

*Серия «Библиотека инженера»*

**М. С. Голубцов**

# **МИКРОКОНТРОЛЛЕРЫ AVR: ОТ ПРОСТОГО К СЛОЖНОМУ**

**Москва  
СОЛОН-Пресс  
2003**

## М. С. Голубцов

Микроконтроллеры AVR: от простого к сложному / М. С. Голубцов — М.: СОЛОН-Пресс, 2003. 288 с. — (Серия «Библиотека инженера»)

ISBN 5-98003-034-4

Прочитав эту книгу, читатели познакомятся с микроконтроллерами семейства AVR, программным обеспечением, необходимым для конструирования и программирования схем на основе этих микроконтроллеров, а при желании познакомятся с их применением на практике, собрав какую-либо из описанных в книге практических конструкций. Большая часть программ, описанных в книге, имеется на прилагаемом к книге компакт-диске.

Книга может быть полезна радиолюбителям, студентам, а также всем, кто интересуется применением микроконтроллеров семейства AVR.

К книге прилагается компакт-диск.

*Эту книгу можно заказать по почте (наложенным платежом — стоимость 227 руб.) двумя способами:*

- 1) выслать почтовую открытку или письмо по адресу: 123242, Москва, а/я 20;
- 2) передать заказ по электронной почте (e-mail) по адресу: [magazin@solon-r.ru](mailto:magazin@solon-r.ru).

Необходимо написать полный адрес, по которому выслать книги.

Обязательно указывать индекс и Ф. И. О. получателя!

При наличии — указать телефон, по которому с вами можно связаться, и адрес электронной почты (E-mail).

**Цена действительна до 15 апреля 2003 г.**

Вы можете в любое время получить свежий каталог издательства «СОЛОН-Пресс» по Интернету, послав пустое письмо на робот-автоответчик по адресу

[katalog@solon-r.ru](mailto:katalog@solon-r.ru),

а также подписаться на рассылку новостей о новых книгах издательства, послав письмо по адресу

[news@solon-r.ru](mailto:news@solon-r.ru)

с текстом «SUBSCRIBE» (без кавычек) в теле письма.

# Введение

Эта книга о микроконтроллерах семейства AVR, производимых известной фирмой Atmel Corporation ([www.atmel.com](http://www.atmel.com)).

Следует отметить, что перед вами не учебник и не инструкция по изготовлению полноценных систем на основе микроконтроллеров, хотя довольно подробно описываются основные их концепции. Нельзя рассматривать книгу также как справочник — в ней достаточно много материала, но тем не менее некоторые вопросы могут быть не освещены или освещены недостаточно подробно. Если необходима документальная информация, имеет смысл попутно с чтением книги обращаться к фирменным описаниям микросхем для уточнения. Например, в книге не рассматривается описание процесса программирования микроконтроллера, предполагается, что читатели воспользуются описанием несложного самодельного программатора и готовой программой к нему. Автор книги предоставляет все материалы, программы и схемы «как есть», без каких-либо гарантий правильности и соответствия фирменным описаниям и не несет никакой ответственности по материальному, или другим видам ущерба, причиненным в результате использования информации, приведенной в настоящей книге.

Прежде чем говорить о микроконтроллерах, давайте выясним, почему они играют такую большую роль в современном мире. Научно-технический прогресс неумолимо идет вперед, в результате не только в промышленной, но и в бытовой технике все шире используются встроенные компьютерные системы на основе микроконтроллеров. Они широко применяются в персональных компьютерах и их периферийных устройствах, стиральных машинах, музыкальных центрах и т. д. Средний импортный автомобиль имеет порядка 15 микроконтроллеров, управляющих различными системами автомобиля.

Основная цель книги — помочь разобраться в том, что такое однокристалльный микроконтроллер, как он работает и как на его основе можно создавать разнообразные устройства, применяющиеся в технике, в быту, в повседневной практической деятельности.

Современный микроконтроллер — довольно сложное устройство, работу которого не удастся описать в деталях вне связи его с дополнительными внешними устройствами. Поэтому в книге имеется достаточно много фрагментов реальных схем, а в 6-й главе — описания нескольких законченных устройств.

Автор надеется, что книга окажется полезной не только будущим конструкторам современного оборудования, но и всем тем, кто стремится расширить свои знания в области применения микропроцессорной техники.

# Глава 1.

## Микроконтроллер

### 1.1. Знакомство с микроконтроллером

В этой главе мы кратко познакомимся с микроконтроллерами. Микроконтроллеры являются сердцем многих современных устройств и приборов, в том числе и бытовых. Самой главной особенностью микроконтроллеров, с точки зрения конструктора-проектировщика, является то, что с их помощью легче и зачастую гораздо дешевле реализовать различные схемы.

На рис. 1.1 изображена структурная схема типичного современного микроконтроллера.

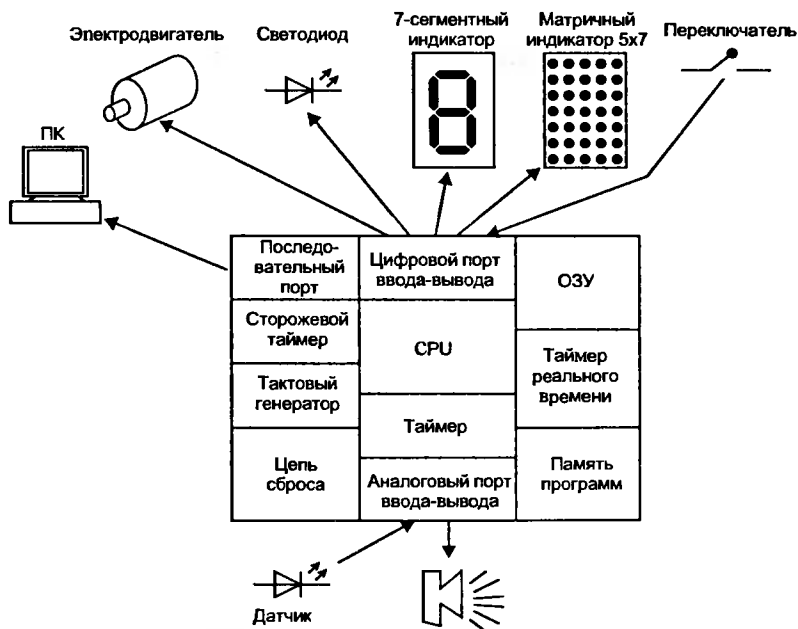


Рис. 1.1. Структурная схема микроконтроллера

Из рисунка видно, что микроконтроллер может управлять различными устройствами и принимать от них данные при минимуме дополнительных узлов, так как большое число периферийных схем уже имеется непосредственно на кристалле микроконтроллера. Это позволяет уменьшить размеры конструкции и снизить потребление энергии от источника питания.

Для сравнения: при использовании традиционных микропроцессоров приходится все необходимые схемы сопряжения с другими устройствами реализовывать на дополнительных компонентах, что увеличивает массу, размеры и потребление электроэнергии.

Давайте рассмотрим типичные схемы, присутствующие в микроконтроллерах.

1. Центральное процессорное устройство (ЦПУ) — сердце микроконтроллера. Оно принимает из памяти программ коды команд, декодирует их и выполняет. ЦПУ состоит из регистров, арифметико-логического устройства (АЛУ) и цепей управления.

2. Память программ. Здесь хранятся коды команд, последовательность которых формирует программу для микроконтроллера.

3. Оперативная память данных. Здесь хранятся переменные программ. У большинства микроконтроллеров здесь расположен также стек.

4. Тактовый генератор. Этот генератор определяет скорость работы микроконтроллера.

5. Цепь сброса. Эта цепь служит для правильного запуска микроконтроллера.

6. Последовательный порт — очень полезный элемент микроконтроллера. Он позволяет обмениваться данными с внешними устройствами при малом количестве проводов.

7. Цифровые линии ввода/вывода. По сравнению с последовательным портом с помощью этих линий возможно управлять одновременно несколькими линиями (или проверять несколько линий).

8. Таймер. Используется для отсчета временных интервалов.

9. Сторожевой таймер. Это специальный таймер, предназначенный для предотвращения сбоев программы. Он работает следующим образом: после запуска он начинает отсчет заданного временного интервала. Если программа не перезапустит его до истечения этого интервала времени, сторожевой таймер перезапустит микроконтрол-

лер. Таким образом, программа должна давать сторожевому таймеру сигнал — все в порядке. Если она этого не сделала, значит, по какой-либо причине произошел сбой.

## **1.2. Разработка конструкций на микроконтроллерах**

Итак, теперь, когда мы имеем первоначальное представление о микроконтроллерах и о том, что можно сделать с их помощью, пришло время поговорить о создании конструкций на микроконтроллерах.

Один из вариантов последовательности действий при разработке конструкций на микроконтроллерах приведен ниже.

1. Во-первых, очень важно точно определить технические требования к конструкции, причем делать это следует письменно — обычно в процессе записи выявляется много нюансов, не сразу заметных при обдумывании конструкций.

2. Составить подробное описание конструкции так называемого верхнего уровня — на этом этапе еще неизвестно ни типа микроконтроллера, ни типа использованных микросхем и схемных решений, поэтому структурная схема представляет собой набор прямоугольников, подписанных наименованием узла, например АЦП. Составляется обобщенная блок-схема, описывающая работу программы. Если нужно — временные диаграммы.

3. Определиться с выбором аппаратных узлов (микросхем и т. д.) для схемы.

4. Выбрать тип микроконтроллера.

5. Убедиться в том, что микроконтроллер подходит для реализации схемы. Следует учитывать быстродействие микроконтроллера, наличие нужной периферии, число линий ввода/вывода, потребляемую мощность и другие, существенные для конкретной конструкции параметры. Не следует «бить из пушки по воробьям» — использовать более мощный микроконтроллер для простейшей задачи, с которой может справиться и более простой (и более дешевый). С другой стороны, не следует увлекаться слабыми микроконтроллерами, усложняя схему, добавляя схему увеличения числа выводов, так как

достаточно часто (но не всегда) экономия, полученная за счет применения более дешевого микроконтроллера, полностью теряется из-за увеличившейся стоимости печатной платы (ведь ее размеры увеличились), стоимости дополнительных элементов и т. д.

6. Теперь следует определиться, какие инструменты (программы) будут использоваться для разработки программы для микроконтроллера. Это может быть транслятор языка ассемблер или компилятор языка высокого уровня, чаще всего С.

7. После того как стали известны используемые узлы микроконтроллера и внешние схемы, подключаемые к нему, можно приступать к написанию и отладке программы. Целесообразно разделить конструкцию на функциональные узлы и отлаживать их следующим образом: изготовить часть схемы, реализующей собой один из узлов, написать фрагмент программы, управляющей этим узлом, и отладить его. После этого аналогично работать со следующим узлом, и так до тех пор, пока все части схемы не будут отлажены отдельно друг от друга. При этом можно пользоваться уже отлаженными узлами для облегчения проверки правильности работы следующих, только отлаживаемых узлов. Например, для простого калькулятора можно выделить следующие узлы: индикатор, клавиатура. Отлаживая индикатор, можно написать программу, выводящую на индикатор какое-либо число. Затем, отлаживая клавиатуру, можно использовать индикатор для вывода, например, номера нажатой клавиши. И только убедившись, что оба узла работают верно, следует переходить к реализации программы собственно калькулятора.

8. Теперь следует объединить все части схемы воедино и отладить их работу совместно. Если в процессе объединения обнаружится, что какой-либо из узлов реализован не совсем удачно, следует вернуться в предыдущий пункт.

9. Очень важно в процессе составления схемы конструкции и написания программы для нее как можно более подробно документировать все изменения в схеме или программе. Это очень важно не только для записи проделанной работы (а в хорошо документированной схеме и программе можно быстро разобраться при необходимости ее повторного использования), но и для облегчения дальнейшего усовершенствования или обслуживания собранной системы.

10. Заключительный этап относится к случаям, когда спроектированная конструкция будет производиться, — подготовка чертежей принципиальной электрической схемы, печатной платы, спецификаций в соответствии со стандартами, принятыми в месте, где будет осуществляться производство конструкции.

