помощью перемычек JP1...JP3 (рис. 16.58).

U_{ADR}	*A1*	*A0*	*JP3*	*JP2*	*JP1*
0 В	0	0	вставлена	извлечена	извлечена
1/3 V_{CC}	0	1	извлечена	извлечена	извлечена
2/3 V_{CC}	1	0	извлечена	извлечена	вставлена
V_{CC}	1	1	извлечена	вставлена	извлечена

Таким образом, микросхема SAA1064 предоставляет в распоряжение четыре адреса для доступа на запись (70_h, 72_h, 74_h и 76_h) и четыре адреса для доступа на чтение (71_h, 73_h, 75_h и 77_h).

Порты ввода/вывода SCL и SDA соответствуют соглашениям шины I^2C, описанным в главе 8. Между входом C_{EXT} и "землей" включен конденсатор емкостью 2,7 нФ, предназначенный для регулировки частоты мультиплексирования.

В динамическом режиме работы выходы мультиплексора MX1 и MX2 попеременно активизируются с частотой, выдаваемой системным осциллятором. В статическом режиме работы постоянно включен выход MX1. Выходной каскад состоит из эмиттерного повторителя, который может напрямую управлять общим анодом светодиодного индикатора, если полученное потребление мощности меньше предела 1000 мВт, допустимого для микросхемы в корпусе DIL. Если это-

го гарантировать нельзя, то следует применять внешние транзисторы, как это показано на рис. 16.57.

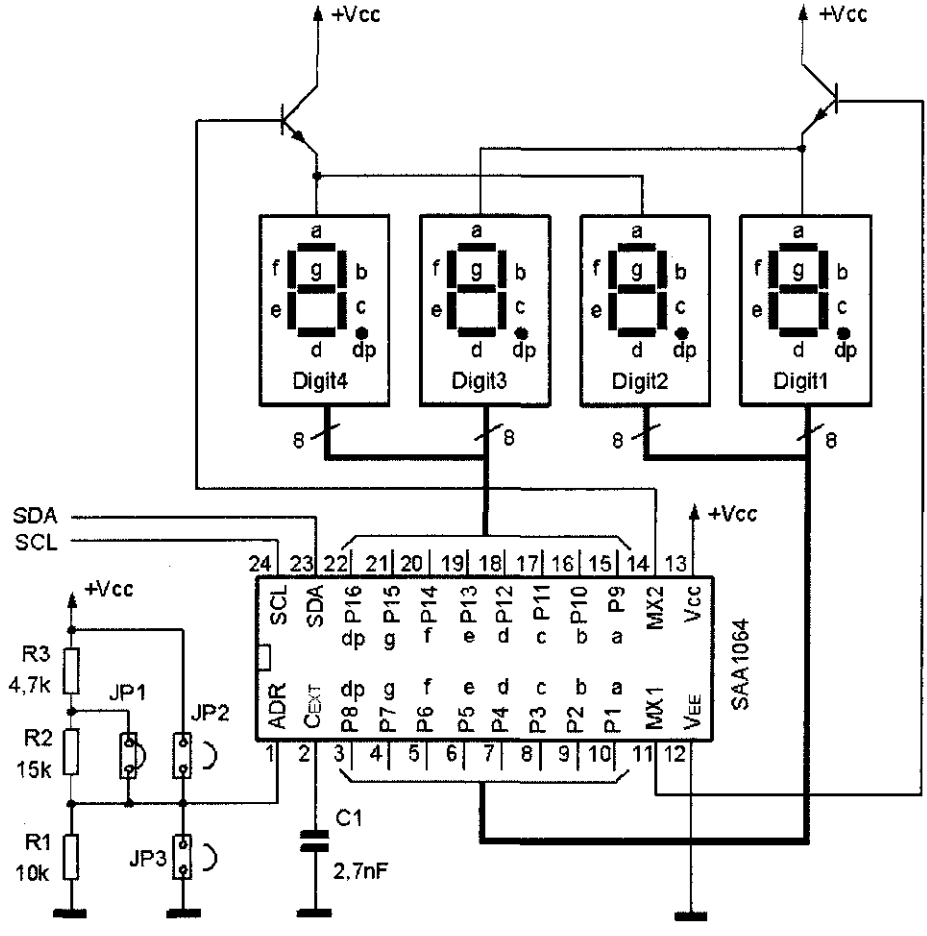


Рис. 16.57. Схема светодиодного табло, подключенного через интерфейс I2C, в динамическом режиме

Сегментные выводы P1...P16 выходят из регуляторов стабилизированного тока, для которых сила выходного тока программируется с помощью разрядов C4...C6 регистра управления микросхемы SAA1064 с шагом 3 мА и может достигать 21 мА.

Сегмент включен, если в соответствующем разряде регистра данных установлена лог. 1 (см. рис. 16.57). Благодаря тому, что все сегменты управляются независимо друг от друга, можно включать отдельные светодиоды, создавая таким образом различные знаки.

Регистры данных Digit1 и Digit2 поставлены в соответствие сегментным выходам P1...P8, а регистры Digit3 и Digit4 — выводам P9...P16. При этом старшим разрядам D17 и D27 регистров Digit1 и Digit2 соответствует вывод P8, а старшим разрядам D37 и D47 регистров Digit3 и Digit4 — вывод P16.

В статическом режиме работы сегменты двух светодиодных индикаторов подключаются непосредственно к выводам P1...P8 (Digit1) или P9...P16 (Digit2), а

на их общий анод подается напряжение V_{CC}. Выводы MX1 и MX2 остаются неподключенными, а вывод C_{EXT} соединен с "землей".

Регистр управления содержит семь разрядов управления C6...C0:

- C6 — если C6 = 1, ток через сегмент повышается на 12 мА;
- C5 — если C5 = 1, ток через сегмент повышается на 6 мА;
- C4 — если C4 = 1, ток через сегмент повышается на 3 мА;
- C3 — если C3 = 1, то все выходы с целью тестирования выдают ток, заданный с помощью разрядов C4...C6;
- C2 — если C2 = 0, то разряды табло 2 и 4 отключаются, в противном случае — включаются;
- C1 — если C1 = 0, то разряды табло 1 и 3 отключаются, в противном случае — включаются;
- C0 — если C0 = 0, то микросхема работает в статическом режиме, и постоянно включены разряды табло 1 и 2. Если C0 = 1, то микросхема работает в динамическом режиме, и мультиплексируются (то есть, попеременно включаются) разряды табло 1+3 и 2+4.

Четыре регистра данных и регистр управления SAA1064 адресуются с помощью регистра-указателя, в который требуемый адрес записывается в виде байта-интрукции (рис. 16.58).

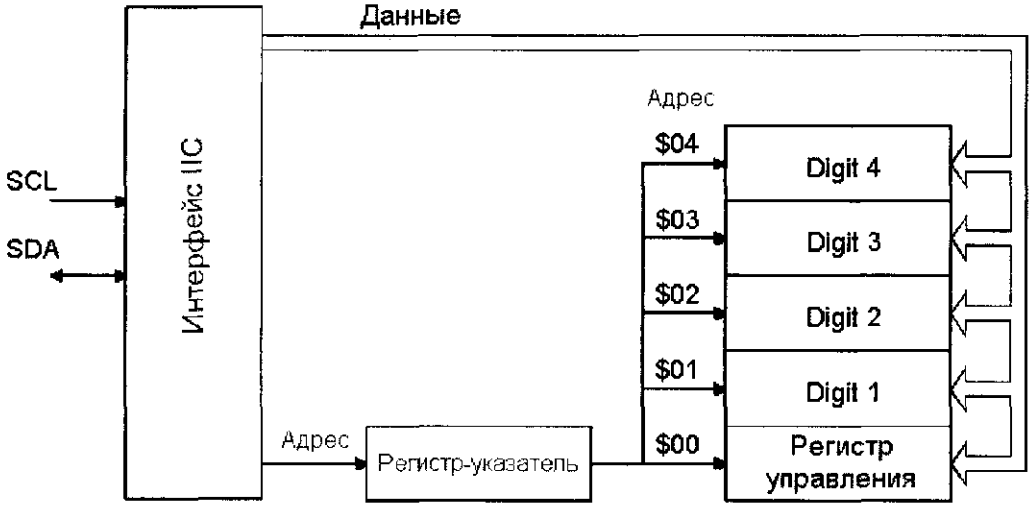


Рис. 16.58. Внутренняя регистровая структура микросхемы SAA1064

После сброса по включению питания этот регистр содержит $00 (указывает на регистр управления). После доступа на запись указатель автоматически увеличивается на 1 (автоинкремент), а вслед за состоянием счетчика 7 идет состояние 0.

Байт-инструкция содержит три адресных разряда SA...SC, формирующих адрес указателя:

0	0	0	0	0	SC	SB	SA

Из SAA1064 можно считать байт состояния, в котором информационную нагрузку несет только седьмой разряд — флаг сброса питания. Если он установлен в лог. 1, то это указывает на то, что со времени последнего чтения байта состояния было зафиксировано изменение или сбой в подаче питания. В результате считывания этот байт обнуляется.

При поступлении сигнала сброса по включению питания все регистры SAA1064 обнуляются, за исключением флага сброса питания, который устанавливается в лог. 1.

Обмен данными по шине I^2C уже был подробно рассмотрен в главе 8, и потому здесь мы затронем только свойства, характерные для микросхемы SAA1064.

Доступ к SAA1064 на чтение заключается только в передаче ведущим устройством адреса, на который ведомое устройство отвечает битом квитирования, после чего ведущее устройство считывает переданный ведомым устройством байт состояния и подтверждает его получение отрицательным квитированием (лог. 1), чтобы ведомое устройство распознало завершение операции чтения (рис. 16.59).

Рис. 16.59. Чтение ведущим устройством байта состояния

В случае доступа на запись последовательность начинается с байта адреса, после которого следует байт-инструкция для инициализации указателя. Следующий байт записывается по адресу, на который указывает указатель. После каждой записи указатель, благодаря функции автоинкремента, увеличивается на 1 и, таким образом, адресует следующую ячейку памяти. За байтом-инструкцией следует байт управления, а за ним — байты данных (рис. 16.60).