### 1.3. Микроконтроллеры семейства AVR

AVR — это новое семейство 8-разрядных RISC-микроконтроллеров фирмы Atmel. Эти микроконтроллеры позволяют решать множество задач встроенных систем. Они отличаются от других распространенных в настоящее время микроконтроллеров большей скоростью работы, большей универсальностью. Быстродействие данных микроконтроллеров позволяет в ряде случаев применять их в устройствах, для реализации которых ранее можно было применять только 16-разрядные микроконтроллеры, что позволяет ощутимо удешевить готовую систему. Кроме того, микроконтроллеры AVR очень легко программируются — простейший программатор можно изготовить самостоятельно буквально в течение 30 минут!

По заявлению фирмы-производителя микроконтроллеров ([www.atmel.com](http://www.atmel.com)) микроконтроллеры семейства AVR можно перепрограммировать до 1000 раз, причем непосредственно в собранной схеме.

Все это делает эти микроконтроллеры очень привлекательными для создания новых разработок.

#### Почему именно AVR?

Микроконтроллеры AVR разработаны фирмой Atmel и обладают следующими основными характеристиками:

- очень быстрая гарвардская RISC-архитектура загрузки и выполнения большинства инструкций в течение **ОДНОГО** цикла тактового генератора. При этом достигается скорость работы примерно 1 MIPS на МГц. Частота тактового генератора многих типов микроконтроллеров AVR может достигать 10...16 МГц (10...16 MIPS!) (MIPS — Millions Instructions per Second — миллионов операций в секунду). Отсутствует внутрен-

нее деление частоты, как, например, в микроконтроллерах PIC. Таким образом, если использован кварцевый резонатор с частотой 16 МГц, микроконтроллер будет работать с быстродействием почти 16 MIPS;

- программы содержатся в электрически перепрограммируемой постоянной памяти программ FLASH ROM. Эта память может быть перепрограммирована до 1000 раз. Это облегчает настройку и отладку систем. Кроме того, возможность внутрисхемного программирования позволяет не вынимать микроконтроллер из целевой схемы в процессе программирования, что значительно ускоряет процесс разработки систем на основе этих микроконтроллеров;
- система команд микроконтроллеров AVR изначально проектировалась с учетом особенностей языка программирования высокого уровня C, что в результате позволяет получать после компиляции программ на C гораздо более эффективный код, чем для других микроконтроллеров. А это уже выигрыш и в размере полученного кода (в объеме памяти на кристалле), и в скорости работы микроконтроллера;
- микроконтроллеры AVR имеют 32 регистра, все из которых напрямую работают с АЛУ. Это значительно уменьшает размер программ. В других микроконтроллерах, как правило, для осуществления, например, сложения один из операндов обязательно должен находиться в специальном регистре — аккумуляторе. Таким образом, необходимо сначала его туда занести, затем после выполнения операции результат из аккумулятора нужно переписать в регистр для хранения результата. Итого получается уже три команды. В микроконтроллерах AVR то же самое займет всего одну команду;
- очень небольшое потребление энергии и наличие нескольких режимов работы с пониженным потреблением энергии делает эти микроконтроллеры идеальными для применения в конструкциях, питающихся от батареек;
- наличие дешевых и простых в использовании программных средств. Многие полноценные программы доступны в свободно распространяемом варианте, как, например, отладчик AVR Studio, ассемблер Wavtasm, множество версий программаторов и

даже компилятор языка C — avr gcc. Некоторые из этих программ имеются на компакт-диске, прилагаемом к книге;

- узлы PWM (широтно-импульсная модуляция), таймеры/счетчики, аналоговый компаратор и последовательный порт UART встроены в микроконтроллеры и могут управляться с помощью прерываний, что значительно упрощает работу с ними;
- имеются относительные команды переходов и ветвлений, что позволяет получать перемещаемый код;
- отсутствует необходимость переключать страницы памяти (в отличие, например, от микроконтроллеров PIC);
- все микроконтроллеры AVR имеют электрически перепрограммируемую постоянную память данных EEPROM, которая может быть перепрограммирована более 100 000 раз!

Имеется три подсемейства микроконтроллеров AVR:

1. tiny AVR — недорогие миниатюрные микроконтроллеры в

8-выводном исполнении;

2. Classic AVR — основная линия микроконтроллеров с производительностью отдельных модификаций до 16 MIPS, FLASH-памятью программ 2...8 Кб, памятью данных EEPROM 64...512 байт, оперативной памятью данных SRAM 128...512 байт;

3. mega AVR с производительностью 4...16 MIPS для сложных приложений, требующих большого объема памяти, FLASH-памятью программ до 128 Кб, памятью данных EEPROM 64...512 байт, оперативной памятью данных SRAM 2...4 Кб, встроенным 10-разрядным 8-канальным АЦП, аппаратным умножителем  $8 \times 8$ .

Интересной особенностью семейства микроконтроллеров AVR является то, что система команд всего семейства совместима при переносе программы со слабого на более мощный микроконтроллер.

На рис. 1.2 и 1.3 приведены таблицы с характеристиками имеющихся в настоящее время и анонсированных к выпуску микроконтроллеров семейства AVR.

По мнению автора, AT90S2313 — наиболее удобный микроконтроллер для первоначального знакомства. Он имеет почти все базовые периферийные устройства, присутствующие в микроконтроллерах серии AVR и отличается от более мощных только меньшим числом линий ввода/вывода, размером памяти программ, данных, числом таймеров (тем не менее он имеет два таймера: 8- и 16-разрядный).

	FLASH-память программ	Память данных EEPROM	ОЗУ данных SRAM	Кол-во команд	Число линий ввода-вывода	Кол-во прерываний	Кол-во внешних прерываний	SPI-интерфейс	Послед. интерфейс UART	8-разрядный таймер	8-разрядный таймер	ШИМ	Сторожевой таймер	Аналоговый компаратор	10-разр. АЦП, число каналов	Встроенный генератор	Детектор сниж-я напряж-я	Внутрисхем. прог-в	Напряжение питания	Тактовая частота, МГц	Тип корпуса
ATiny11L	1	—	—	90	6	4	1	—	—	1	—	—	Y	Y	—	Y	—	Y	2,7–5,5	0–2	8-Pin DIP
																					8-Pin SOIC
ATiny11	1	—	—	90	6	4	1	—	—	1	—	—	Y	Y	—	Y	—	Y	4,0–5,5	0–6	8-Pin DIP
																					8-Pin SOIC
ATiny12V	1	64	—	90	6	5	1	—	—	1	—	—	Y	Y	—	Y	Y	Y	1,8–5,5	0–1	8-Pin DIP
																					8-Pin SOIC
ATiny12L	1	64	—	90	6	5	1	—	—	1	—	—	Y	Y	—	Y	Y	Y	2,7–5,5	0–2	8-Pin DIP
																					8-Pin SOIC
ATiny12	1	64	—	90	6	5	1	—	—	1	—	—	Y	Y	—	Y	Y	Y	4,0–5,5	0–8	8-Pin DIP
																					8-Pin SOIC
ATiny15L	1	64	—	90	6	8	1	—	—	2	—	1	Y	Y	4	Y	Y	Y	2,7–5,5	1,6	8-Pin DIP
																					8-Pin SOIC
ATiny28V	2	—	—	90	20	5	2	—	—	1	—	—	Y	Y	—	Y	—	Y	1,8–5,5	0–1	8-Pin DIP
																					32-Pin MLF
																					32-Pin TQFP
ATiny28L	2	—	—	90	20	5	2	—	—	1	—	—	Y	Y	—	Y	—	Y	2,7–5,5	0–4	8-Pin DIP
																					32-Pin MLF
																					32-Pin TQFP
AT90S1200	1	64	—	89	15	3	1	—	—	1	—	—	Y	Y	—	Y	—	Y	2,7–6,0	0–12	20-Pin DIP
																					32-Pin SOIC
																					32-Pin SSOP
AT90S2313	2	128	128	120	15	10	2	—	1	1	1	1	Y	Y	—	—	—	Y	2,7–6,0	0–10	8-Pin DIP
																					8-Pin SOIC
AT90LS2323	2	128	128	120	3	2	1	—	—	1	—	—	Y	—	—	—	—	Y	2,7–6,0	0–4	8-Pin DIP
																					8-Pin SOIC
AT90S2323	2	128	128	120	3	2	1	—	—	1	—	—	Y	—	—	—	—	Y	4,0–6,0	0–10	8-Pin DIP
																					8-Pin SOIC
AT90LS2343	2	128	128	120	4	2	1	—	—	1	—	—	Y	—	—	Y	—	Y	2,7–6,0	0–1	8-Pin DIP
																					8-Pin SOIC
AT90LS2343	2	128	128	120	4	2	1	—	—	1	—	—	Y	—	—	Y	—	Y	2,7–6,0	0–4	8-Pin DIP
																					8-Pin SOIC
AT90S2343	2	128	128	120	4	2	1	—	—	1	—	—	Y	—	—	Y	—	Y	4,0–6,0	0–10	8-Pin DIP
																					8-Pin SOIC
AT90LS4433	4	256	128	120	20	14	2	1	1	1	1	1	Y	Y	6	—	Y	Y	2,7–6,0	0–4	28-Pin DIP
																					32-Pin TQFP
AT90S4433	4	256	128	120	20	14	2	1	1	1	1	1	Y	Y	6	—	Y	Y	4,0–6,0	0–8	28-Pin DIP
																					32-Pin TQFP
AT90S8515	8	512	512	120	32	11	2	1	1	1	1	2	Y	Y	—	—	—	Y	2,7–6,0	0–4	40-Pin DIP
																					44-Pin PLCC
																					44-Pin TQFP
AT90S8515	6	512	512	120	32	11	2	1	1	1	1	2	Y	Y	—	—	—	Y	4,0–6,0	0–8	40-Pin DIP
																					44-Pin PLCC
																					44-Pin TQFP

Рис. 1.2. Характеристики микроконтроллеров семейства AVR

		FLASH-память программы	Память данных EEPROM	ОЗУ данных SRAM	Кол-во команд	Число линий ввода-вывода	Кол-во прерываний	Кол-во внешних прерываний	SPI-интерфейс	Послед. интерфейс UART	Интерф. ТМ1, совм. с I <sup>2</sup> C	Аппаратное умножение	8-разрядный таймер	16-разрядный таймер	ШИМ	Сторожевой таймер	Таймер реал. врем. RTC	Аналоговый компаратор	10-разр. АЦП, число каналов	Встроенный генератор	Детектор снм-я напряж-я	Внутрисхем. прогн-е	Самопрограммирование	Напряжение питания	Тактовая частота, МГц	Тип корпуса
AT90LS8535	8	512	512	120	32	15	2	1	1	-	-	2	1	3	Y	Y	Y	8	-	-	Y	-	2,7-6,0	0-4	40-Pin DIP	
																									44-Pin PLCC	
																									44-Pin TQFP	
AT90S8535	8	512	512	120	32	15	2	1	1	-	-	2	1	3	Y	Y	Y	8	-	-	Y	-	4,7-6,0	0-8	40-Pin DIP	
																									44-Pin PLCC	
																									44-Pin TQFP	
ATmega8L	8	512	1K	130	23	16	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y	Y	Y	Y	2,7-5,5	0-8	28-Pin DIP	
																									32-Pin MLF	
																									32-Pin TQFP	
ATmega8	8	512	1K	130	23	16	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y	Y	Y	Y	4,0-5,5	0-16	28-Pin DIP	
																									32-Pin MLF	
																									32-Pin TQFP	
ATmega161L	16	512	1K	130	35	20	3	1	2	-	Y	2	1	4	Y	Y	Y	-	-	Y	Y	Y	2,7-5,5	0-4	40-Pin DIP	
																									44-Pin TQFP	
ATmega161	18	512	1K	130	35	20	3	1	2	-	Y	2	1	4	Y	Y	Y	-	-	Y	Y	Y	4,0-5,5	0-4	40-Pin DIP	
																									44-Pin TQFP	
ATmega169L	16	512	1K	130	32	17	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y	Y	Y	Y	2,7-5,5	0-4	40-Pin DIP	
																									44-Pin TQFP	
ATmega163	16	512	1K	130	32	17	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y	Y	Y	Y	4,0-5,5	0-8	40-Pin DIP	
																									44-Pin TQFP	
ATmega16L	16	512	1K	130	32	17	3	1	1	1	Y	2	1	3	Y	Y	Y	8	Y	Y	Y	Y	2,7-5,5	0-8	40-Pin DIP	
																									44-Pin TQFP	
ATmega16	16	512	1K	130	32	17	3	1	1	1	Y	2	1	3	Y	Y	Y	8	Y	Y	Y	Y	4,0-5,5	0-16	40-Pin DIP	
																									44-Pin TQFP	
ATmega323L	32	1K	2K	130	32	19	3	1	1	1	Y	2	1	4	Y	Y	Y	8	Y	Y	Y	Y	2,7-5,5	0-4	40-Pin DIP	
																									44-Pin TQFP	
ATmega323	32	1K	2K	130	32	19	3	1	1	1	Y	2	1	4	Y	Y	Y	8	Y	Y	Y	Y	4,0-5,5	0-8	40-Pin DIP	
																									44-Pin TQFP	
ATmega103L	128	4K	4K	121	48	16	8	1	1	-	-	2	1	4	Y	Y	Y	8	-	-	Y	-	2,7-3,8	0-4	64-Pin TQFP	
ATmega103L	128	4K	4K	121	48	16	8	1	1	-	-	2	1	4	Y	Y	Y	8	-	-	Y	-	4,0-5,5	0-4	64-Pin TQFP	
ATmega128L	128	4K	4K	133	48	27	8	1	2	1	Y	2	2	6+2	Y	Y	Y	8	Y	Y	Y	Y	2,7-5,5	0-8	64-Pin TQFP	
ATmega128	128	4K	4K	133	48	27	8	1	2	1	Y	2	2	6+2	Y	Y	Y	8	Y	Y	Y	Y	4,0-5,5	0-16	64-Pin TQFP	

Рис. 1.3. Характеристики микроконтроллеров семейства AVR (продолжение)

В результате вполне справедливо будет сказать, что, изучив в достаточной степени микросхему AT90S2313, читатели легко смогут использовать более мощные микроконтроллеры. Микроконтроллер AT90S1200 не подходит для этой цели по причине отсутствия у него оперативной памяти данных — SRAM, что значительно отличает его возможности от остальных микроконтроллеров семейства.

По этой причине во второй главе приводится достаточно подробное описание микроконтроллера AT90S2313. Совсем нет необходимости изучать эту главу страница за страницей, хотя прочесть хотя бы обзорно ее стоит. Те, кто хочет поскорее перейти к практическому изготовлению каких-либо схем, могут просмотреть главу 4, где описаны программы для работы с микроконтроллерами семейства AVR, и переходить к главам 5 и 6, содержащим описания фрагментов схем и завершенных схем. В дальнейшем можно обращаться к главе 2 как к справочному пособию при возникновении вопросов и разборе работы той или иной схемы.

## **Глава 2.**

# **Описание микроконтроллера AT90S2313 фирмы Atmel**

AT90S2313 — современный 8-битовый КМОП-микроконтроллер. AT90S2313 имеет производительность около 1 MIPS на мегагерц за счет того, что почти все команды он выполняет за один период тактового генератора.

Микроконтроллеры семейства AVR построены на основе расширенной RISC-архитектуры, объединяющей развитый набор команд и 32 регистра общего назначения. Все 32 регистра непосредственно подключены к арифметико-логическому устройству (АЛУ), что дает доступ к любым двум регистрам в течение одного машинного цикла. Подобная архитектура обеспечивает почти десятикратный выигрыш в производительности по сравнению с традиционными микроконтроллерами, например, серии 8051.

Микроконтроллер AT90S2313 имеет следующие характеристики: 2 Кб загружаемой флэш-памяти; 128 байтов EEPROM; 15 линий ввода/вывода общего назначения; 32 рабочих регистра; два таймера/счетчика, один 8-разрядный, другой 16-разрядный; внешние и внутренние прерывания; встроенный последовательный порт; программируемый сторожевой таймер со встроенным генератором; последовательный порт SPI для загрузки программ; два выбираемых программно режима низкого энергопотребления.

Флэш-память на кристалле может быть перепрограммирована прямо в системе через последовательный интерфейс SPI.

## **2.1. Описание выводов**

VCC — вывод источника питания.

GND — общий провод («земля»).

PORT B (PB7...PB0) — порт B является 8-битовым двунаправленным параллельным портом ввода/вывода с встроенными подтягивающими резисторами. У выводов порта предусмотрены внутренние

подтягивающие резисторы (их можно включать или выключать для каждого бита отдельно). Выводы PB0 и PB1 также являются положительным (AIN0) и отрицательным (AIN1) входами встроенного аналогового компаратора. Выходные буферы порта В могут поглощать ток до 20 мА и непосредственно управлять светодиодными индикаторами. Это означает, что микроконтроллер способен управлять нагрузкой до 20 мА при состоянии логического 0 на выходе порта. Таким образом, для управления светодиодом его следует подсоединить одним выводом к выводу порта микроконтроллера, а другим — к напряжению питания +Vcc. Соответственно светиться светодиод (а значит, и потреблять ток) будет при значении 0 на соответствующей линии порта. Если выводы PB0...PB7 используются как входы и извне устанавливаются в низкое состояние, они являются источниками тока, если включены внутренние подтягивающие резисторы. Кроме того, порт В обслуживает некоторые специальные функции, которые будут описаны ниже.

PORT D (PD6...PD0) — порт D является 7-битовым двунаправленным параллельным портом ввода/вывода с встроенными подтягивающими резисторами. Выходные буферы порта D также могут поглощать ток до 20 мА. Как входы, установленные в низкое состояние, выводы порта D являются источниками тока, если задействованы подтягивающие резисторы. Кроме того, порт D обслуживает некоторые специальные функции, которые будут описаны ниже.

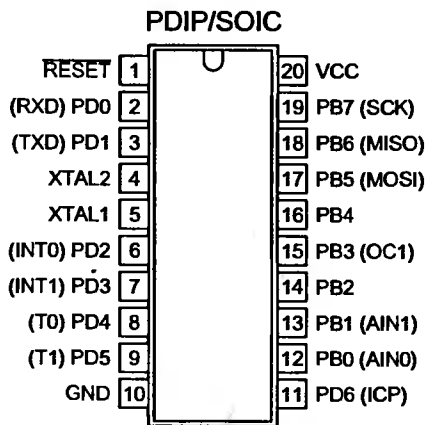


Рис. 2.1. Выводы микроконтроллера AT90S2313

RESET — вход сброса. Удержание на входе низкого уровня в течение двух машинных циклов (если работает тактовый генератор), перезапускает микроконтроллер.

XTAL1 — вход инвертирующего усилителя генератора и вход внешнего тактового сигнала.

XTAL2 — выход инвертирующего усилителя генератора.

## Кварцевый генератор

Выводы XTAL1 и XTAL2 являются входом и выходом инвертирующего усилителя, на котором можно собрать генератор тактовых импульсов. Можно использовать как кварцевые, так и керамические резонаторы. Если нужно использовать внешний тактовый сигнал, он подается на вывод XTAL1, а вывод XTAL2 при этом остается неподключенным.

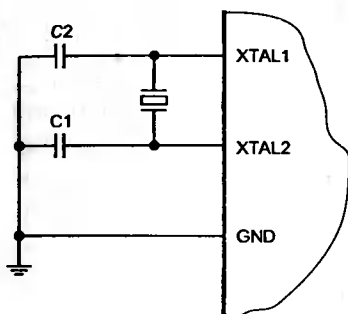


Рис. 2.2. Подключение кварцевого резонатора к микроконтроллеру

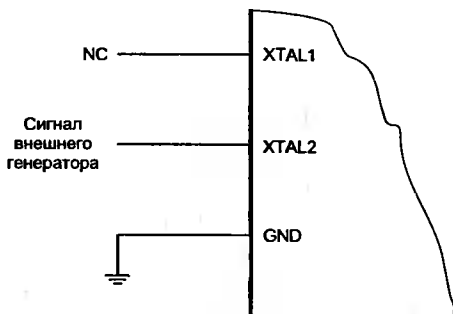


Рис. 2.3. Подключение внешнего источника тактовых импульсов

## 2.2. Обзор архитектуры AT90S2313

### Файл регистров общего назначения

Регистровый файл микроконтроллера содержит 32 8-разрядных регистра общего назначения, доступ к которым осуществляется за один машинный цикл. Благодаря этому микроконтроллер может выполнить большинство команд за один цикл тактовой частоты.

АЛУ поддерживает арифметические и логические операции с регистрами, с константами и регистрами.

Кроме регистровых операций, для работы с регистровым файлом могут использоваться доступные режимы адресации, так как регистровый файл занимает адреса \$00-\$1F в области данных, обращаться к ним можно и как к ячейкам памяти.

Пространство ввода/вывода состоит из 64 адресов для периферийных функций процессора, таких, как управляющие регистры, таймеры/счетчики и др. Доступ к пространству ввода/вывода может осуществляться непосредственно как к ячейкам памяти, расположенным после регистрового файла (\$20—\$5F).

Большинство команд, использующих регистры, могут использовать любые регистры общего назначения. Исключение составляют пять команд, оперирующих с константами: SBCI, SUBI, CPI, ANDI, ORI и команда LDI, загружающая регистр константой. Эти команды работают только со второй половиной регистрового файла — R16...R31.

Каждому регистру присвоен адрес в пространстве данных, они отображаются на первые 32 ячейки ОЗУ. Хотя регистровый файл физически размещен вне ОЗУ, подобная организация памяти дает гибкий доступ к регистрам.

Шесть из 32 регистров — R26...R31 — можно использовать как три 16-разрядных адресных указателя в адресном пространстве данных. Один из трех адресных указателей (регистр Z) можно использовать для адресации таблиц в памяти программ. Эти регистры обозначаются как X, Y, Z и определены следующим образом:

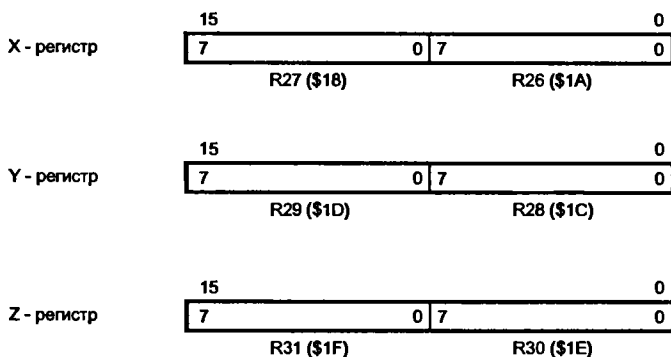


Рис. 2.4. Регистры X, Y, Z

При различных режимах адресации эти регистры могут использоваться как фиксированный адрес, для адресации с автоинкрементом или с автодекрементом.

При разработке микроконтроллеров семейства AVR была использована так называемая гарвардская архитектура. Смысл ее состоит в том, что память программ и данных программы располагается в разных областях памяти. На рис. 2.5 изображена структура памяти микроконтроллеров AVR.

Доступ к памяти программ осуществляется следующим образом: во время выполнения одной команды следующая команда выбирается из памяти программ. Это дает возможность выполнять по одной команде за каждый машинный цикл.

При помощи команд относительных переходов и вызова подпрограмм осуществляется доступ ко всему адресному пространству. Большая часть команд микроконтроллеров AVR имеет размер 16-разрядов одно слово. Каждый адрес в памяти программ содержит одну 16- или 32-разрядную команду.