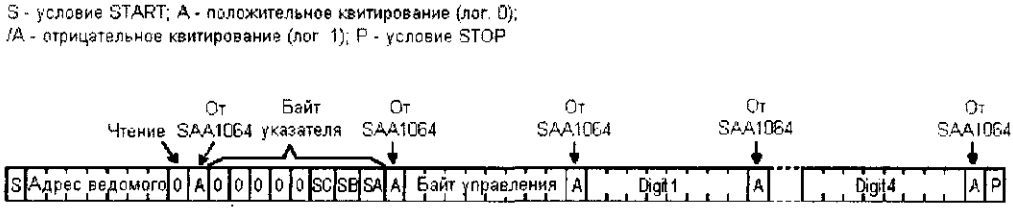


Рис. 16.60. Типичный случай доступа на запись в SAA1064

Если должны быть только выведены цифры на табло, без изменения режима работы, то байты данных могут следовать сразу же после байта-инструкции.

Пример работы микросхемы SAA1064 в качестве ведомого устройства I^2C, когда в роли ведущего выступает микроконтроллер семейства AVR, рассматривается в следующем разделе.

Использование микроконтроллера AVR в качестве ведущего устройства I^2C

В этом примере демонстрируется считывание температуры, измеренной датчиком LM75 (см. выше раздел "Подключение к шине I^2C датчика температуры LM75"), ее обработка и последующий вывод в схему управления SAA1064 (см. предыдущий раздел).

Эта программа была введена в микроконтроллер AT90S8515 и протестирована с помощью модуля STK200. Выводу PB0 соответствует линия SDA, а выводу PB1 — линия SCL, к которым для данной цели подключены подтягивающие резисторы сопротивлением 4,7 кОм (рис. 16.61).

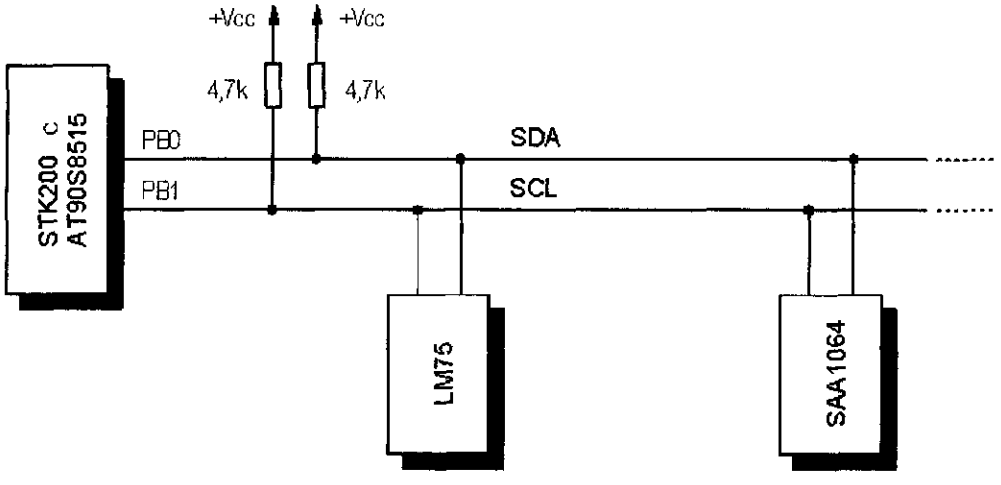


Рис. 16.61. Подключение модулей к STK200 по шине I^2C

Схема подключения модуля SAA1064 показана на рис. 16.57. Для установки адресов вставлена перемычка JP2, что соответствует адресу записи $76 и адресу чтения $77 (см. табл. 16.7). Ток, потребляемый SAA1064, не может быть обеспечен модулем STK200, поэтому используется внешний источник напряжения.

Принципиальная схема модуля LM75 показана на рис. 16.45. Для того чтобы получить адрес записи $90 и адрес чтения $91 каждый из входов выбора адреса (A0...A2) соединен с "землей". Выход OS не используется и остается не подключенным.

Структура программы соответствует рис. 16.62.

Запись через интерфейс I^2C всегда выполняется по одной и той же схеме, поэтому ее рационально представить в виде одной целой части, без разбиения на несколько подпрограмм, поскольку при каждом вызове подпрограммы тратится дополнительных семь тактовых циклов на выполнение команд `call` и `ret`. Таким образом, подпрограмма, соответствующая процессу записи, будет проходить максимально возможный путь: полную последовательность от условия ReStart через запись байта до считывания ответа ведомого устройства (квитирование).

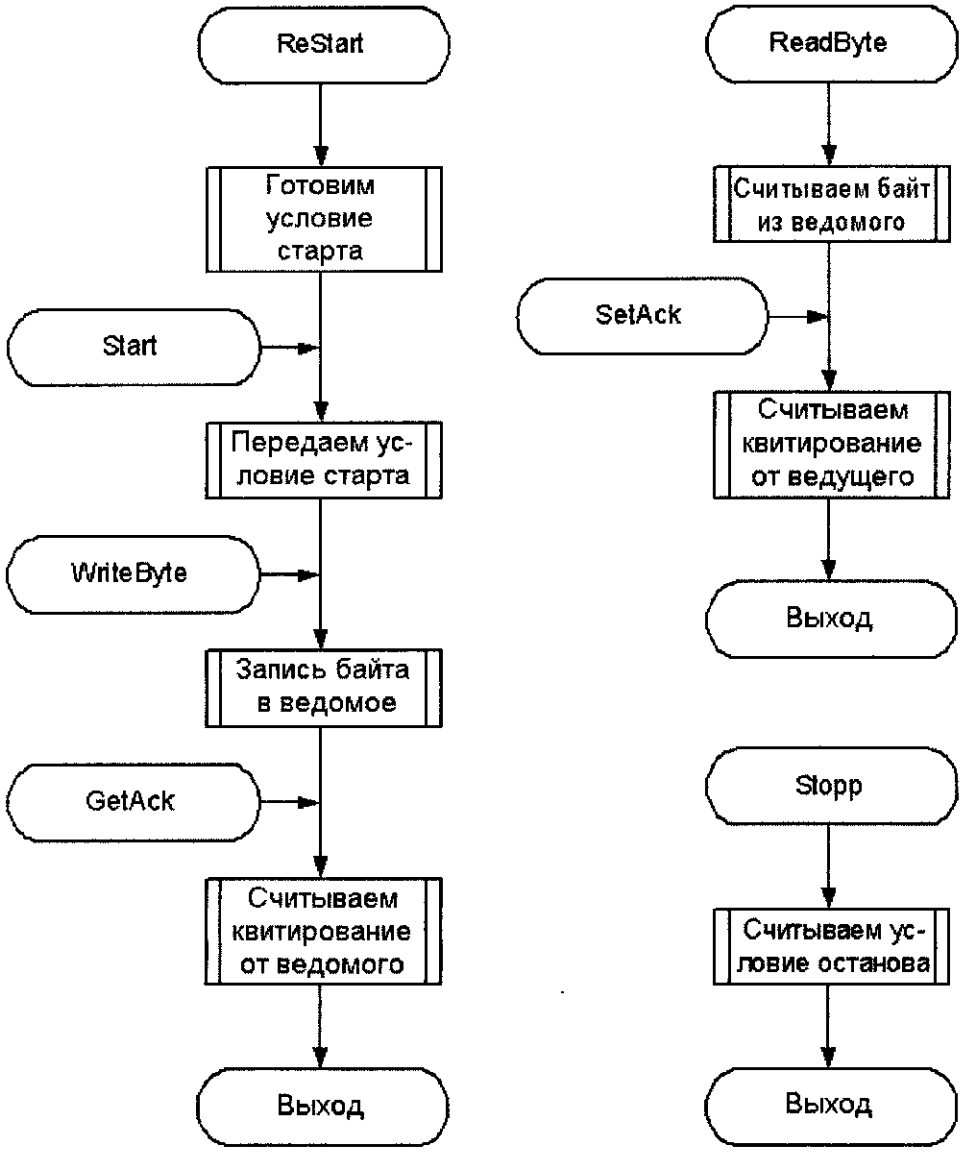


Рис. 16.62. Блок-схемы подпрограмм для обращения к интерфейсу I^2C на запись и чтение, а также для условия останова

Поскольку процессу записи не предшествует изменение направления передачи, которое потребовало бы повторения условия старта (ReStart), то процедура начинается просто переходом к метке `Start`. Аналогично, если необходимо переслать только очередной байт из последовательности нескольких байтов, то достаточно просто перейти к метке `WriteByte`.

Каждая запись завершается считыванием бита квитирования, полученного от ведомого устройства. Если этот бит содержит лог. 0, то это означает, что переданный байт был принят корректно. Если же ведущее устройство получает в ответ

лог. 1, то это означает, что соответствующее ведомое устройство отсутствует или занято, или же что произошел сбой во время передачи.

После условия Start или ReStart при передаче через интерфейс I^2C всегда следует обращение на запись к ведомому устройству, для чего отправляется адрес, и только после этого может быть осуществлен доступ на чтение первого байта. Поэтому подпрограмма, которая обращается к ведомому устройству для чтения, начинается непосредственно с чтения байта, после чего следует передача бита квитирования ведущим устройством. Если после обращения на чтение ведущее устройство захочет прочитать еще один байт, оно, как правило, передает бит положительного квитирования (лог. 0), если же оно хочет завершить доступ на чтение, то передает бит отрицательного квитирования (лог. 1).

Передача завершается условием останова, переводящим шину в ждущий режим. Полный исходный текст программы передачи данных через интерфейс I^2C представлен в следующем листинге.