При обработке прерываний и вызове подпрограмм адрес возврата запоминается в стеке. Стек размещается в оперативной памяти данных общего назначения (SRAM), его размер ограничен только размером доступной памяти SRAM и ее использованием в программе. Все программы пользователя должны инициализировать указатель стека (SP) сразу после запуска микроконтроллера, до того как вызываются подпрограммы и разрешаются прерывания. Исключение составляют

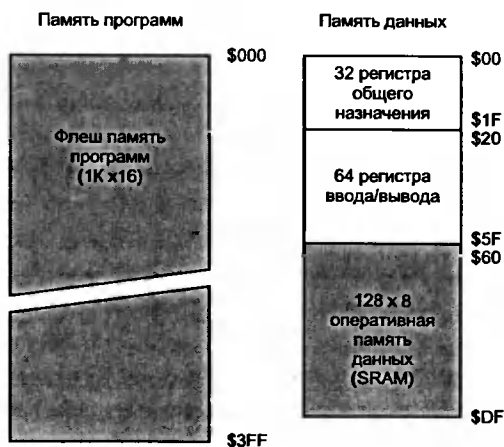


Рис. 2.5. Структура памяти микроконтроллеров AVR

микроконтроллеры, не имеющие оперативной памяти данных (SRAM), например, AT90S1200. У этих микроконтроллеров реализован аппаратный стек глубиной 3. Это обязательно следует учитывать при написании для них программ.

Все пространство памяти AVR является линейным и непрерывным.

Модуль прерываний имеет собственный управляющий регистр в пространстве ввода/вывода, и флаг глобального разрешения прерываний в регистре состояния. Каждому прерыванию назначен свой вектор в начальной области памяти программ. Различные прерывания имеют приоритет в соответствии с расположением их векторов. По младшим адресам расположены векторы с большим приоритетом.

**Режимы адресации**

**Прямая регистровая адресация с одним регистром Rd**

Данные, над которыми осуществляется операция (или используются при выполнении операции), находятся в регистре d (Rd).

**Прямая регистровая адресация с двумя регистрами — Rd и Rr**

Данные, над которыми осуществляется операция, находятся в регистрах r (Rr) и d (Rd). Результат операции сохраняется в регистре d (Rd).

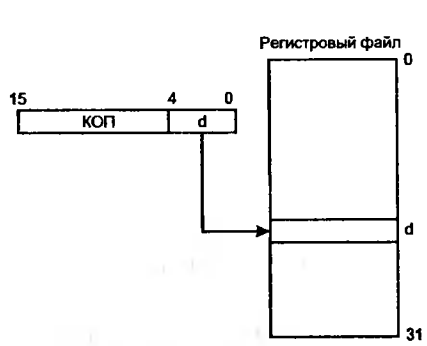


Рис. 2.6. Прямая регистровая адресация с одним регистром

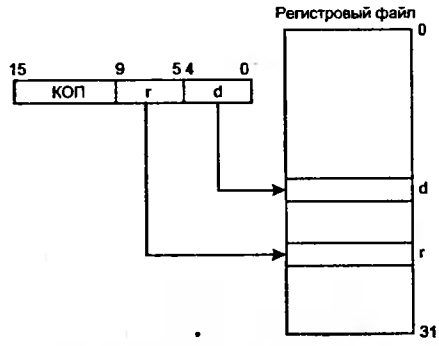
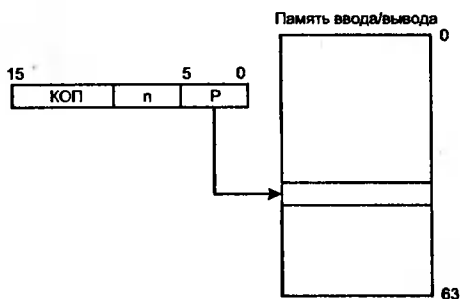
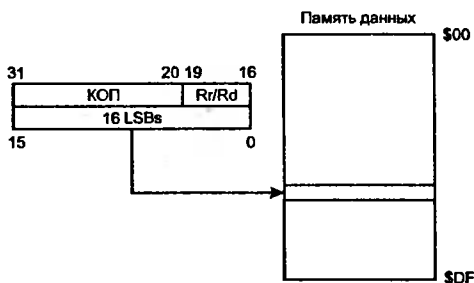


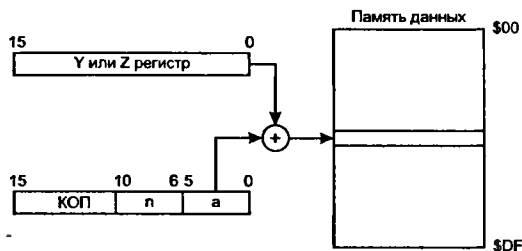
Рис. 2.7. Прямая регистровая адресация с двумя регистрами

**Прямая адресация к области ввода/вывода****Рис. 2.8.** Прямая адресация к области ввода/вывода

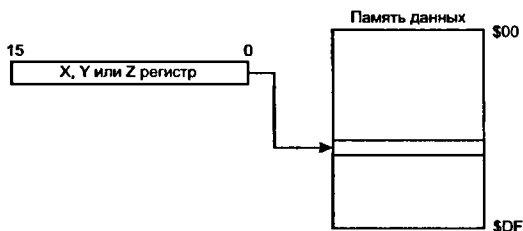
$n$  — адрес регистра, используемого в операции, находится непосредственно в коде команды, в битах 0...5.

**Прямая адресация к памяти данных****Рис. 2.9.** Прямая адресация к памяти данных

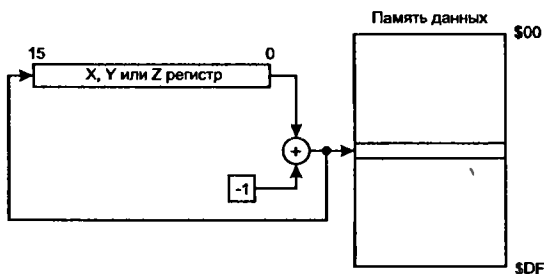
16-разрядный адрес ячейки памяти данных находится в коде команды, состоящей из двух слов.  $Rr/Rd$  определяет регистр, используемый при работе с памятью данных (т. е. регистр, куда записываются результаты операции либо откуда они берутся для выполнения операции).

**Косвенная адресация к памяти данных со смещением****Рис. 2.10.** Косвенная адресация к памяти данных со смещением

Адрес операнда определяется как сумма содержимого Z или Y регистра и бит 0...5 кода команды.

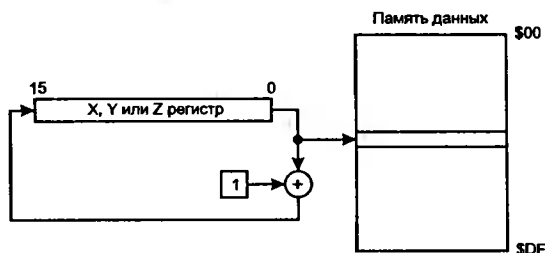
**Косвенная адресация к памяти данных****Рис. 2.11.** Косвенная адресация к памяти данных

Адрес операнда находится в X-, Y- или Z-регистре.

**Косвенная адресация к памяти данных с предварительным декрементом****Рис. 2.12.** Косвенная адресация к памяти данных с предварительным декрементом

Адрес операнда находится в X-, Y- или Z-регистре. Однако перед выполнением операции соответствующий индексный регистр X-, Y- или Z уменьшается на единицу.

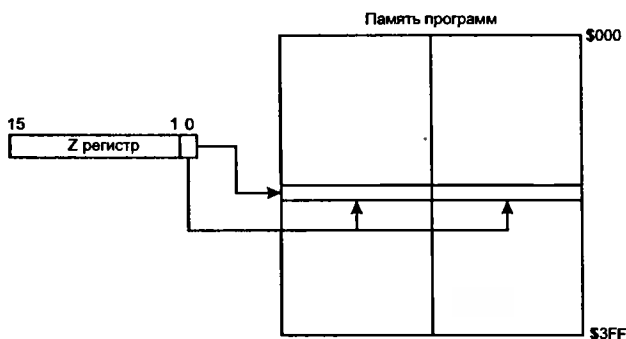
### **Косвенная адресация к памяти данных с постинкрементом**



**Рис. 2.13.** Косвенная адресация к памяти данных с постинкрементом

Адрес операнда находится в X-, Y- или Z-регистре. После выполнения операции соответствующий индексный регистр X-, Y- или Z увеличивается на единицу.

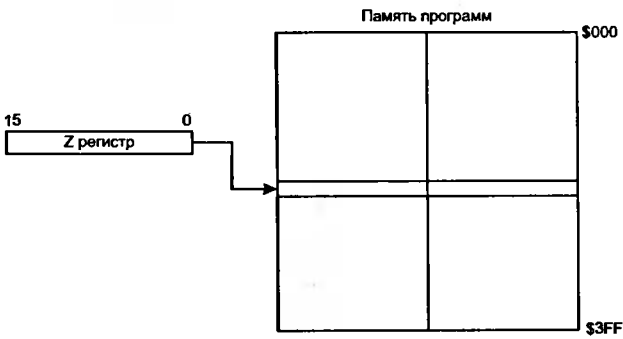
### **Адресация к константам в памяти программ**



**Рис. 2.14.** Адресация к константам в памяти программ

Адрес константы — байта содержится в регистре Z. 15 старших битов определяют адрес слова, а младший (0) бит — младший или старший байт константы в памяти программ. Если в младшем бите 0 — выбран младший байт, 1 — старший байт.

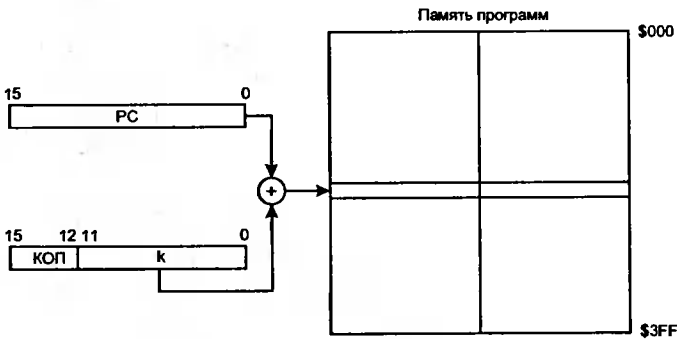
**Косвенная адресация памяти программ**



**Рис. 2.15.** Косвенная адресация памяти программ

После операций **IJMP** или **ICALL** выполнение программы продолжается с адреса, записанного в **Z-регистре** (т. е. в **PC** счетчик команд микроконтроллера записывается содержимое **Z-регистра**).

**Относительная адресация памяти программ**



**Рис. 2.16.** Относительная адресация памяти программ

После операций **RJMP** или **RCALL** выполнение программы продолжается с адреса **PC+k+1**. Относительный адрес **k** может составлять от **-2048** до **2047**.

## **Арифметико-логическое устройство**

Арифметико-логическое устройство (АЛУ) микроконтроллера непосредственно подключено к 32 регистрам общего назначения. За один машинный цикл АЛУ производит операции между регистрами регистрового файла. АЛУ может выполнять арифметические, логические и битовые операции.

## **Память программ**

AT90S2313 содержит 2 Кб флэш-памяти для хранения программ. Флэш-память организована как  $1K \times 16$ . В фирменном описании микроконтроллера утверждается, что флэш-память можно перепрограммировать до 1000 раз.

Программный счетчик имеет ширину 10 битов и позволяет адресовать 1024 слов памяти программ.

Способы занесения информации (т. е. программ) во флэш-память микроконтроллера будут рассмотрены дальше.

## **EEPROM память данных**

AT90S2313 содержит 128 байтов электрически стираемой энергонезависимой памяти (EEPROM). EEPROM организована как отдельная область данных, каждый байт которой может быть прочитан и при необходимости перезаписан. EEPROM выдерживает не менее 100 000 циклов записи/стирания. К этой памяти может обращаться программа, считывая или записывая какие-либо данные. Кроме того, данные в эту память можно занести с помощью специального устройства — программатора, на этапе изготовления и программирования конструкции. Ее удобно использовать для хранения каких-либо констант.

## **Оперативная память данных**

На рис. 2.17 показана организация памяти данных в AT90S2313.

224 ячейки памяти включают в себя регистровый файл, память ввода/вывода и оперативную память данных.

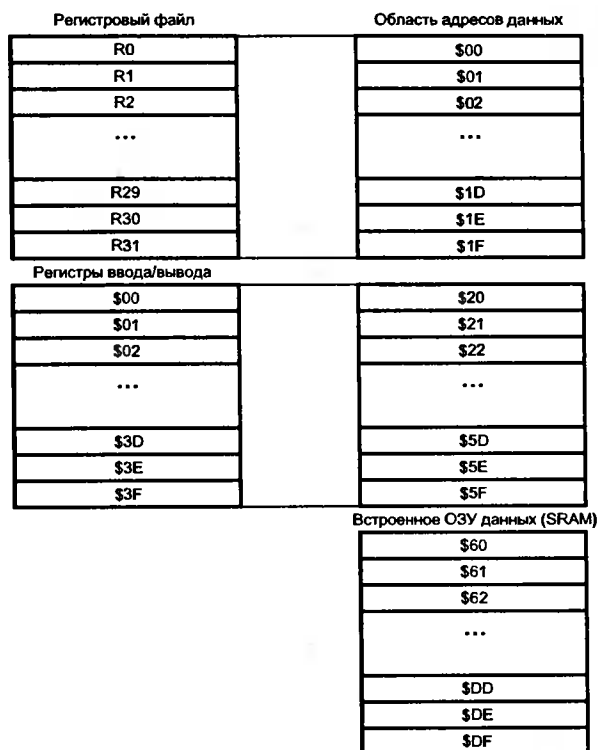


Рис. 2.17. Организация памяти данных в микроконтроллере AT90S2313

Первые 96 адресов используются для регистрового файла и памяти ввода/вывода, следующие 128 — для ОЗУ данных.

При обращении к памяти используются пять различных режимов адресации: прямой, непосредственный со смещением, непосредственный, непосредственный с предварительным декрементом и непосредственный с постинкрементом. Регистры R26...R31 регистрового файла используются как указатели для непосредственной адресации.

Прямая адресация имеет доступ ко всей памяти данных. Непосредственная адресация со смещением используется для доступа к 63 ячейкам, базовый адрес которых задается содержимым регистров Y или Z.

Для непосредственной адресации с инкрементом и декрементом адреса используются адресные регистры X, Y и Z.

При помощи любого из этих режимов можно осуществлять доступ ко всем 32 регистрам общего назначения, 64 регистрам ввода/вывода и 128 ячейкам ОЗУ.

### Время выполнения команд

ЦПУ процессора AVR управляется тактовой частотой, генерируемой внешним резонатором. Внутреннее деление частоты генератора в микроконтроллерах семейства AVR не используется.

В микроконтроллере процесс выполнения команд организован так, что при выборе команды из памяти программ происходит выполнение предыдущей команды. Это позволяет достичь быстродействия 1 MIPS на МГц.

Таблица 2.1. Пространство ввода/вывода AT90S2313

Адрес	Название	Назначение
\$3F (\$5F)	SREG	Регистр состояния
\$3D (\$5D)	SPL	Младший байт указателя стека
\$3B (\$5B)	GIMSK	Общий регистр маски прерываний
\$3A (\$5A)	GIFR	Общий регистр флагов прерываний
\$39 (\$59)	TIMSK	Регистр маски прерываний от таймеров/счетчиков
\$38 (\$58)	TIFR	Регистр флагов прерываний от таймеров/счетчиков
\$35 (\$55)	MCUCR	Общий регистр управления микроконтроллера
\$33 (\$53)	TCCR0	Регистр управления таймером/счетчиком 0
\$32 (\$52)	TCNT0	Таймер/счетчик 0 (8 битов)
\$2F (\$4F)	TCCR1A	Регистр А управления таймером/счетчиком 1
\$2E (\$4E)	TCCR1B	Регистр В управления таймером/счетчиком 1
\$2D (\$4D)	TCNT1H	Старший байт таймера/счетчика 1
\$2C (\$4C)	TCNT1L	Младший байт таймера/счетчика 1
\$2B (\$4B)	OCR1AH	Выход регистра совпадения 1, старший байт
\$2A (\$4A)	OCR1AL	Выход регистра совпадения 1, младший байт

Продолжение табл. 2.1

Адрес	Название	Назначение
\$25 (\$45)	ICR1H	Регистр захвата таймера/счетчика 1, старший байт
\$24 (\$44)	ICR1L	Регистр захвата таймера/счетчика 1, младший байт
\$21 (\$41)	WDTCSR	Регистр управления сторожевым таймером
\$1E (\$3E)	EEAR	Регистр адреса EEPROM
\$1D (\$3D)	EEDR	Регистр данных EEPROM
\$1C (\$3C)	EECR	Регистр управления EEPROM
\$18 (\$38)	PORTB	Регистр данных порта B
\$17 (\$37)	DDRB	Регистр направления данных порта B
\$16 (\$36)	PINB	Входные линии порта B
\$12 (\$32)	PORTD	Регистр данных порта D
\$11 (\$31)	DDRD	Регистр направления данных порта D
\$10 (\$30)	PIND	Входные линии порта D
\$0C (\$2C)	UDR	Регистр данных последовательного порта UART
\$0B (\$2B)	USR	Регистр состояния последовательного порта UART
\$0A (\$2A)	UCR	Регистр управления последовательного порта UART
\$09 (\$29)	UBRR	Регистр задания скорости последовательного порта UART
\$08 (\$28)	ACSR	Регистр управления и состояния аналогового компаратора
Примечание. Зарезервированные и неиспользуемые в AT90S2313 ячейки не показаны.		

Все устройства ввода/вывода и периферийные устройства AT90S2313 располагаются в пространстве ввода/вывода. Различные ячейки этого пространства доступны через команды IN и OUT, пересылающие данные между одним из 32 регистров общего назначения и пространством ввода/вывода. К регистрам \$00...\$1F можно осуществлять побитовый доступ командами SBI и CBI. Значение отдельного бита этих регистров можно проверить командами SBIC и SBIS. Дополнительную информацию по этому вопросу можно найти в описании системы команд.

При использовании специальных команд IN, OUT, SBIS и SBIC должны использоваться адреса \$00...\$3F. При доступе к регистру ввода/вывода как к ячейке ОЗУ к его адресу необходимо добавить \$20. В приведенной выше таблице адреса регистров в памяти данных приведены в скобках.

### Регистр состояния — SREG

Бит	7	6	5	4	3	2	1	0	
\$3F (\$5F)	I	T	H	S	V	N	Z	C	SREG
Чт./зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.18. Регистр состояния

Регистр состояния расположен по адресу \$3F (\$5F) пространства ввода/вывода и определен следующим образом:

**Бит 7 — I:** общее разрешение прерываний. Для разрешения прерываний этот бит должен быть установлен в единицу. Управление отдельными прерываниями производится регистрами маски прерываний — GIMSK и TIMSK. Если бит I = 0, независимо от состояния GIMSK/TIMSK прерывания запрещены. Бит I обнуляется аппаратно после входа в прерывание и восстанавливается в состояние 1 командой возврата из подпрограммы обработки прерываний RETI, для разрешения обработки последующих прерываний.

**Бит 6 — T:** хранение копируемого бита. Бит из регистра регистравого файла может быть скопирован в T командой BST, бит T может быть скопирован в бит регистрового файла командой BLD.

**Бит 5 — H:** флаг половинного переноса. Этот флаг индицирует перенос из младшей половины байта при некоторых арифметических операциях. Подробнее об этом можно прочитать в описании системы команд.

**Бит 4 — S:** бит знака,  $S = N \text{ XOR } V$ . Бит S равен исключающему ИЛИ между флагами N (отрицательный результат) и V (переполнение дополнения до двух). Если после операций сложения или вычитания чисел со знаком флаг переполнения V будет установлен в 1, то результатом будет 9-разрядное число, причем старшим (т. е. знаковым) разрядом числа будет флаг S, а 8 остальных битов результата

будут храниться в 8-разрядном регистре-приемнике выполнявшейся операции.

**Бит 3 — V:** флаг переполнения до двух. Этот флаг надо проверять после операций сложения или вычитания чисел в дополнительном коде. Он устанавливается в том случае, если результат операции выходит за пределы диапазона от -128 до +127. Это — допустимый диапазон чисел со знаком, которые можно сохранить в 8 разрядах. Соответственно, бит V установится в 1, если при сложении двух положительных чисел получится число, большее 127, или при сложении двух отрицательных чисел — меньше -128.

**Бит 2 — N:** флаг отрицательного результата. Этот флаг устанавливается в 1, если старший (знаковый) разряд (бит 7) равен 1, что означает, что результат отрицателен. Следует, однако, отметить, что флаг может быть также установлен и в результате логических операций.

**Бит 1 — Z:** флаг нулевого результата. Этот флаг индицирует нулевой результат различных арифметических и логических операций. Подробнее об этом можно прочитать в описании системы команд.

**Бит 0 — C:** флаг переноса. Этот флаг индицирует перенос в арифметических и логических операциях. Подробнее об этом можно прочитать в описании системы команд.

### Указатель стека SP

Этот 8-разрядный регистр с адресом \$3D (\$5D) хранит указатель стека процессора AT90S2313. Восьми разрядов достаточно для адресации ОЗУ в пределах \$60 — \$DF.

Бит	7	6	5	4	3	2	1	0	
\$3D (\$5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
Чт./зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.19. Указатель стека

Указатель стека указывает на область памяти, в которой сохраняется адрес возврата из подпрограмм и прерываний. Также с помощью стека подпрограммы могут получать или передавать данные.

Область стека в ОЗУ должна быть задана до того, как произойдет любой вызов подпрограммы или будут разрешены прерывания. Указатель стека уменьшается на 1 при записи данных в стек командой PUSH, и уменьшается на 2 при вызове подпрограммы командой CALL или обработке прерывания. Указатель стека увеличивается на 1 при выборе данных из стека командой POP и увеличивается на 2 при выполнении команд возврата из подпрограммы или обработчика прерывания (RET или RETI).

### 2.3. Перезапуск микроконтроллера (сброс) и обработка прерываний

В AT90S2313 предусмотрены 10 источников прерываний. Эти прерывания и сброс имеют различные векторы в области памяти программ. Каждому из прерываний присвоен отдельный бит, разрешающий данное прерывание при установке бита в 1 (если, конечно, прерывания вообще разрешены, т. е. бит I = 1).

Самые младшие адреса памяти программ определены как векторы прерываний. Полный список векторов прерываний приведен в табл. 2.2. Этот список определяет и приоритет различных прерываний. Меньшие адреса соответствуют более высокому уровню приоритета. Самый высокий уровень у сброса, следующий приоритет у INT0 — внешнего запроса прерывания 0 и т. д.

Ниже приведен типичный фрагмент программы обработки сброса и векторов прерываний:

Адрес	Команда	; Комментарий
\$000	rjmp RESET	; Обработка сброса
\$001	rjmp EXT_INT0	; Обработка IRQ0
\$002	rjmp EXT_INT1	; Обработка IRQ1
\$003	rjmp TIM_CAPT1	; Обработка захвата таймера 1
\$004	rjmp TIM_COMP1	; Обработка совпадения таймера 1
\$005	rjmp TIM_OVF1	; Обработка переполнения таймера 1
\$006	rjmp TIM_OVF0	; Обработка переполнения таймера 0
\$007	rjmp UART_RXC	; Обработка приема байта
\$008	rjmp UART_DRE	; Обработка освобождения UDR
\$009	rjmp UART_TXC	; Обработка передачи байта
\$00a	rjmp ANA_COMP	; Обработка аналогового компаратора
\$00b MAIN;		; Начало основной программы

Таблица 2.2. Сброс и векторы прерываний

№ вектора	Адрес	Источник	Наименование прерывания
1	\$000	RESET	Аппаратный вывод Reset, сброс по включению питания и прерывание по сторожевому таймеру
2	\$001	INT0	Внешнее прерывание 0
3	\$002	INT1	Внешнее прерывание 1
4	\$003	Timer 1 CAPT1	Захват таймера/счетчика 1
5	\$004	Timer 1 COMP1	Совпадение таймера/счетчика 1
6	\$005	Timer 1 OVFI	Переполнение таймера/счетчика 1
7	\$006	Timer 0 OVFO	Переполнение таймера/счетчика 0
8	\$007	UART, RX	Прием с последовательного порта завершен
9	\$008	UART, UDRE	Регистр данных последовательного порта пуст
10	\$009	UART, TX	Передача с последовательного порта завершена
11	\$00A	ANA COMP	Аналоговый компаратор

### Источники сброса

AT90S2313 имеет три источника сброса (перезапуска микроконтроллера):

- сброс по включению питания. Процессор сбрасывается при подаче питания на выводы VCC и GND;
- внешний сброс. Процессор сбрасывается при подаче низкого уровня на вывод RESET на время более двух периодов тактовой частоты;
- сброс от сторожевого таймера. Процессор сбрасывается по окончании времени отработки сторожевого таймера, если разрешена его работа.