Имя файла на прилагаемом к книге компакт-диске: `\Programm\1615MSTR.asm`

```
;**** Использование AT90S8515 в качестве ведущего устройства I2C ********
;*
;* Чистая программная реализация ведущего устройство I2C с помощью AT90S8515.
;* Этот Master через равномерные промежутки времени опрашивает температуру с
;* подключенного к шине I2C ведомого датчика LM75 и выводит ее по той же шине
;* на ведомую микросхему SAA1064. Для наглядности используем систему с одним
;* ведущим устройством.
;* Функцию реализуем без использования прерываний AVR.
;* Таким образом, все прерывания можно использовать для других целей.
;* Пример написан для микроконтроллера AT90S8515, однако, поскольку, из всей
;* аппаратной периферии используется только T/C0, он применим для всех
;* микроконтроллеров семейства AVR.
;* Тактовая частота микроконтроллеров AVR: 4 МГц (Стандарт STK200).
;*
;* Подпрограммы, которые могут быть использованы в других случаях применения:
;*
;* > ReStart:  Создает новое условие старта; должна следовать непосредственно
;*             перед подпрограммой "Start".
;* > Start:    Передает условие старта и адрес ведомого устройства, хранимый
;*             в "dbyt"; должна следовать непосредственно
;*             перед подпрограммой "WriteByte".
;* > WriteByte: Передает ведомому устройству байт, хранимый в "dbyt";
;*             при считывании значения с ведомого устройства должна следовать
;*             непосредственно перед подпрограммой "GetAck"
;* > GetAck:   Считывает бит квитирования.
;* > ReadByte: Считывает байт с ведомого устройства, записываемый в "dbyt";
;*             при выдаче бита квитирования ведущим устройством должна
;*             следовать непосредственно перед подпрограммой "SetAck"
;* > SetAck:   Выводит бит квитирования от ведущего устройства.
;* > Stopp:    Завершает передачу выдачей условия завершения.
;*
;*************************************************************************
.nolist
.include "8515def.inc"
.list

.def    tmp1 = r16      ; Рабочий регистр
```

```
.def    time = r17      ; Счетчик цикла задержки
.def    dbyt = r18      ; Байт данных для передачи по шине I2C
.def    Flg = r19       ; Флаги программы
.def    Dig43 = r20     ; Цифры 4 и 3 для вывода
.def    Dig21 = r21     ; Цифры 2 и 1 для вывода

; Флаги в регистре Flg:
.equ    Ack = 0         ; Флаг квитирования = разряд 0
.equ    DP1 = 4         ; Десятичная точка цифры 1
.equ    DP2 = 5         ; Десятичная точка цифры 2
.equ    DP3 = 6         ; Десятичная точка цифры 3
.equ    DP4 = 7         ; Десятичная точка цифры 4

; Входы/выходы I2C:
.equ    SDA = PB0       ; SDA = порт B / разряд 0
.equ    SCL = PB1       ; SCL = порт B / разряд 1

; Адреса ведомых устройств I2C:
.equ    DispAdr = $76 ; Адрес SAA1064 (Вывод ADR (1) - Vcc)
.equ    TempAdr = $90 ; Адрес LM75

; Макроопределения:
.macro SCL_Up           ; Нарастающий фронт SCL
cbi DDRB,SCL            ; SCL - высокоомный
.endmacro
.macro SCL_Dwn          ; Ниспадающий фронт SCL
sbi DDRB,SCL            ; SCL = лог. 0
.endmacro
.macro SDA_Hi           ; Установка SDA в 1
cbi DDRB,SDA            ; SDA - высокоомный
.endmacro
.macro SDA_Lo           ; Установка SDA в 0
sbi DDRB,SDA            ; SDA = лог. 0
.endmacro

.cseg

000000 c0cb    rjmp Initial          ; После сброса - к главной программе

Wait250:                             ; Задержка на 250 мс
000001 e000    ldi tmp1,0            ; Загружаем интервал времени
000002 bf02    out TCNT0,tmp1        ; Инициализируем T/C0
000003 e002    ldi tmp1,1<<TOV0      ; Загружаем битовую позицию флага TOV0
000004 bf08    out TIFR,tmp1         ; Сбрасываем флаг TOV0
000005 e014    ldi time,4            ; 4 обращения к таймеру
Wait:
000006 b708    in tmp1,TIFR          ; Извлекаем содержимое регистра TIFR
000007 ff01    sbrs tmp1,TOV0        ; Пропускаем следующую команду, если
                                     ; произошло переполнение таймера
000008 cffd    rjmp Wait             ; Дальше ожидаем переполнения
000009 e002    ldi tmp1,1<<TOV0      ; Загружаем битовую позицию флага TOV0
00000a bf08    out TIFR,tmp1         ; Сбрасываем флаг TOV0
00000b 951a    dec time
00000c f7c9    brne Wait             ; 4 прохода = 250 мс
00000d 9508    ret
Wait5us:                             ; Цикл задержки 5 мкс при f = 4 МГц
```

```
00000e e014    ldi time,4
W5:
00000f 951a    dec time              ; 4 x 3 такта + nop + rcall + ret
000010 f7f1    brne W5               ; = 20 тактов
000011 0000    nop
000012 9508    ret

Wait4us:                             ; Цикл задержки 4 мкс при f = 4 МГц
000013 e013    ldi time,3
W4:
000014 951a    dec time              ; 3 x 3 такта + rcall + ret
000015 f7f1    brne W4               ; = 16 тактов
000016 9508    ret

ReStart:                             ; Повторяем условие старта
000017 + SCL_Dwn                     ; Ниспадающий фронт SCL
000018 0000    nop
000019 + SDA_Hi                      ; Установка SDA в 1
00001a 0000    nop
00001b dff2    rcall wait5us         ; Цикл задержки
00001c 0000    nop
00001d + SCL_Up                      ; Нарасающий фронт SCL
RS1:
00001e 9bb1    sbis PinB,SCL         ; Состояние ожидания ведомого устройства ?
00001f cffe    rjmp RS1
000020 dfed    rcall wait5us         ; Цикл задержки

Start:
000021 + SDA_Lo                      ; SDA = 0: условие старта
000022 dff0    rcall wait4us         ; Цикл задержки

WriteByte:                           ; Запись байта в ведомое устройство
000023 9408    sec                   ; Флаг переноса C = 1
000024 1f22    rol dbyt              ; Сдвигаем влево, C->LSB, MSB->C
000025 c001    rjmp bit1             ; Особая обработка разряда 1
WriteBit:
000026 0f22    lsl dbyt              ; Если dbyt пустой, ...
bit1:
000027 f069    breq GetAck           ; ... то передача завершена
000028 + SCL_Dwn                     ; Ниспадающий фронт SCL
000029 f418    brcc WriteLow         ; Переход, если флаг C = 0
00002a 0000    nop                   ; Чтобы уравновесить время выполнения
00002b + SDA_Hi                      ; Установка SDA в 1
00002c c002    rjmp SCL_High

WriteLow:
00002d + SDA_Lo                      ; Установка SDA в 0
00002e c000    rjmp SCL_High         ; Чтобы уравновесить длительность такта
SCL_High:
00002f dfe3    rcall wait4us         ; Цикл задержки
000030 + SCL_Up                      ; Ниспадающий фронт SCL
WB1:
000031 9bb1    sbis PinB,SCL         ; Состояние ожидания ведомого устройства ?
000032 cffe    rjmp WB1
000033 dfda    rcall wait5us         ; Цикл задержки
000034 cff1    rjmp WriteBit
```

```
GetAck:                                 ; Считывание бита квитирования от ведомого
000035 + SCL_Dwn                        ; Ниспадающий фронт SCL
000036 + SDA_Hi                         ; SDA - высокоомное состояние
000037 dfd6    rcall wait5us            ; Цикл задержки
000038 + SCL_Up                         ; Нарастающий фронт SCL
GA1:
000039 9bb1    sbis PinB,SCL            ; Состояние ожидания ведомого устройства ?
00003a cffe    rjmp GA1
00003b 7f3e    cbr Flg,1<<Ack           ; Флаг Ack = 0
00003c 99b0    sbic PinB,SDA            ; Если SDA = 1,
00003d 6031    sbr Flg,1<<Ack           ;   флаг Ack = 1
00003e dfd4    rcall wait4us            ; Цикл задержки
00003f 9508    ret

ReadByte:
000040 e021    ldi dbyt,1               ; Назначаем в качестве счетчика битов
RB1:
000041 + SCL_Dwn                        ; Ниспадающий фронт SCL
000042 dfca    rcall wait5us            ; Цикл задержки
000043 + SCL_Up                         ; Нарастающий фронт SCL
RB2:
000044 9bb1    sbis PinB,SCL            ; Состояние ожидания ведомого устройства ?
000045 cffe    rjmp RB2
000046 dfc6    rcall wait5us            ; Цикл задержки
000047 9488    clc                      ; Флаг C = 0
000048 99b0    sbic PinB,SDA            ; Если SDA = 1
000049 9408    sec                      ; Флаг C = 1
00004a 1f22    rol dbyt                 ; Сдвигаем прочитанный бит в байте
00004b f7a0    brcc RB1                 ; Переход, если чтение не завершено

SetAck:                                 ; Вывод бита квитирования
00004c + SCL_Dwn                        ; Ниспадающий фронт SCL
00004d ff30    sbrs Flg,Ack             ; Пропускаем следующую команду, если Ack=1
00004e c002    rjmp SA1                 ; Переход, если Ack = 0
00004f + SDA_Hi                         ; Установка SDA в 1
000050 c001    rjmp SA2
SA1:
000051 + SDA_Lo                         ; Установка SDA в 0
SA2:
000052 dfba    rcall wait5us            ; Цикл задержки
000053 + SCL_Up                         ; Нарастающий фронт SCL
SA3:
000054 9bb1    sbis PinB,SCL            ; Состояние ожидания ведомого устройства ?
000055 cffe    rjmp SA3
000056 dfb6    rcall wait5us            ; Цикл задержки
000057 9508    ret

Stopp:                                  ; Вывод условия завершения
000058 + SCL_Dwn                        ; Ниспадающий фронт SCL
000059 + SDA_Lo                         ; Установка SDA в 0
00005a dfb2    rcall wait5us            ; Цикл задержки
00005b + SCL_Up                         ; Установка SCL в 1
00005c dfb0    rcall wait5us            ; Цикл задержки
00005d + SDA_Hi                         ; Нарастающий фронт SCL
00005e dfae    rcall wait5us            ; Цикл задержки
```

```
00005f 9508    ret

InitDisp:                          ; Передача байта управления в SAA1064
000060 e726    ldi dbyt,DispAdr    ; Адрес SAA1064, направление - запись
000061 dfbe    rcall Start         ; Передаем условие старта + адрес ведомого
000062 fd30    sbrc Flg,Ack        ; Пропускаем следующую команду, если квит.
000063 9508    ret                 ; Завершаем, если отрицательное квитиров.
000064 e020    ldi dbyt,$00        ; Подадрес - регистр управления
000065 dfbc    rcall WriteByte     ; Передаем
000066 fd30    sbrc Flg,Ack        ; Пропускаем следующую команду, если квит.
000067 9508    ret                 ; Завершаем, если отрицательное квитиров.
000068 e727    ldi dbyt,$77        ; Максим. ток, вывод цифр, динамич. режим