Во время сброса все регистры ввода/вывода устанавливаются в начальные значения, программа начинает выполняться с адреса \$000, по этому адресу должна быть записана команда RJMP — относительный переход на программу обработки сброса. Если в программе не разрешаются прерывания и векторы прерываний не используются, программа может начинаться с нулевого адреса.

### **Сброс по включении питания**

Специальная схема, встроенная в микроконтроллер, — цепь сброса по включении питания, обеспечивает запрет включения процессора до тех пор, пока напряжение питания не достигнет безопасного уровня. После того как напряжение питания достигнет уровня включения, процессор не включается до тех пор, пока встроенный таймер не обработает несколько рабочих периодов сторожевого таймера.

Поскольку к выводу RESET подключен подтягивающий резистор, этот вывод может оставаться неподключенным, если не требуется внешний сброс. Время включения после подачи питания может быть увеличено удержанием вывода сброса на низком уровне.

### **Внешний сброс**

Внешний сброс обрабатывается по низкому уровню на выводе RESET. Вывод должен удерживаться в низком состоянии, по крайней мере, два периода тактовой частоты. После снятия сигнала 0 с вывода Reset через некоторое время (так же как при сбросе по включению питания) микроконтроллер запускается.

### **Сброс по сторожевому таймеру**

После отработки заданного при его инициализации интервала времени сторожевой таймер вырабатывает импульс сброса, перезапуская при этом микроконтроллер.

## **2.4. Обработка прерываний**

AT90S2313 имеет два регистра маски прерываний GIMSK — общий регистр маски прерываний и TIMSK — регистр маски прерываний от таймера/счетчика.

Когда возникает прерывание, бит глобального разрешения прерываний I сбрасывается (ему присваивается значение 0) и все прерывания запрещаются. Программа пользователя может установить этот бит для разрешения прерываний. Флаг разрешения прерываний I ав-

томатически устанавливается в 1 при выполнении команды выхода из прерывания — RETI.

Для прерываний, вызываемых статическими событиями, например переполнение таймера 0, флаг соответствующего прерывания устанавливается при возникновении события. Если флаг прерывания очищен и присутствует условие возникновения прерывания, флаг не будет установлен, пока не произойдет следующее событие.

Когда программный счетчик устанавливается на текущий вектор прерывания для его обработки, соответствующий флаг, сгенерированный прерыванием, аппаратно сбрасывается. Некоторые флаги прерывания могут быть сброшены записью логической единицы в бит, соответствующий флагу.

Бит	7	6	5	4	3	2	1	0	
\$3B (\$5B)	INT1	INT0	—	—	—	—	—	—	GIMSK
Чт./зап.	R/W	R/W	R	R	R	R	R	R	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.20. Общий регистр маски прерываний GIMSK

**Бит 7 — INT1:** запрос внешнего прерывания 1 разрешен. Когда этот бит установлен, а также установлен бит I, разрешается прерывание от внешнего вывода. Биты управления запуском прерывания (ISC11 и ISC10) в регистре управления микроконтроллером (MCUCR) определяют, по какому событию обрабатывается прерывание — по спадающему или нарастающему фронту или по уровню. Активность на выводе приводит к возникновению прерываний, даже если вывод сконфигурирован как выход. При возникновении прерывания выполняется переход по адресу \$001 для выполнения подпрограммы обработки прерывания.

**Бит 6 — INT0:** запрос внешнего прерывания 0 разрешен. Когда этот бит установлен, а также установлен бит I, разрешается прерывание от внешнего вывода. Биты управления запуском прерывания (ISC01 и ISC00) в регистре управления микроконтроллером (MCUCR) определяют, по какому событию обрабатывается прерывание — по спадающему или нарастающему фронту или по уровню. Активность на выводе приводит к возникновению прерываний, даже если вывод сконфигурирован как выход. При возникновении прерывания

вания выполняется переход по адресу \$001 для выполнения подпрограммы обработки прерывания.

**Биты 5...0** — в AT90S2313 эти биты зарезервированы и всегда читаются как 0.

Бит	7	6	5	4	3	2	1	0	
\$3A (\$5A)	INTF1	INTF0	—	—	—	—	—	—	GIFR
Чт./зап.	R/W	R/W	R	R	R	R	R	R	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.21. Общий регистр флагов прерываний GIFR

**Бит 7 — INTF1:** флаг внешнего прерывания 1. При возникновении на выводе INT1 события, вызывающего прерывание, INTF1 устанавливается в 1. Если установлены бит I регистра SREG и бит INT1 в GIMSK, происходит переход на вектор прерывания по адресу \$002. Флаг очищается после выполнения обработчика прерывания. Кроме того, флаг можно очистить, записав в него логическую единицу.

**Бит 6 — INTF0:** флаг внешнего прерывания 0. При возникновении на выводе INT0 события, вызывающего прерывание, INTF0 устанавливается в 1. Если установлены бит I регистра SREG и бит INT0 в GIMSK, происходит переход на вектор прерывания по адресу \$001. Флаг очищается после выполнения обработчика прерывания. Кроме того, флаг можно очистить, записав в него логическую единицу.

**Биты 5...0** — в AT90S2313 зарезервированы и всегда читаются как 0.

Бит	7	6	5	4	3	2	1	0	
\$39 (\$59)	TOIE1	OCIE1A	—	—	TICIE1	—	TOIE0	—	TIMSK
Чт./зап.	R/W	R/W	R	R	R/W	R	R/W	R	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.22. Регистр маски прерываний от таймера/счетчика TIMSK

**Бит 7 — TOIE1:** разрешение прерывания по переполнению таймера/счетчика 1. Если установлены этот бит и бит разрешения прерываний в регистре состояния, разрешены прерывания по переполнению таймера/счетчика 1. Соответствующее прерывание (вектор \$005) выполняется при переполнении таймера/счетчика 1. В регистре

флагов таймеров/счетчиков (TIFR) устанавливается флаг переполнения. Если таймер/счетчик 1 работает в режиме широтно-импульсной модуляции (ШИМ), флаг переполнения устанавливается при изменении направления счета, при значении \$0000.

**Бит 6 — OCIE1A:** разрешение прерывания по совпадению таймера/счетчика 1. Если установлены бит OCIE1A и бит разрешения прерывания в регистре состояния, разрешены прерывания по совпадению таймера/счетчика 1. Прерывание (вектор \$004) выполняется при равенстве таймера/счетчика 1 и регистра совпадения. Во флаговом регистре TIFR устанавливается в 1 флаг совпадения.

**Биты 5, 4 —** в AT90S2313 зарезервированы и всегда читаются как 0.

**Бит 3 — TICIE1:** разрешение прерывания по входу захвата. Если установлены бит TICIE1 и бит разрешения прерывания в регистре состояния, разрешены прерывания по входу захвата. Соответствующее прерывание (вектор \$003) выполняется по сигналу захвата на выводе 11 (PD6/ICP). Во флаговом регистре TIFR устанавливается флаг захвата.

**Бит 2 —** в AT90S2313 зарезервирован и всегда читается как 0.

**Бит 1 — TOIE0:** разрешение прерывания по переполнению таймера/счетчика 0. Если этот бит установлен в 1 и бит I в регистре состояния установлен в 1, разрешены прерывания по переполнению таймера/счетчика 0. При возникновении переполнения выполняется переход на соответствующий вектор прерывания (\$006). Флаг переполнения (TOV0) во флаговом регистре прерываний (TIFR) таймеров/счетчиков устанавливается в 1.

**Бит 0 —** в AT90S2313 зарезервирован и всегда читается как 0.

Бит	7	6	5	4	3	2	1	0	
\$38 (\$58)	TOV1	OCF1A	—	—	ICF1	—	TOV0	—	TIFR
Чт./зап.	R/W	R/W	R	R	R/W	R	R/W	R	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.23. Регистр флагов прерываний от таймеров/счетчиков TIFR

**Бит 7 — TOV1:** флаг переполнения таймера/счетчика 1. Флаг TOV1 устанавливается в 1 при возникновении переполнения таймера/счетчика 1. Флаг TOV1 сбрасывается аппаратно при выполнении соответствующего вектора обработки прерывания. Кроме того, флаг

можно сбросить, записав в него логическую единицу. Если установлены бит I в SREG и бит TOIE1 в TIMSK, при установке бита TOV1 выполняется прерывание по переполнению таймера/счетчика 1. В режиме ШИМ этот бит устанавливается, когда таймер/счетчик 1 изменяет направление счета при значении \$0000.

**Бит 6 — OCF1A:** флаг выхода совпадения 1А. Флаг устанавливается в 1 если происходит совпадение значения таймера/счетчика 1 и данных в регистре OCR1A. Флаг очищается аппаратно при выполнении соответствующего вектора прерывания. Кроме того, флаг можно сбросить, записав в него логическую единицу. Если установлены бит I в SREG и бит OCIE1A в TIMSK, при установке бита OCF1A выполняется прерывание.

**Биты 5, 4** — в AT90S2313 зарезервированы и всегда читаются как 0.

**Бит 3 — ICF1:** флаг входа захвата 1. Бит устанавливается в 1 при возникновении события захвата по входу, он индицирует, что значение таймера/счетчика 1 скопировано в регистр захвата по входу ICR1. ICF1 очищается при выполнении соответствующего вектора обработки прерывания. Кроме того, флаг можно очистить, записав в него логическую единицу.

**Бит 2** — в AT90S2313 зарезервирован и всегда читается как 0.

**Бит 1 — TOV0:** флаг переполнения таймера счетчика 1. Флаг TOV0 устанавливается при переполнении таймера/счетчика 0. Флаг сбрасывается аппаратно при выполнении соответствующего вектора прерывания. Кроме того, флаг можно очистить, записав в него логическую единицу. Если установлены бит I в SREG и бит TOIE0 в TIMSK, при установке бита TOV0 выполняется прерывание по переполнению таймера/счетчика 0.

**Бит 0** — в AT90S2313 зарезервирован и всегда читается как 0.

### Внешние прерывания

Внешние прерывания управляются выводами INT0 и INT1. Обратите внимание на то, что прерывания обрабатываются, даже когда выводы сконфигурированы как выходы. Это позволяет генерировать программные прерывания. Внешние прерывания могут возникать по спадающему или нарастающему фронту, а также по низкому уровню. Это устанавливается в регистре управления процессором MCUCR.

Если внешние прерывания разрешены и сконфигурированы на обработку по уровню, прерывание будет вырабатываться до тех пор, пока вывод удерживается в низком состоянии.

Управление работой внешних прерываний рассмотрено при описании регистра управления процессором MCUCR.

### Время реакции на прерывание

Минимальное время реакции на любое из предусмотренных процессоре прерываний — 4 периода тактовой частоты. После 4 циклов вызывается программный вектор, обрабатывающий данное прерывание. За эти 4 цикла программный счетчик записывается в стек, указатель стека уменьшается на 2. Программный вектор представляет собой относительный переход на подпрограмму обслуживания прерывания, и этот переход занимает 2 периода тактовой частоты. Если прерывание происходит во время выполнения команды, длящейся несколько циклов, перед вызовом прерывания завершается выполнение этой команды.

Выход из программы обслуживания прерывания занимает 4 периода тактовой частоты. За эти 4 периода из стека восстанавливается программный счетчик. После выхода из прерывания процессор всегда выполняет еще одну команду, прежде чем обслужить любое отложенное прерывание.

Заметим, что регистр состояния SREG аппаратно не сохраняется процессором как при вызове подпрограмм, так и при обслуживании прерываний. Если программа требует сохранения SREG, оно должно производиться программой пользователя.

Бит	7	6	5	4	3	2	1	0	
\$35 (\$55)	—	—	SE	SM	ISC11	ISC10	ISC01	ISC00	MCUCR
Чт./зап.	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.24. Регистр управления микроконтроллером MCUCR

**Биты 7, 6** — в AT90S2313 эти биты зарезервированы и всегда читаются как 0.

**Бит 5** — SE: разрешение режима Sleep. Этот бит должен быть установлен в 1, чтобы при выполнении команды SLEEP процессор

переходил в режим пониженного энергопотребления. Этот бит должен быть установлен в 1 до исполнения команды SLEEP.

**Бит 4 — SM:** режим Sleep. Этот бит выбирает один из двух режимов пониженного энергопотребления. Если бит сброшен, в качестве режима Sleep выбирается холостой режим (Idle mode). Если бит установлен — выбирается экономичный режим (Power Down). Особенности каждого из режимов будут рассмотрены ниже.

**Биты 3, 2 — ISC11, ISC10:** биты управления срабатыванием прерывания 1. Внешнее прерывание активируется выводом INT1, если установлен флаг I регистра состояния SREG и установлена соответствующая маска в регистре GIMSK. Срабатывание по уровню и фронтам задается следующим образом:

Таблица 2.3. Управление срабатыванием прерывания 1

ISC11	ISC10	Описание
0	0	Запрос на прерывание генерируется по низкому уровню на входе INT1
0	1	Зарезервировано
1	0	Запрос на прерывание по спадающему фронту на входе INT1
1	1	Запрос на прерывание по нарастающему фронту на входе INT1

Примечание. При изменении битов ISC11/ISC10 прерывание INT1 должно быть запрещено очисткой соответствующего бита в регистре GIMSK. В противном случае прерывание может возникнуть во время изменения битов.

Таблица 2.4. Управление срабатыванием прерывания 0

ISC01	ISC00	Описание
0	0	Запрос на прерывание генерируется по низкому уровню на входе INT0
0	1	Зарезервировано
1	0	Запрос на прерывание по спадающему фронту на входе INT0
1	1	Запрос на прерывание по нарастающему фронту на входе INT0

Примечание. При изменении битов ISC01 и ISC00, прерывания по входу INT0 должны быть запрещены сбросом бита разрешения прерывания в регистре GIMSK. В противном случае прерывание может произойти при изменении значения битов.

**Биты 1, 0 — ISC01, ISC00:** биты управления срабатыванием прерывания 0. Внешнее прерывание активируется выводом INT0, если установлен флаг 1 регистра состояния SREG и установлена соответствующая маска в регистре GIMSK. В табл. 2.4 приведена установка битов для задания срабатывания по уровню и фронтам.

## 2.5. Режимы пониженного энергопотребления

Для запуска режима пониженного энергопотребления должен быть установлен в состояние 1 бит SE регистра MCUCR и должна быть выполнена команда SLEEP. Если во время нахождения в режиме пониженного потребления происходит одно из разрешенных прерываний, процессор начинает работать, исполняет подпрограмму обработки прерывания и продолжает выполнение программы с команды, следующей за SLEEP. Содержимое регистрового файла и памяти ввода/вывода не изменяется. Если в режиме пониженного потребления происходит сброс, процессор начинает выполнение программы с вектора сброса.

Если для вывода из экономичного режима используется прерывание по уровню, низкий уровень должен удерживаться на время, достаточное для запуска генератора тактовых импульсов, — не менее 16 мс. Иначе флаг прерывания может вернуться в 0 до того, как процессор начнет работу.

### Режим холостого хода

Когда бит SM сброшен, команда SLEEP переводит процессор в режим холостого хода (Idle mode). ЦПУ останавливается, но таймеры/счетчики, сторожевой таймер и система прерываний продолжают работать. Это позволяет процессору возобновлять работу как от внешних прерываний, так и по переполнению таймеров/счетчиков или по сбросу от сторожевого таймера. Если прерывание от аналогового компаратора не требуется, аналоговый компаратор может быть отключен установкой бита ACD регистра ACSR. Это уменьшает потребляемую мощность.

## Экономичный режим

Когда бит  $SM = 1$ , команда SLEEP переводит процессор в экономичный режим (Power Down Mode). В этом режиме останавливается генератор тактовых импульсов. Программист может разрешить работу сторожевого таймера в этом режиме. Если сторожевой таймер разрешен, процессор выходит из экономичного режима после отработки периода сторожевого таймера. Если сторожевой таймер запрещен, выход из экономичного режима может произойти только по внешнему сбросу или внешнему прерыванию по уровню.

## 2.6. Таймеры/счетчики

В AT90S2313 предусмотрены два таймера/счетчика общего назначения: 8-разрядный и 16-разрядный. Каждый из таймеров индивидуально подключается к одному из выходов 10-разрядного предварительного делителя частоты. Оба таймера могут использоваться как таймеры с внутренним источником импульсов или как счетчики импульсов, поступающих извне.

В качестве источника импульсов для таймеров можно выбрать сигнал тактовой частотой микроконтроллера (СК), импульсы предварительного делителя (СК/8, СК/64, СК/256 или СК/1024) или импульсы с соответствующего внешнего вывода. На рис. 2.25 изображена структурная схема предварительного делителя частоты.

Кроме того, таймеры могут быть остановлены.

### 8-разрядный таймер/счетчик 0

8-разрядный таймер/счетчик может получать импульсы тактовой частоты — СК, импульсы с предварительного делителя (СК/8, СК/64, СК/256 или СК/1024), импульсы с внешнего вывода или быть остановлен соответствующими установками регистра TCCR0. Флаг переполнения таймера находится в регистре TIFR. Биты управления таймером расположены в регистре TCCR0. Разрешение и запрещение прерываний от таймера управляется регистром TIMSK.

При работе таймера/счетчика от внешнего сигнала этот сигнал синхронизируется с тактовым генератором микроконтроллера. Для

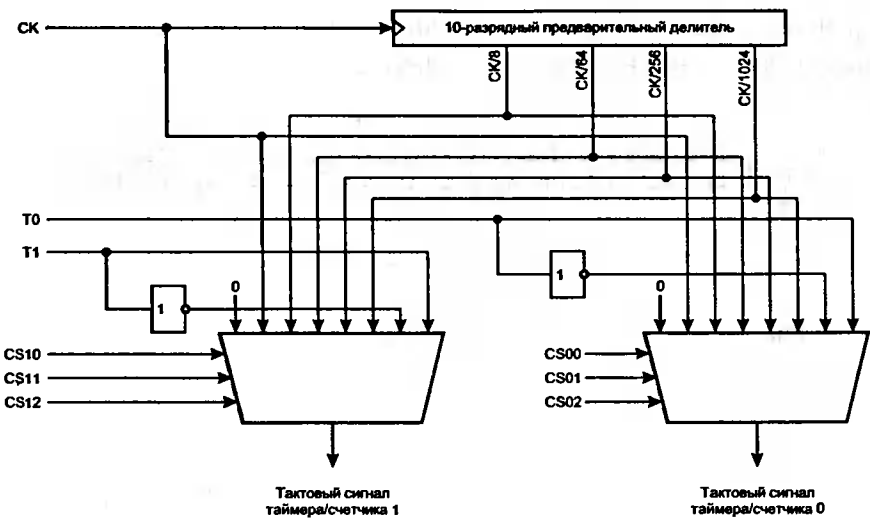


Рис. 2.25. Предварительный делитель тактовой частоты для таймеров

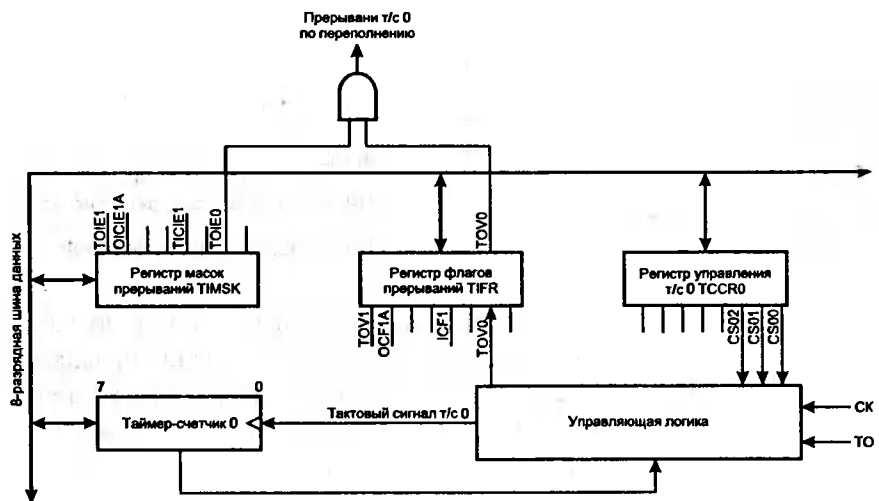


Рис. 2.26. Блок-схема таймера/счетчика 0

правильной обработки внешнего сигнала минимальное время между соседними импульсами должно превышать период тактовой частоты

процессора. Сигнал внешнего источника обрабатывается по спадающему фронту тактовой частоты процессора.

Бит	7	6	5	4	3	2	1	0	
\$33 (\$53)	—	—	—	—	—	CS02	CS01	CS00	TCCR0
Чт./зап.	R	R	R	R	R	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.27. Регистр управления таймером/счетчиком 0 TCCR0

**Биты 7...3** — в AT90S2313 зарезервированы и всегда читаются как 0.

**Биты 2, 1, 0** — CS02, CS01, CS00 — выбор тактовой частоты. Эти биты задают коэффициент деления предварительного делителя.

Таблица 2.5. Выбор коэффициента предварительного деления

CS02	CS01	CS00	Описание
0	0	0	Таймер/счетчик остановлен
0	0	1	СК
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Внешний вывод T0, нарастающий фронт
1	1	1	Внешний вывод T0, спадающий фронт

Условие «Таймер/счетчик остановлен» запрещает или разрешает функционирование таймера/счетчика. В режимах деления используется частота тактового генератора микроконтроллера. При работе от внешнего источника предварительно должен быть установлен соответствующий бит регистра направления данных.

Бит	7	6	5	4	3	2	1	0	
\$32 (\$52)	MSB							LSB	TCNT0
Чт./зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.28. Таймер/счетчик 0 TCNT0

Таймер/счетчик реализован как нарастающий счетчик с возможностью чтения и записи. При записи таймера/счетчика, если присутствуют тактовые импульсы, таймер/счетчик продолжает счет в следующем за операцией записи тактовом цикле таймера.

### 16-разрядный таймер/счетчик 1

16-разрядный таймер/счетчик может получать импульсы тактовой частоты — СК, импульсы с предварительного делителя (СК/8, СК/64, СК/256 или СК/1024), импульсы с внешнего вывода или быть

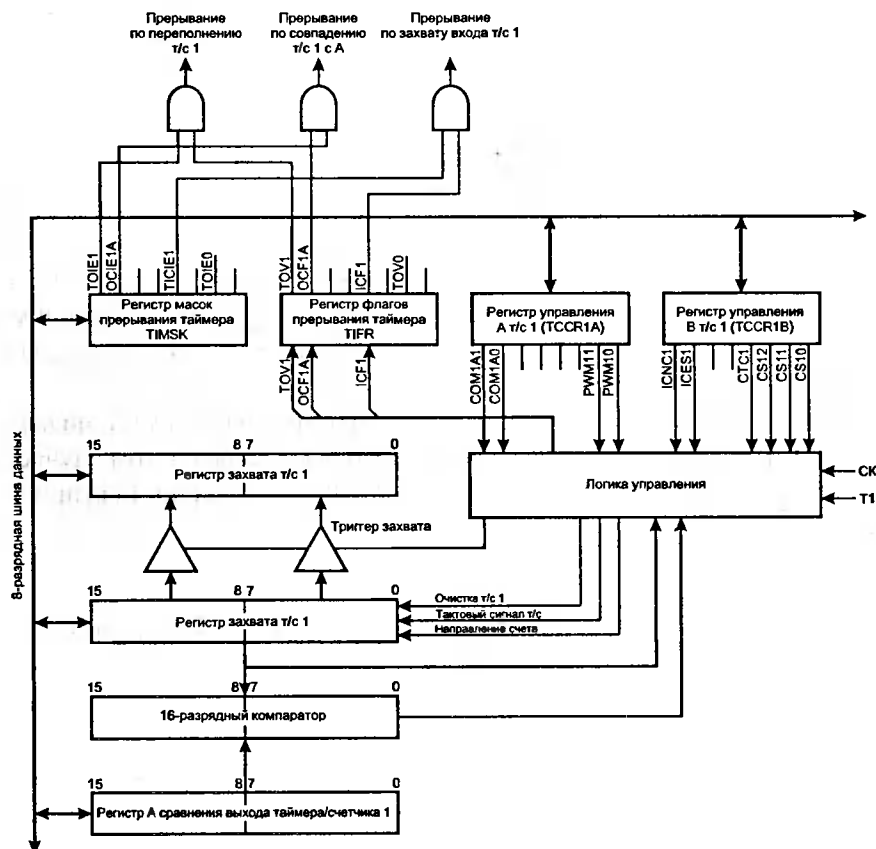


Рис. 2.29. Блок-схема таймера/счетчика 1

остановлен соответствующими установками регистра TCCR1A. Флаги состояния таймера (переполнения, совпадения и захвата) и управляющие сигналы находятся в регистре TIFR. Разрешение и запрещение прерываний от таймера 1 управляется регистром TIMSK.