000069 dfb8    rcall WriteByte     ; Передаем
00006a 9508    ret

OutDisp:                           ; Вывод 4-х цифр в SAA1064
00006b e726    ldi dbyt,DispAdr    ; Адрес SAA1064, направление - запись
00006c dfb3    rcall Start         ; Передаем условие старта + адрес ведомого
00006d fd30    sbrc Flg,Ack        ; Пропускаем следующую команду, если квит.
00006e 9508    ret                 ; Завершаем, если отрицательное квитиров.
00006f e021    ldi dbyt,$01        ; Подадрес = Digit 1
000070 dfb1    rcall WriteByte     ; Передаем
000071 fd30    sbrc Flg,Ack        ; Пропускаем следующую команду, если квит.
000072 9508    ret                 ; Завершаем, если отрицательное квитиров.
000073 2f05    mov tmp1,Dig21      ; Цифры 1 (младшая) и 2 (старшая)
000074 700f    andi tmp1,$0F       ; Очищаем старший полубайт (цифра 2)
000075 d01e    rcall Get7Segm      ; Загружаем 7-сегментный код младшего
                                   ; полубайта
000076 fd34    sbrc Flg,DP1        ; Пропускаем следующую команду, если
                                   ; в цифре 1 нет десятичной точки
000077 6820    sbr dbyt,1<<7       ; Устанавливаем десятичную точку
000078 dfa9    rcall WriteByte     ; Передаем цифру 1
000079 fd30    sbrc Flg,Ack        ; Пропускаем следующую команду, если квит.
00007a 9508    ret                 ; Завершаем, если отрицательное квитиров.
00007b 2f05    mov tmp1,Dig21      ; Цифры 1 (младшая) и 2 (старшая)
00007c 9502    swap tmp1           ; Меняем местами младшую и старшую цифры
00007d 700f    andi tmp1,$0F       ; Очищаем старший полубайт (цифра 1)
00007e d015    rcall Get7Segm      ; Загружаем 7-сегментный код младшего
                                   ; полубайта
00007f fd35    sbrc Flg,DP2        ; Пропускаем следующую команду, если
                                   ; в цифре 2 нет десятичной точки
000080 6820    sbr dbyt,1<<7       ; Устанавливаем десятичную точку
000081 dfa0    rcall WriteByte     ; Передаем цифру 2
000082 fd30    sbrc Flg,Ack        ; Пропускаем следующую команду, если квит.
000083 9508    ret                 ; Завершаем, если отрицательное квитиров.
000084 2f04    mov tmp1,Dig43      ; Цифры 3 (младшая) и 4 (старшая)
000085 700f    andi tmp1,$0F       ; Очищаем старший полубайт (цифра 4)
000086 d00d    rcall Get7Segm      ; Загружаем 7-сегментный код младшего
                                   ; полубайта
000087 fd36    sbrc Flg,DP3        ; Пропускаем следующую команду, если
                                   ; в цифре 3 нет десятичной точки
000088 6820    sbr dbyt,1<<7       ; Устанавливаем десятичную точку
000089 df98    rcall WriteByte     ; Передаем цифру 3
00008a fd30    sbrc Flg,Ack        ; Пропускаем следующую команду, если квит.
00008b 9508    ret                 ; Завершаем, если отрицательное квитиров.
00008c 2f04    mov tmp1,Dig43      ; Цифры 3 (младшая) и 4 (старшая)
```

```
00008d 9502    swap tmp1              ; Меняем местами младшую и старшую цифру
00008e 700f    andi tmp1,$0F          ; Очищаем старший полубайт (цифра 3)
00008f d004    rcall Get7Segm         ; Загружаем 7-сегментный код младшего
                                      ; полубайта
000090 fd37    sbrc Flg,DP4           ; Пропускаем следующую команду, если
                                      ; в цифре 4 нет десятичной точки
000091 6820    sbr dbyt,1<<7          ; Устанавливаем десятичную точку
000092 df8f    rcall WriteByte        ; Передаем цифру 4
000093 9508    ret

Get7Segm:
000094 e020    ldi dbyt,0
000095 bb2f    out EEARH,dbyt         ; Старший байт адреса EEPROM = 0
000096 bb0e    out EEARL,tmp1         ; Младший байт адреса EEPROM = tmp1
000097 e001    ldi tmp1,$01           ; Устанавливаем разрешение чтения
000098 bb0c    out EECR,tmp1          ; Начинаем доступ к EEPROM на чтение
000099 b32d    in dbyt,EEDR           ; Считанный байт - в tmp1
00009a 9508    ret

GetTemp:                              ; Считываем температуру из LM75
00009b e920    ldi dbyt,TempAdr       ; Адрес LM75, направление - запись
00009c df83    rcall Start            ; Передаем условие старта + адрес ведомого
00009d fd30    sbrc Flg,Ack           ; Пропускаем следующую команду, если квит.
00009e 9508    ret                    ; Завершаем, если отрицательное квитиров.
00009f e020    ldi dbyt,0             ; Указатель LM75 (указывает на темп.)
0000a0 df81    rcall WriteByte        ; Передаем
0000a1 fd30    sbrc Flg,Ack           ; Пропускаем следующую команду, если квит.
0000a2 9508    ret                    ; Завершаем, если отрицательное квитиров.
0000a3 e921    ldi dbyt,TempAdr+1     ; Адрес LM75, направление - чтение
0000a4 df71    rcall ReStart          ; Передаем новое условие старта + адрес
0000a5 fd30    sbrc Flg,Ack           ; Пропускаем следующую команду, если
                                      ; бит квитирования = 0
0000a6 9508    ret                    ; Завершаем, если отрицательное квитиров.
0000a7 df97    rcall ReadByte         ; Считываем байт, бит квитирования = 0
0000a8 2f42    mov Dig43,dbyt         ; Сохраняем старший байт в буфер
0000a9 6031    sbr Flg,1<<Ack         ; Нет квитирования
0000aa df94    rcall ReadByte         ; Считываем байт, бит квитирования = 1
0000ab 2f52    mov Dig21,dbyt         ; Сохраняем младший байт в буфер
0000ac dfab    rcall stopp            ; Передаем условие завершения
Calc:
0000ad fb47    bst Dig43,7            ; Копируем знак в флаг T
0000ae f446    brtc Positiv           ; Переход, если знак положительный
0000af ff57    sbrs Dig21,7           ; Пропускаем следующую команду, если
                                      ; старший разряд в позиции после точки = 1
0000b0 c003    rjmp GT1               ; Переход, если старший разряд в позиции
                                      ; после точки = 0
0000b1 9540    com Dig43              ; Дополнение до единицы
0000b2 e055    ldi Dig21,$05          ; Позиция после точки -xx,5 °C (цифра 1)
0000b3 c007    rjmp HexDez
GT1:
0000b4 9541    neg Dig43              ; Дополнение до двух
0000b5 e050    ldi Dig21,$00          ; Позиция после точки -xx,0 °C (цифра 1)
0000b6 c004    rjmp HexDez
Positiv:
0000b7 e005    ldi tmp1,$05           ; Позиция после точки xx,5 °C (цифра 1)
```

```
0000b8 ff57    sbrs Dig21,7          ; Пропускаем следующую команду, если
                                     ; старший разряд в позиции после точки = 1
0000b9 e000    ldi tmp1,$00          ; Позиция после точки -xx,0 °C (цифра 1)
0000ba 2f50    mov Dig21,tmp1        ; Принимаем позицию после точки
HexDez:                              ; Шестнадцатеричное число в Dig43<$64(100)
0000bb e000    ldi tmp1,0            ; Счетчик для BCD-преобразования
HD1:
0000bc 504a    subi Dig43,10         ; Вычитаем 10
0000bd f010    brlo HD2              ; Переход, если результат отрицательный
0000be 9503    inc tmp1              ; Старший полубайт BCD-числа
0000bf cffc    rjmp HD1              ; Деление еще не завершено
HD2:
0000c0 5f46    subi Dig43,-10        ; Прибавляем 10 = остаток от деления
0000c1 3000    cpi tmp1,0            ; Ведущий нуль в старшем полубайте ?
0000c2 f409    brne GT2              ; Переход, если нет
0000c3 e00e    ldi tmp1,$0E          ; Пробел вместо ведущего нуля
GT2:
0000c4 9542    swap Dig43            ; Меняем местами старш. и младш. полубайты
0000c5 2b54    or Dig21,Dig43        ; Старший полубайт  - в Dig21
0000c6 ef40    ldi Dig43,$F0         ; Установка старшего полубайта (знак "-")
0000c7 2b40    or Dig43,tmp1         ; Устанавливаем старший BCD-полубайт
0000c8 f40e    brtc HD3              ; Переход к позиции знака
0000c9 9508    ret
HD3:
0000ca 7e4f    cbr Dig43,1<<4        ; $E= пробел, $F= минус
0000cb 9508    ret

Initial:
0000cc e52f    ldi dbyt,Low(RAMEND)  ; Функция вспомогательного регистра
0000cd bf2d    out SPL,dbyt
0000ce e022    ldi dbyt,High(RAMEND)
0000cf bf2e    out SPH,dbyt          ; Инициализируем указатель стека
0000d0 2722    clr dbyt              ; Функция вспомогательного регистра
0000d1 bb28    out PortB,dbyt        ; Обнуляем все разряды
0000d2 bb22    out PortD,dbyt        ; Обнуляем все разряды
0000d3 bb27    out DDRB,dbyt         ; Переключаем выходной драйвер в
                                     ; высокоомное состояние
0000d4 bb21    out DDRD,dbyt         ; Переключаем выходной драйвер в
                                     ; высокоомное состояние
0000d5 e005    ldi tmp1,5            ; Загружаем делитель
0000d6 bf03    out TCCR0,tmp1        ; Запускаем T/C0, коэффициент дел. = 1024
0000d7 e031    ldi Flg,$01           ; Сбрасываем в 0 все флаги, кроме Ack
0000d8 6230    sbr Flg,1<<DP2        ; Десятичная точка возле второй цифры
Haupt:
0000d9 df86    rcall InitDisp        ; Инициализируем SAA1064
0000da df7d    rcall stopp           ; Выводим условие завершения
0000db fd30    sbrc Flg,Ack          ; Пропускаем следующую команду, если
                                     ; получено квитирование
0000dc cffc    rjmp Haupt            ; Новая попытка, если нет квитирования
Next:
0000dd dfbd    rcall GetTemp         ; Считываем температуру из LM75
0000de df8c    rcall OutDisp         ; Показываем температуру
0000df df78    rcall stopp           ; Завершаем вывод
0000e0 fd30    sbrc Flg,Ack          ; Пропускаем следующую команду, если
                                     ; получено квитирование
0000e1 cff7    rjmp Haupt            ; Новая попытка, если нет квитирования
```

```
0000e2 df1d    rcall Wait250          ; Задержка 250 мс
0000e3 cff9    rjmp Next              ; Следующее значение

.eseg
HexTo7Segm:                           ; Таблица 7-сегментных кодов
.db $3F,$06,$5B,$4F,$66,$6D,$7D,$07 ; Шестнадцатеричные 0..7
000000 3f
000001 06
000002 5b
000003 4f
000004 66
000005 6d
000006 7d
000007 07
.db $7F,$6F,$77,$7C,$39,$5E,$00,$40 ; Шестнадцатеричные 8..D,E=пробел,F=минус
000008 7f
000009 6f
00000a 77
00000b 7c
00000c 39
00000d 5e
00000e 00
00000f 40
;.db $7F,$6F,$77,$7C,$39,$5E,$79,$71 ; Альтернатива: шестнадцатеричные 8..F
```

В этой программе после обычного определения регистров-переменных и констант следует описание макросов. Поскольку все четыре макроса состоят только из одной команды, их, в принципе, можно было бы и не объявлять как макросы, однако использование макросов делает программу более наглядной. Это объясняется тем, что сигналы I^2C выдаются по технологии открытого коллектора, и потому в случае реализации для микроконтроллеров AVR порты соответствующих выходов постоянно находятся в состоянии лог. 0, а уровень переключается с помощью регистра направления передачи данных (см. главу 10). Так, на шину в результате установки направления записи как "Выход" (посредством 1) выдается лог. 0. Аналогичным образом, переключение в высокоомное состояние (лог. 1, получаемая в шине с помощью подтягивающего сопротивления) осуществляется посредством установки 0 в соответствующем регистре направления передачи данных.

Описание подпрограмм

Каждая из описанных ниже подпрограмм завершается в момент окончания фазы высокого уровня тактового сигнала. Это гарантиурет определенную точку входа для следующей части программы.

Подпрограмма `Wait250` вызывает задержку приблизительно на 250 мс. Для этого T/C0 инициализируется значением $00, а его флаг переполнения сбрасывается. Затем отсчитываются 256 импульсов на тактовом входе до тех пор, пока не возникнет переполнение. Частота входного такта получается делением частоты системной синхронизации 4 МГц на 1024, и потому задержка составляет 256 x 1024 / 4 МГц = 65,536 мс. Как только опять устанавливается флаг переполнения,

цикл задержки прерывается, а затем повторяется еще трижды, чтобы получить требуемую задержку 262 мс.

В подпрограммах `Wait5us` и `Wait4us` реализованы задержки на 5 мкс и на 4 мкс соответственно. Эти задержки необходимы для тактирования шины I^2C.

Подпрограмма `ReStart` готовит повторение условия старта для I^2C. Для этого высокий уровень такта SCL, оставшийся от предыдущей операции квитирования, снимается, и в линию SDA выдается сигнал высокого уровня. После задержки на 5 мкс следует нарастающий фронт сигнала SCL. Далее ожидаем, пока ведомое устройство на перейдет в состояние ожидания (лог. 0 в линии SCL), и после еще одной задержки в 5 мкс переходим к следующей части программы.

Подпрограмма `Start` — переход в режим выдачи условия начала передачи, которое представляет собой установку низкого уровня сигнала SDA при высоком уровне сигнала SCL. После описанной выше задержки можно выполнить передачу адреса ведомого устройство с помощью подпрограммы `WriteByte`.

Подпрограмма `WriteByte` передает один байт ведомому устройству. Сразу же после условия старта должен следовать адрес ведомого устройства. Тем не менее, во время передачи ведомому устройству может пересылаться последовательность из нескольких байтов, поэтому подпрограмма `WriteByte` может также перейти непосредственно к выдаче байта (как в подпрограмме `OutDisp`). Все восемь разрядов байта, представленного в виде переменной `dbyt`, последовательно сдвигаются влево и передаются по линии SDA (начиная со старшего разряда).

Подпрограмма `GetAck` считывает реакцию ведомого устройства на только что переданный байт. Лог. 0 указывает на корректный прием (положительное квитирование), а лог. 1 — на сбой (отрицательное квитирование). Здесь также после установки сигнала высокого уровня в линии SCL реализована задержка до перехода ведомого устройства в состояние ожидания.

Подпрограмма `ReadByte` считывает байт из адресуемого ведомого устройства. По аналогии с подпрограммой `WriteByte`, она последовательно сдвигает влево восемь принятых битов (начиная со старшего), сохраняя их в переменной `dbyt`.

В подпрограмме `SetAck` реализована выдача ведущим устройством бита квитирования для прочитанного байта. Посредством лог. 0 Master извещает о том, что может быть считан еще один байт, а посредством лог. 1 — о завершении операции чтения.

Наконец, подпрограмма `Stopp` завершает передачу данных по шине I^2C. Для этого в линию SDA выдается лог. 1, в то время как в линии SCL также установлен уровень лог. 1.

Подпрограмма `InitDisp` с помощью микросхемы SAA1064 инициализирует модуль вывода. После передачи заданного адреса ведомого устройства $76, определяющего направление записи, для программирования SAA1064 последовательно записывается байт-инструкция $00 для установки адреса указателя и байт управления $77 (C6...C4 = лог. 1 → максимальный ток через сегмент, C3 = лог. 0 → тест сегмента отключен, C2...C1 = лог. 1 → все цифры активны, C0 = лог. 1 → мультиплексный режим). После каждой записи оценивается значение принятого бита квитирования. Если он равен лог. 1 (отрицательное квитирование) выполнение подпрограммы досрочно прерывается.

Подпрограмма OutDisp выводит в модуль SAA1064 четыре цифры значения температуры. После передачи заданного адреса ведомого устройства $76, определяющего направление записи, записывается байт-инструкция $01 для установки адреса указателя. Четыре цифры передаются в подпрограмму OutDisp в виде байтов Dig21 и Dig43. Шестнадцатеричное значение первой отображаемой цифры содержится в младшем полубайте регистра Dig21. С помощью подпрограммы Get7Segm это значение преобразовывается в семисегментный код и сохраняется в переменной dbyt. Десятичная точка, соответствующая первой цифре, должна быть активна, поэтому в регистре Flg устанавливается флаг DP1. В данном случае устанавливается старший разряд DP1. Затем значение dbyt может быть передано в SAA1064 для отображения первой цифры. Подобным же образом обрабатываются и передаются цифры 2...4. При получении отрицательного квитирования выполнение подпрограммы досрочно прерывается.

Подпрограмма Get7Segm с помощью таблицы HexTo7Segm, расположенной в памяти EEPROM, определяет по шестнадцатеричному числу, хранимому в tmp1, семисегментный код и записывает его в регистр dbyt. Коды в таблице HexTo7Segm начинаются по адресу $000, поэтому шестнадцатеричное значение используется в качестве обычного смещения адреса. Поскольку отображение реализуется в десятичной форме, коды A_h...F_h используются на усмотрение программиста. Так как требуется отображать знак минуса и пробел, им можно назначить шестнадцатеричные коды E_h и F_h.

Подпрограмма GetTemp считывает текущую температуру, измеренную модулем LM75. После передачи заданного адреса ведомого устройства $90, определяющего направление записи, записывается байт-инструкция $00 для установки адреса указателя. В результате указатель ссылается на регистр температуры. Поскольку такая установка выбирается по умолчанию при включении питания, явная инициализация указателя показана здесь только для полноты изложения. После каждой записи оценивается полученный бит квитирования. Если он содержит лог. 1 (отрицательное квитирование), то выполнение подпрограммы досрочно прерывается.

Поскольку теперь направление передачи должно измениться на "Чтение", передается условие рестарта и адрес ведомого устройства $91. После этого старший байт текущей температуры может быть считан в регистр Dig43 и квитирован посредством лог. 0 (извещение о чтении второго байта из ведомого устройства). Младший байт считывается в регистр Dig21 и квитируется посредством лог. 1 (завершение операции чтения). Передача завершается выдачей условия останова.

В части Calc этой подпрограммы считанное шестнадцатеричное число преобразовывается в десятичное. Старший разряд шестнадцатеричного слова, хранимого в регистровой паре Dig43:Dig21, содержит знак. Если он равен лог. 0, то измеренная температура $\geq 0^{\circ}C$, и выполнение программы продолжается с метки Positiv. Здесь на основании значения старшего разряда Dig21 определяется содержимое позиции после десятичной точки. Если разряд 7 = лог. 1, то позиция после точки имеет вид x,5°C, и сохраняется в регистре Dig21. В противном случае позиция после точки имеет вид x,0°C, а регистр Dig21 обнуляется.

Если старший разряд регистра Dig43 соответствует отрицательному знаку, то считанное значение в Dig43, включая позицию после точки в Dig21, преобразовывается в положительное шестнадцатеричное число посредством дополнения до одного или до двух.

Теперь в любом случае температура представлена в виде положительного шестнадцатеричного числа и может быть преобразована в десятичное значение в части программы, обозначенной меткой HexDez. Преобразование выполняется путем последовательного деления с остатком (см. первый раздел этой главы). Поскольку для отображения значения имеется только четыре цифры, предполагается, что результат меньше 99,5°C. Если в каком-либо случае применения требуется диапазон до 125°C, в программу необходимо внести следующее изменение: поскольку установленное значение в любом случае удовлетворяет допуску ±2°C, можно отказаться от позиции после десятичной точки.

Деление реализуется посредством постоянного вычитания числа $0A до тех пор, пока не будет получена отрицательная разница, после чего последнее вычитание отменяется. Последняя разница ≥ 0 представляет собой остаток, а количество выполненных вычитаний — частное от деления. Поскольку предполагается, что результат ≤ 99, остаток от деления соответствует целой, а частное — дробной части числа. Если десятичный разряд равен нулю, то этот нуль отображается в виде пробела, которому в таблице HexTo7Segm соответствует код E_h.

В завершение позиция после точки и целая часть результата объядиняются в Dig21, а дробная часть и знак (флаг T) — в Dig43. Если результат отрицательный, то на табло отображается знак минуса, которому в таблице HexTo7Segm соответствует код F_h. В противном случае, вместо него отображается пробел (код E_h).

Описание главной программы

После сброса по включению питания происходит переход по адресу $000 — к части инициализации, обозначенной меткой Initial. Когда инициализирован указатель стека, порты B и D определяются как тристабильные выводы. Поскольку порты PB0 и PB1 работают как выходы с открытым коллектором, после такой инициализации ими можно управлять с помощью регистра направления передачи данных DDRB.

Инициализация порта D выполняется здесь только для полноты изложения, поскольку регистр DDRD после поступления сигнала сброса по включению питания автоматически обнуляется, и, следовательно, порт D определен как вход.

По сигналу сброса регистр TCCR0 также автоматически обнуляется, останавливая тем самым T/C0. Для того чтобы запустить счетчик T/C0, для него в качестве входного такта определяется такт системной синхронизации, деленный с помощью мультиплексора на 1024. Для этого в регистр TCCR0 записывается значение $05 (см. табл. 4.2), и T/C0 начинает счет.

В завершение части инициализации сбрасываются флаги десятичной точки DP4, DP3 и DP1, а также устанавливаются флаги DP2 и Ack.

В основной части программы модуль вывода инициализируется с помощью микросхемы SAA1064. Процесс инициализации завершается условием останова.

Если после этого не последовало подтверждения, попытка инициализации повторяется, в противном случае выполняется переход к бесконечному циклу, обозначенному меткой `Next`, в котором каждые 0,25 с считывается и отображается значение температуры.

В сегменте памяти EEPROM размещена таблица `HexTo7Segm` с семисегментными кодами, соответствующими шестнадцатеричным значениям 0_h...F_h, включая служебные символы E_h (пробел) и F_h (минус). При программировании микросхемы AVR отдельно должен быть создан соответствующий файл с расширением `*.eep`.

17 Приложение

Технические характеристики микроконтроллеров семейства AVR

Для успешного применения интегральных схем важно знать и соблюдать пределы рабочих параметров, чтобы избежать сбоев и повреждений. Технические характеристики, параметры и временные диаграммы для представителей базовой серии микроконтроллеров AVR находятся на прилагаемом к книге компакт-диске в папке `\Atmel\Sheets` в файлах `90S1200.pdf`, `90S2313.pdf`, `90S4414.pdf` и `90S8515.pdf`. Эти паспорта, предоставленные компанией Atmel, кроме статических параметров потребляемого тока и рабочего напряжения четырех микросхем, также содержат описание таких динамических характеристик как длительность импульсов и их фронтов, диапазоны частот и др.

Кроме того, в них представлено множество различных графиков зависимостей, например, потребляемого тока от частоты системной синхронизации и питающего напряжения; выходного тока от выходного напряжения и т.д. Обновленные версии этих паспортов можно также загрузить с домашней страницы компании Atmel (`www.atmel.com`).

Документированные компанией Atmel ошибки и пути их устранения

Несмотря на то, что микроконтроллеры семейства AVR от компании Atmel разрабатываются и тестируются с большой тщательностью, при поступлении на рынок они все же время от времени проявляют те или иные сбои. Рассмотрим кратко перечень официально опубликованных ошибок.

AT90S1200:

- Могут возникнуть проблемы при последовательном программировании с рабочим напряжением ниже 3 В. Для внутрисхемного программирования рекомендуется выбирать напряжение выше 3 В.
- Если микроконтроллер AT90S1200 работает на максимальной допустимой частоте системной синхронизации, то при последовательном программировании памяти EEPROM может не выполняться верификация данных. Эта проблема не зависит от используемой тактовой частоты SPI. Рекомендуется или уменьшить частоту системной синхронизации, или отказаться от функции верификации при последовательном программировании памяти EEPROM.
- Если во время программирования памяти EEPROM на микросхему поступит сигнал сброса, то результат программирования будет неопределенным, поскольку регистр адреса EEPROM в результате сброса обнулится, и неопределенным будет содержимое как целевой ячейки памяти, так и ячейки по адресу $00. Если нельзя гарантировать, что во время программирования не

поступит сигнал сброса, рекомендуется отказаться от использования ячейки памяти EEPROM с адресом $00.

AT90S2313:

- Проблемы, характерные для микроконтроллера AT90S1200, могут также встречаться и при работе с AT90S2313.
- В некоторых образцах возможна ситуация, когда при высоких значениях питающего напряжения разряды блокировки не обнуляются при поступлении команды очистки кристалла. В результате микросхема остается заблокированной для дальнейшего программирования. Перед тем как выполнить очистку кристалла рекомендуется снизить питающее напряжение до уровня менее 4 В.

AT90S4414 и AT90S8515:

- Проблемы, характерные для микроконтроллера AT90S1200, могут также встречаться и при работе с AT90S4414 и AT90S8515.
- Если после команды пропуска (`sbrs`, `sbrc`, `sbis`, `sbic` или `cpse`) следует двухбайтная команда (`lds`, `sts`), то для ее выполнения требуется три тактовых цикла. Если во время первого или второго такта выполнения команды поступает запрос на прерывание, то в стек вместо адреса возврата из подпрограммы обработки прерывания помещается адрес второго командного слова. Это означает, что по завершении подпрограммы обработки прерывания в качестве следующей команды для выполнения будет распознано второе слово двухбайтной команды.

  Для того чтобы обойти такую ситуацию, рекомендуется компилировать С-программы с помощью компилятора IAR версии 1.40b или выше. Если в программе разрешены прерывания, то во время программирования в ассемблере следует избегать использования команд пропуска перед двухбайтными командами.
- Если в момент сброса происходит изменение сигнала на выводе SCK или этот вывод не подключен, то в некоторых случаях может быть неопределенным значение инициализации флага прерывания от SPI. В результате данное прерывание может быть вызвано еще до того как будет разрешено. Во избежание этого, перед разрешением прерывания от SPI соответствующий флаг рекомендуется сбрасывать.
- В режиме SPI-Master может получиться так, что завершенная передача начинается еще раз, если точно в момент нарастающего фронта тактового импульса, которым заканчивается передача, в буфер SPI записывается новый байт. Новый байт записывать рекомендуется только после полного завершения текущей передачи.

Этот список постоянно обновляется и может быть также найден на домашней странице компании Atmel (`www.atmel.com`).

18 Содержимое прилагаемого к книге компакт-диска

Папка DATEN

Для того чтобы избавить читателя от трудоемкого поиска технических паспортов описанных в этой книге интегральных схем, они были включены в состав парилагаемого к книге компакт-диска:

- 43256.pdf — SRAM;
- 74HC00.pdf — 4 вентиля "И-НЕ";
- 74HC125.pdf — 4 вентиля буфера шины с тристабильными выходами;
- 74HC165.pdf — сдвиговой регистр;
- 74HC4051.pdf — КМОП-ключ;
- 74HC4053.pdf — КМОП-ключ;
- 74HC573.pdf — восемь фиксаторов;
- 74HC595.pdf — сдвиговой регистр;
- 78L05.pdf — стабилизатор напряжения;
- DS9636A.pdf — интерфейс RS423;
- DS9637A.pdf — интерфейс RS423;
- DS9638.pdf — интерфейс RS422;
- HD44780.pdf — контроллер ЖК-дисплея;
- LF356.pdf — операционный усилитель;
- LM311.pdf — компаратор;
- LM317T.pdf — стабилизатор напряжения;
- LM336.pdf — опорный диод;
- LM75.pdf — датчик температуры с интерфейсом I^2C;
- MAX202.pdf — интерфейс RS232;
- MAX232.pdf — интерфейс RS232;
- MAX5154.pdf — ЦАП;
- MAX809.pdf — схема сброса;
- MAX811.pdf — схема сброса;
- S24022.pdf — контроллер сброса + EEPROM;
- SAA1064.pdf — схема управления светодиодными индикаторами с интерфейсом I^2C;

- TL082.pdf — операционный усилитель;
- TL084.pdf — операционный усилитель;
- TLV1572.pdf — АЦП.

Все перечисленные паспорта предоставлены непосредственно разработчиками (последние версии могут быть бесплатно загружены с соответствующих домашних страниц, перечисленных в конце книги). Выражаю сердечную признательность всем компаниям за разрешение включить эти файлы в состав компакт-диска.

Папка Atmel

Подпапка Sheets

Здесь находятся технические паспорта представителей базовой серии микроконтроллеров AVR. Все они предоставлены компанией Atmel и их последние версии могут быть бесплатно загружены со страницы www.atmel.com. Выражаю сердечную признательность компании Atmel за разрешение включить эти файлы в состав компакт-диска.

Подпапка Applicat

Здесь находятся указания к применению представителей базовой серии микроконтроллеров семейства AVR, предоставленные компанией Atmel:

- AVR000.pdf — определения регистров и имен разрядов для микроконтроллеров AVR;
- AVR032.pdf — командные файлы компоновки для компилятора IAR ICCA90;
- AVR034.pdf — смешивание ассемблерного и С-кода с помощью встроенных средств IAR для AVR;
- AVR070.pdf — модификация AT90ICEPRO для поддержки эмуляции AT90S8535;
- AVR100.pdf — доступ к памяти EEPROM микроконтроллера AT90S1200;
- AVR102.pdf — подпрограммы копирования блоков данных;
- AVR128.pdf — настройка и использование аналогового компаратора;
- AVR134.pdf — часы реального времени с использованием асинхронного таймера;
- AVR180.pdf — внешняя защита от провалов напряжения;
- AVR200.pdf — подпрограммы умножения и деления;
- AVR202.pdf — 16-разрядная арифметика;
- AVR204.pdf — BCD-арифметика;
- AVR220.pdf — сортировка методом “пузырька”;

- AVR222.pdf — фильтр на восемь позиций с применением метода скользящего среднего;
- AVR236.pdf — проверка памяти программ с помощью циклического избыточного кода;
- AVR240.pdf — кнопочная панель 4x4 для выхода из режима пониженного энергопотребления;
- AVR242.pdf — 8-разрядная схема управления светодиодными индикаторами в мультиплексном режиме с применением кнопочной панели 4x4;
- AVR300.pdf — программная реализация интерфейса ведущего устройства I^2C;
- AVR302.pdf — программная реализация интерфейса ведомого устройства I^2C;
- AVR304.pdf — программная реализация полудуплексного приемопередатчика UART, активируемого по прерыванию;
- AVR305.pdf — программная реализация полудуплексного компактного приемопередатчика UART;
- AVR313.pdf — взаимодействие с клавиатурой PC AT;
- AVR320.pdf — программная реализация ведущего устройства SPI;
- AVR360.pdf — контроллер шагового двигателя;
- AVR400.pdf — дешевый АЦП;
- AVR401.pdf — АЦП с точностью 8 разрядов;
- AVR910.pdf — внутрисистемное программирование;
- ICEPRO.pdf — понимание регистров ввода/вывода AVR ICEPRO;
- L_Delay.pdf — формирование долгих задержек с помощью микроконтроллеров AVR.

Подпапка Software

Здесь находятся программы, соответствующие перечисленным выше указаниям к применению (файлы с расширением .exe — это самораспаковывающиеся архивы; с расширением .zip — архивы, которые распаковываются с помощью программы WinZIP, а файлы с расширением .asm — обычные файлы исходного кода AVR-ассемблера):

- 8515def.inc — определения регистров и имен разрядов для AT90S8515;
- 4414def.inc — определения регистров и имен разрядов для AT90S4414;
- 2313def.inc — определения регистров и имен разрядов для AT90S2313;
- 1200def.inc — определения регистров и имен разрядов для AT90S1200;
- avr000.exe — определения регистров и имен разрядов для микроконтроллеров семейства AVR;

- avr032.zip — командные файлы компоновки для компилятора IAR ICCA90;
- avr100.asm — доступ к памяти EEPROM в AT90S1200;
- avr102.asm — подпрограммы копирования блоков данных;
- avr128.asm — настройка и использование аналогового компаратора;
- avr134.c — часы реального времени с использованием асинхронного таймера;
- avr200.exe — подпрограммы умножения и деления;
- avr202.asm — 16-разрядная арифметика;
- avr204.asm — BCD-арифметика;
- avr220.asm — сортировка методом "пузырька";
- avr222.asm — фильтр на восемь позиций с применением метода скользящего среднего;
- avr235.asm — проверка памяти программ с помощью циклического избыточного кода;
- avr240.asm — кнопочная панель 4x4 для выхода из режима пониженного энергопотребления;
- avr242.asm — 8-разрядная схема управления светодиодными индикаторами в мультиплексном режиме с применением кнопочной панели 4x4;
- avr300.asm — программная реализация интерфейса ведущего устройства I^2C;
- avr302.asm — программная реализация интерфейса ведомого устройства I^2C;
- avr304.asm — программная реализация полудуплексного приемопередатчика UART, активируемого по прерыванию;
- avr305.asm — программная реализация полудуплексного компактного приемопередатчика UART;
- avr313.zip — взаимодействие с клавиатурой PC AT;
- avr320.asm — программная реализация ведущего устройства SPI;
- avr400.asm — дешевый АЦП;
- avr401.asm — АЦП с точностью 8 разрядов;
- avr910.asm — внутрисистемное программирование.

Подпапка Tools

Здесь в виде самораспаковывающихся архивов находятся пакеты установки AVR-ассемблера и среды AVR-Studio:

- asmpack.exe — AVR-ассемблер 1.30;
- astudio.exe — AVR-Studio 2.0.

Порядок установки описан в главах 13 и 14.

Папка Programm

Здесь находятся программы, рассмотренные в примерах из этой книги:

- EEPR8515.asm — программирование памяти EEPROM и операции чтения для микроконтроллера AT90S8515;
- EEPR1200.asm — программирование памяти EEPROM и операции чтения для микроконтроллера AT90S1200;
- 161bcd.asm — средства BCD-арифметики;
- 162inout.asm — базовые операции ввода/вывода;
- 163lcd.asm — подключение ЖК-модуля;
- 164tc0.asm — формирование импульсов определенной длины с помощью таймера/счетчика T/C0;
- 165tc0.asm — программная реализация автоматической перезагрузки T/C0;
- 166tc1.asm — выработка с помощью T/C1 импульсов с частотой 50 Гц и коэффициентом заполнения 0,025;
- 167da.asm — трехканальный ЦАП с разрешающей способностью 10 бит;
- 168ad.asm — реализация с помощью T/C1 четырехканального АЦП с двойным интегрированием и разрешающей способностью 11 бит;
- 168lm311.asm — реализация с помощью T/C1 четырехканального АЦП с двойным интегрированием и разрешающей способностью 11 бит;
- 169uart.asm — программная реализация приемопередатчика UART для микроконтроллера AT90S1200;
- 16105154.asm — подключение к микроконтроллеру AT90S8515 микросхемы ЦАП MAX5154 через интерфейс SPI;
- 1611shft.asm — расширение портов ввода/вывода микроконтроллера AT90S4414 с помощью интерфейса SPI;
- 16121572.asm — программная реализация интерфейса SPI для модели AT90S1200 при подключении АЦП TLV1572;
- 1615mstr.asm — Использование микроконтроллера AVR в качестве ведущего устройства I^2C.

INTERNET-ССЫЛКИ

Ниже перечислены Internet-адреса некоторых компаний, упомянутых в книге:

- Atmel (производитель микроконтроллеров семейства AVR) — `http://www.atmel.com`;
- Ventec (дистрибьютор микроконтроллеров семейства AVR) — `http://www.ventec.com`;
- E-LAB (Pascal-компилятор для микроконтроллеров семейства AVR) — `http://www.e-lab.de`;
- IAR Systems (C-компилятор для микроконтроллеров семейства AVR) — `http://www.iar.com`;
- Silicon Studio (BASIC-компилятор для микроконтроллеров семейства AVR) — `http://www.sistudio.com`;
- Adobe (разработчик программы Acrobat Reader, необходимой для чтения файлов `.pdf`) — `http://www.adobe.com`;
- Hitachi (разработчик полупроводниковых элементов) — `http://www.hitachi.com`;
- MAXIM (разработчик полупроводниковых элементов) — `http://www.maxim-ic.com`;
- Motorola (разработчик полупроводниковых элементов) — `http://www.motorola.com`;
- National Semiconductor (разработчик полупроводниковых элементов) — `http://www.national.com`;
- NEC (разработчик полупроводниковых элементов) — `http://www.nec.com`;
- Philips (разработчик полупроводниковых элементов) — `http://www-us.semiconductors.philips.com`;
- Summit (разработчик полупроводниковых элементов) — `http://www.summitmicro.com`;
- Texas Instruments (разработчик полупроводниковых элементов) — `http://www.ti.com`.

Предметный указатель

A

AVR-Studio, 298
Окна, 300
Установка, 298

C

CISC, 17

E

EEPROM, 26, 64
EPROM, 18

L

LIFO, 27

M

MCUCR, 49

R

RamEnd, 43
RISC, 17

S

SRAM, 26, 41
SREG, 48
STK200, 314
Кнопочные выключатели, 323
Подключения портов, 318
Программное обеспечение, 323
Функции перемычек, 317

U

UART, 123
Программная реализация, 392
Ресивер, 131
Скорость передачи, 137
трансмиттер, 130
Физическое устройство, 129

А

Адресация
Ведомых устройств, 158
Констант в памяти программ, 79
Косвенная памяти данных, 75
Косвенная памяти данных с последующим приращением адреса, 77
Косвенная памяти данных с предварительным уменьшением адреса, 77
Косвенная памяти программ, 81
Косвенная регистров в микроконтроллере AT90S1200, 76
Круговая, 82
Относительная памяти данных, 78
Относительная памяти программ, 82
Прямая двух регистров Rd и Rr, 73
Прямая области ввода/вывода, 73
Прямая одного регистра, 72
Прямая памяти данных, 74
Прямая памяти программ, 80
Аналоговый компаратор, 166, 381
Арбитраж шины I²C, 164
Арифметико-логическое устройство, 25
Арифметико-логическом устройство, 41
Архитектура
CISC, 17, 20
RISC, 17, 20, 21
Гарвардская, 21
Памяти, 25
Ассемблер, 275
Выражения, 293
Операнды, 293
Операторы, 294
Синтаксис, 280
Установка, 279
Функции, 293
АЦП
Программная реализация интерфейса SPI, 413
Четырехканальный с двойным интегрированием и разрешением 11 разрядов, 381

Б

Байты сигнатуры, 180
Бит
Подтверждения, 156
Биты
Условий, 48

В

Вывод
/RD, 177
/RESET, 32
/RST, 88
/SS, 174
/WR, 177
AIN0, 174
AIN1, 174
ALE, 32, 53
DOUT, 404
GND, 28
ICP, 32, 177
INT0, 177
INT1, 177
MISO, 173
MOSI, 174
OC1, 174
OC1A, 177
OC1B, 32
RxD, 177
SCK, 173
T0, 175, 177
T1, 175, 177
TxD, 177
UPO, 404
VCC, 28
XTAL1, 32, 193
XTAL2, 32

Выводы
PINA, 172
PIND, 178
PINB, 175
PINC, 176
Выражения, 293
Выход с открытым коллектором, 178

Г

Гарвардская архитектура, 21

Д

Датчик температуры, 419
Двоично-десятичная арифметика, 327
Директива
.BYTE, 285
.CSEG, 285
.DB, 286
.DEF, 286
.DEVICE, 287
.DSEG, 287
.DW, 288
.ENDMACRO, 288
.EQU, 288
.ESEG, 289
.EXIT, 289
.INCLUDE, 289
.LIST, 290
.LISTMAC, 290
.MACRO, 291
.NOLIST, 291
.ORG, 291
.SET, 292
Директивы ассемблера, 280, 284
Дребезг контакта кнопки, 379

Ж

Ждущий режим, 98
ЖК-модули, 341

З

Знакогенератор, 344

И

Инжекции горячих электронов, 62
Интерфейс
ISP STK200, 319
RS232 STK200, 319
RS232C, 124
RS422A, 126
RS423A, 125
SPI, 139, 399, 407, 413
V.10, 126
V.11, 125
V.24, 124
ЖКИ STK200, 320
Токовый, 123

К

Команда
adc, 222
add, 223
adiw, 223
and, 224
andi, 224
asr, 225
bclr, 225
bld, 226
brbc, 226
brbs, 227
brcc, 227
brcs, 228
breq, 228
brge, 229
brhc, 229
brhs, 230
brid, 230
brie, 231
brlo, 231
brlt, 232
brmi, 232
brne, 233
brpl, 233

Команда
brsh, 234
brtc, 234
brts, 235
brvc, 235
brvs, 236
bset, 236
bst, 237
call, 80, 237
cbi, 238
cbr, 238
clc, 239
clh, 239
cli, 239
cln, 240
clr, 240
cls, 240
clt, 241
clv, 241
clz, 241
com, 242
cp, 242
cpc, 243
cpi, 243
cpse, 244
dec, 245
eor, 245
icall, 81, 246
ijmp, 81, 246
in, 247
inc, 247
jmp, 80, 248
ld, 248
ldd, 249, 250
ldi, 251
lds, 251
lpm, 252
lsl, 252
lsr, 253
mov, 253
mul, 254
neg, 254
nop, 255
or, 255
ori, 256
out, 256
pop, 57, 257
push, 57, 257
rcall, 83, 258
ret, 258
reti, 259
rjmp, 83, 85, 259
rol, 260
ror, 260
sbc, 261
sbci, 261
sbi, 262
sbic, 262
sbis, 263
sbiw, 263
sbr, 264
sbrc, 264
sbrs, 265
sec, 265
seh, 266
sei, 266
sen, 266
ser, 267
ses, 267
set, 267
sev, 268
sez, 268
sleep, 99, 268
st, 269
std, 270, 271
sts, 272
sub, 272
subi, 273
swap, 273
tst, 274
wdr, 274
Команды, 222
Арифметические, 211
Логические, 211
Пересылки, 206
Перехода, 214
Поразрядных операций, 219
Комментарий, 280
Конвейерная обработка, 21
Коэффициент заполнения, 364

Л

Линия
/SS, 140, 143
MISO, 140, 141
MOSI, 140, 141
RxD, 125
SCK, 141, 142
SCL, 152
SDA, 153
TxD, 125

М

Меню
AVR-Studio, 308
Ассемблера, 282
Программного обеспечения ISP, 324
Метки, 280
Микросхема
DS9636A, 126
DS9637A, 126
DS9638, 126
HD44780, 341
LM75, 419, 429
MAX202, 320
MAX5154, 399
MAX809, 89
MAX811, 89
S24022, 89
SAA1064, 424, 429
TLV1572, 413
MAX232, 124
Мнемоники команд, 280

О

Область ввода/вывода, 26, 45
Обработка прерываний, 84, 91
Окно
IO, 305
Memory, 303
Message, 308
Processor, 302
Registers, 301
Окно
Trace, 307
Watch, 307
Исходного кода, 300
Операнды, 293
Операторы, 294
Осциллятор
RC, 34
Встроенный кварцевый STK200, 322
Интегрированный кварцевый, 33
Отладка программ, 298

П

Память
EEPROM, 26, 64
EPROM, 18
Flash-EPROM, 59
RamEnd, 43
Адресация констант в памяти программ, 79
Архитектура, 25
Внешняя SRAM, 52
Внутренняя SRAM, 51
Команд, 59
Косвенная адресация памяти данных, 75
Косвенная адресация памяти данных с последующим приращением адреса, 77
Косвенная адресация памяти данных с предварительным уменьшением адреса, 77
Косвенная адресация памяти программ, 81
Косвенная адресация регистров в микроконтроллере AT90S1200, 76
Область ввода/вывода, 26, 45
Относительная адресация памяти данных, 78
Относительная адресация памяти программ, 82
Подключение внешних блоков, 44
Программ, 26

Память
Программирование, 179
Процесс программирования, 61
Процесс стирания, 63
Прямая адресация двух регистров Rd и Rr, 73
Прямая адресация области ввода/вывода, 73
Прямая адресация одного регистра, 72
Прямая адресация памяти данных, 74
Прямая адресация памяти программ, 80
Расширение в STK200, 323
Регистровый файл, 44
Режимы адресации, 72
Статическая, 26, 41
Панель инструментов
Ассемблера, 281
Среды ISP, 326
Передача данных
Асинхронная, 123
Последовательная, 151
Синхронная, 139
Перемычки STK200, 317
Переход
Условный, 205
Плавающий затвор, 61
Подпрограмма обработки прерывания, 27
Поиск ошибок, 283
Порт
A, 28, 172
B, 28, 173
C, 31, 175
D, 32, 176
Порты, 169
STK200, 318
Операции ввода/вывода, 334
Расширение с помощью SPI, 407
Предварительный делитель частоты, 100
Прерывания, 26
Внешние, 92
Источники, 84
Обработка, 84, 91
Разрешение, 84
Программирование
Памяти, 61, 179
Памяти EEPROM, 68
Параллельный режим, 181
Последовательный режим, 192
Процесс, 180
Разряды блокировки, 190
Разряды предохранения, 190

P

Разряд
ACD, 99, 167
ACI, 167
ACIC, 168
ACIE, 167
ACIS0, 168
ACIS1, 168
ACO, 167
CHR9, 137
COM1A0, 109
COM1A1, 109
COM1B0, 109
COM1B1, 109
CPHA, 144, 148
CPOL, 148
CTC1, 110
DORD, 148
EEMWE, 68
EERE, 67
EEWE, 67
FE, 135
FSTRT, 179
ICES1, 110
ICF1, 97
ICNC1, 111
INT0, 93
INT1, 93
INTF0, 94
INTF1, 94

Разряд
ISC10, 50
ISC11, 50
LB1, 179
LB2, 179
MSTR, 148
OCF1A, 96
OCF1B, 97
OCIE1A, 95
OCIE1B, 95
OR, 135
PWM10, 109
RCEN, 179
RXB8, 137
RXC, 134
RXCIE, 136
RXEN, 133, 136
SE, 50, 98
SM, 50
SPE, 148
SPIE, 140, 148
SPIEN, 179
SPIF, 149
SPR0, 148
SPR1, 148
SRE, 49
SRW, 49
TICIE1, 95
TOIE0, 96
TOIE1, 95
TOV0, 97
TOV1, 96
TXB8, 137
TXC, 131, 135
TXCIE, 136
TXEN, 131, 136
UDRE, 131, 135
UDRIE, 136
WCOL, 149
WDE, 99, 121
WDTOE, 122
Направления передачи данных, 159

Разряды
CS10–CS12, 109
PWM11, 109
WDP0–WDP2, 121
Блокировки памяти программ, 179
Предохранения, 179
Условий, 200

Регистр
ACSR, 167
DDRA, 172
DDRB, 175
DDRC, 176
DDRD, 177
EEAR, 66
EECR, 66
EEDR, 66
GIFR, 91, 93
GIMSK, 91, 93
MCUCR, 49
OCR1A, 113
OCR1B, 113
SPCR, 140, 148
SPDR, 149
SPSR, 149
SREG, 48
TCCR0, 102, 104
TCCR1A, 108
TCCR1B, 109
TCNT0, 102, 104
TCNT1, 107
TIFR, 91, 96
TIMSK, 91, 94
UBRR, 129, 137
UCR, 129, 136
UDR, 129, 134
USR, 129, 134
WDTCR, 121
Данных порта A, 172
Данных порта B, 173
Данных порта C, 175
Данных порта D, 176
Захвата по входу ICR1, 113
Состояния, 48
Счетный, 104, 107
Указатель, 422, 427

Регистровый файл, 44
Регистры
UART, 134
X, 44
Y, 44
Z, 44
Двойной длины, 44
Сравнения OCR1A и OCR1B, 113
Режим пониженного энергопотребления, 99
Режимы адресации, 72

С

Сброс, 84
Аппаратный, 90
Внешний, 90
Источники, 86
От сторожевого таймера, 91
По включению питания, 87
Схема, 85
Светодиодное табло, 424
Светодиоды, 322
Семейство AVR, 21
AT90S1200, 29, 413
AT90S4414, 407
AT90S8515, 30, 399
Базовая серия, 20, 23
Базовая структура, 24
Обзор команд, 206
Периферийные функции, 27
Программирование, 35
Расположение выводов, 28
Система команд, 199
Стек, 54
Стендовые испытания, 36
Схема сброса, 85
Таймеры/счетчики, 100
Симулятор, 312
Синхронизация
При обращении к внешней памяти SRAM, 53
При обращении к внутренней памяти SRAM, 51
Система управления, 41
Спящие режимы центрального процессора, 97
Стек, 26, 54
Аппаратный, 55
Сторожевой таймер, 120
Структура команд, 25

Т

Таймер/счетчик, 100
T/C0, 102
T/C1, 104
Автоматическая перезагрузка, 360
Области применения, 118
Широтно-импульсная модуляция, 114
Такт
Системной синхронизации, 33
Сторожевого таймера, 120
Туннелирование, 63

У

Указатель
X, 248, 269
Y, 249, 270
Z, 250, 271
стека, 55, 56
Стека, 27
Условие
Завершения передачи, 158
Начала передачи, 156
Условный переход, 205
Без учета знака, 205
Прямой опрос флагов, 206
С учетом знака, 206

Ф

Фильтр нижних частот, 369
Флаг
ACI, 167
C, 48, 201
FE, 133, 135
H, 49, 204
I, 49, 205

Флаг
ICF1, 97
INTF0, 94
INTF1, 94
N, 48, 203
OCF1A, 96
OCF1B, 97
OR, 135
RXC, 134
S, 49, 204
SPIF, 140, 149
T, 49, 205
TOV0, 97, 103
TOV1, 96
TXC, 131, 135
UDRE, 131, 135
V, 49, 203
WCOL, 147, 149
Z, 48, 203
Знака, 49, 204
Копирования, 49, 205
Нулевой, 48, 203
Общего разрешения прерываний, 49, 205
Отрицательного результата, 48, 203
Переноса, 48, 201
Флаг
Переполнения при вычислениях в дополнительных кодах, 49, 203
Половинного переноса, 49, 204
Флаги, 48, 200
Прерываний, 91
Таймеров/счетчиков, 91
Формат
Atmel Generic, 277
Intel Intellec 8/MDS, 277
Motorola S-record, 278
Функции, 293

Ц

ЦАП
MAX5154, 399
Трехканальный с разрешением 10 разрядов, 369
Центральный процессор, 41
Спящие режимы, 97

Ш

Шина I^2C, 151, 419, 424, 429
Широтно-импульсная модуляция, 109, 114, 371

ББК 32.973-04
Т 65
УДК 004.312

Трамперт В.

Т65 AVR-RISC мікроконтролери.: Пер. з нім. — К.: "МК-Пресс", 2006. — 464 с., іл.

ISBN 966–8806-07-7 (рос.)

В книзі вичерпно описана базова серія мікроконтролерів сімейства AVR від компанії Atmel, побудованих на базі прогресивної архітектури RISC з використанням флеш-пам'яті EPROM, що програмується. Крім того, докладно розглянуто програмування мікроконтролерів цієї серії на мові асемблеру, а також середовище AVR-Studio та програмно-апаратний набір STK200.

Книга призначена для всіх, хто вже володіє основними пізнаннями в галузі побудови та функціонування мікрокомп'ютерів, бажає вивчити однокристальні мікроконтролери AVR і з успіхом впроваджувати в життя задачі внутрішньоплатного управління.

ББК 32.973-04

Головний редактор: Ю. О. Шпак

Ніяка частина цього видання ні в яких цілях не може бути відтворена в будь-якій формі та будь-яким засобами, як то електронні чи механічні, включаючи фотокопіювання та запис на магнітний носій, якщо на це нема письмового дозволу видавництва Franzis' Verlag GmbH.

Authorized translation from the German language edition published by Franzis', Copyright © 2003.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

Russian language edition published by MK-Press according to the Agreement with Franzis' Verlag GmbH, Copyright © 2006.

ISBN 966–8806-07-7 (рос.)
ISBN 3-7723-5476-9 (нім.)

© "МК-Пресс", 2006
© Franzis' Verlag GmbH, 2003