При работе таймера/счетчика 1 от внешнего сигнала этот сигнал синхронизируется с тактовым генератором микроконтроллера. Для правильной обработки внешнего сигнала минимальное время между соседними импульсами должно превышать период тактовой частоты процессора. Сигнал внешнего источника обрабатывается по спадающему фронту тактовой частоты процессора.

Таймер/счетчик 1 поддерживает функцию совпадения, используя регистр совпадения OCR1A в качестве источника для сравнения с содержимым счетчика. Функция совпадения поддерживает очистку счетчика и переключение выхода по совпадению.

Таймер/счетчик 1 можно использовать как 8-, 9- или 10-разрядный широтно-импульсный модулятор (ШИМ). В этом режиме счетчик и регистр OCR1 работают как защищенный от дребезга независимый ШИМ с отцентрованными импульсами. Подробнее эта функция будет описана ниже.

Функция захвата по входу предусматривает захват содержимого таймера/счетчика 1 в регистр захвата ICR1 и управляется внешним сигналом на входе захвата — ICP. Работа режима захвата определяется управляющим регистром TCCR1.

При работе захвата по входу может быть включена схема подавления шума, при этом сигнал захвата возникает только в том случае, если событие, управляющее захватом, наблюдается на протяжении 4 машинных циклов.

Бит	7	6	5	4	3	2	1	0	
\$2F (\$4F)	COM1A1	COM1A0	—	—	—	—	PWM11	PWM10	TCCR1A
Чт./зап.	R/W	R/W	R	R	R	R	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.30. Регистр А управления таймером/счетчиком 1 TCCR1A

Биты 7, 6 — COM1A1, COM1A0: режим выхода совпадения, биты 1 и 0. Эти управляющие биты задают отклик вывода OC1 процессора на совпадение регистра сравнения и таймера/счетчика 1. Поскольку это альтернативная функция порга, соответствующий бит

направления должен устанавливать вывод на выход. Конфигурация управляющих битов показана в следующей таблице.

Таблица 2.6. Установка режима совпадения

COM1A1	COM1A0	Описание
0	0	Таймер/счетчик 1 отключен от вывода OC1
0	1	Переключение выхода OC1
1	0	Сброс (0) вывода OC1
1	1	Установка (1) вывода OC1

В режиме ШИМ эти биты имеют другие функции, которые указаны в табл. 2.10.

При изменении битов COM1A1 и COM1A0 прерывание по совпадению должно быть запрещено очисткой соответствующего бита в регистре TIMСК. В противном случае прерывание может произойти во время изменения этих битов.

Биты 5...2 — в AT90S2313 зарезервированы и всегда читаются как 0.

Биты 1, 0 — PWM11, PWM10: биты установки ШИМ. Эти биты устанавливают режим работы таймера/счетчика 1 в качестве ШИМ (табл. 2.7). Подробнее этот режим будет рассмотрен ниже.

Таблица 2.7. Установка режима работы ШИМ

PWM11	PWM10	Описание
0	0	Работа ШИМ запрещена
0	1	8-разрядный ШИМ
1	0	9-разрядный ШИМ
1	1	10-разрядный ШИМ

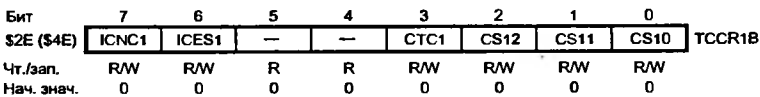


Рис. 2.31. Регистр управления таймером/счетчиком 1 TCCR1B

Бит 7 — ICNC1: подавитель входного шума входа захвата. Если ICNC1 = 0, подавление входного шума входа захвата запрещено. При

этом захват срабатывает по первому заданному (нарастающему или спадающему) фронту сигнала на выводе ICP. При установке бита обрабатываются четыре последовательные выборки сигнала на выводе ICP. Для срабатывания захвата все выборки должны соответствовать уровню, заданному битом ICES1. Частота выборок равна тактовой частоте процессора.

**Бит 6 — ICES1:** выбор фронта сигнала захвата. Если бит ICES1 = 0, содержимое таймера/счетчика 1 переписывается в регистр захвата по спадающему фронту сигнала на выводе ICP. Если бит установлен — по нарастающему фронту сигнала.

**Биты 5, 4** — в AT90S2313 зарезервированы и всегда читаются как 0.

**Бит 3 — CTC1:** очистка таймера/счетчика 1 по совпадению. Если бит CTC1 = 1, таймер/счетчик 1 устанавливается в \$0000 в такте, следующем за событием совпадения. Если бит сброшен, таймер/счетчик 1 продолжает считать, пока не будет остановлен, сброшен, произойдет его переполнение или изменение направления счета. В режиме ШИМ этот бит не работает.

**Биты 2, 1, 0 — CS12, CS11, CS10:** выбор источника тактирования. Эти биты определяют источник счетных импульсов для таймера/счетчика 1.

Таблица 2.8. Выбор источника счетных импульсов

CS12	CS11	CS10	Описание
0	0	0	Таймер/счетчик 1 остановлен
0	0	1	СК
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Спадающий фронт на выводе T1
1	1	1	Нарастающий фронт на выводе T1

Бит	15	14	13	12	11	10	9	8	
\$2D (\$4D)	MSB								TCNT1H
\$2C (\$4C)								LSB	TCNT1L
Чт./зап.	7	6	5	4	3	2	1	0	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Рис. 2.32. Таймер/счетчик 1 TCNT1H и TCNT1L

Это 16-разрядный регистр, содержащий текущее значение таймера/счетчика 1. Чтобы чтение и запись двух байтов счетчика происходило синхронно, для работы с ним используется временный регистр (TEMP).

**Запись в таймер/счетчик 1:** при записи старшего байта в TCNT1H записываемые данные помещаются в регистр TEMP. Затем, при записи младшего байта, он вместе с данными из TEMP переписывается в таймер/счетчик 1. Таким образом, при записи 16-разрядного значения первым должен записываться байт в TCNT1H;

**Чтение таймера/счетчика 1:** при чтении младшего байта из TCNT1L он посылается в процессор, а данные из TCNT1H переписываются в регистр TEMP, т. е. одновременно читаются все 16 разрядов. При последующем чтении регистра TCNT1H данные берутся из регистра TEMP.

Таймер/счетчик 1 организован как суммирующий счетчик (в режиме ШИМ — суммирующий/вычитающий) с возможностью чтения и записи. Если выбран источник тактовых импульсов для таймера/счетчика 1, после записи в него нового значения он продолжает счет в следующем после записи периоде тактовой частоты.

Бит	15	14	13	12	11	10	9	8	
\$2B (\$4B)	MSB								OCR1AH
\$2A (\$4A)								LSB	OCR1AL
Чт./зап.	7	6	5	4	3	2	1	0	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Рис. 2.33. Регистр совпадения А таймера/счетчика 1 — OCR1AH и OCR1AL

Регистр совпадения — 16-разрядный регистр, доступный для чтения и записи.

В этом регистре хранятся данные, которые непрерывно сравниваются с текущим значением таймера/счетчика 1. Действие по совпадению задается регистрами управления таймером/счетчиком 1 и регистром состояния.

Поскольку регистр OCR1A 16-разрядный, при записи нового значения в регистр, для того чтобы оба байта регистра записывались одновременно, используется временный регистр. При записи старшего байта данные помещаются во временный регистр, который переписывается в OCR1AH при записи младшего байта в OCR1AL. Таким образом, для записи в регистр первым должен записываться старший байт.

Бит	15	14	13	12	11	10	9	6	
\$25 (\$45)	MSB								ICR1H ICR1L
\$24 (\$44)								LSB	
	7	6	5	4	3	2	1	0	
Чт./зап.	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Нач. знач.	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Рис. 2.34. Регистр захвата таймера/счетчика 1 — ICR1H и ICR1L

Регистр захвата — 16-разрядный регистр, доступный только для чтения. По нарастающему или спадающему фронту (в соответствии с выбором фронта импульса захвата ICES1) сигнала на выводе ICP текущее значение таймера/счетчика 1 переписывается в регистр захвата ICR1. В это же время устанавливается флаг захвата ICF1.

Поскольку регистр захвата является 16-разрядным, для чтения его значения, чтобы оба байта прочитались одновременно, используется временный регистр. При чтении младшего байта ICR1L старший байт регистра ICR1H переписывается во временный регистр. При чтении старшего байта он принимается из временного регистра. Таким образом, для чтения 16-разрядного регистра первым должен читаться младший байт.

### Таймер/счетчик в режиме ШИМ

При выборе режима широтно-импульсной модуляции (ШИМ) таймер/счетчик 1 и регистр совпадения OCR1A формируют 8-, 9- или 10-разрядный непрерывный, свободный от «дрожания» и правиль-

ный по фазе сигнал, выводимый на вывод PB3(OC1). Таймер/счетчик 1 работает как реверсивный счетчик, считающий от 0 до конечного значения (табл. 2.9). При достижении конечного значения счетчик начинает считать в обратную сторону до нуля, после чего рабочий цикл повторяется. Когда значение счетчика совпадает с 8, 9 или 10 младшими битами регистра OCR1A, вывод PD1(OC1) устанавливается или сбрасывается в соответствии с установками битов COM1A1 и COM1A0 в регистре TCCR1 (табл. 2.10).

Таблица 2.9. Конечное значение таймера и частота ШИМ

Разрешение ШИМ	Конечное значение таймера	Частота ШИМ
8 битов	\$00FF (255)	Ftc1/510
9 битов	\$01FF (511)	Ftc1/1022
10 битов	\$03FF (1023)	Ftc1/2046

В режиме ШИМ при записи в регистр OCR1A 10 младших битов передаются во временный регистр и переписываются только при достижении таймером/счетчиком конечного значения. При этом устраняется появление несимметричных импульсов (дрожания), которые неизбежны при асинхронной записи OCR1A.

Таблица 2.10. Установка режима совпадения при работе ШИМ

COM1A1	COM1A0	Влияние на вывод OC1
0	0	Не подключен
0	1	Не подключен
1	0	Очищается при совпадении, для возрастания счетчика и сбрасывается для уменьшения (неинвертирующий ШИМ)
1	1	Очищается при совпадении, для уменьшения счетчика и сбрасывается для возрастания (инвертирующий ШИМ)

Если OCR1A содержит значение \$0000 или конечное значение (TOP), вывод OC1 остается в том состоянии, которое определяется установками COM1A1 и COM1A0. Это показано в табл. 2.11.

**Таблица 2.11. Выход ШИМ для OCR = \$0000 или TOP**

COM1A1	COM1A0	OCR1A	Вывод OC1
1	0	\$0000	Низкий
1	0	TOP	Высокий
1	1	\$0000	Высокий
1	1	TOP	Низкий

В режиме ШИМ флаг переполнения таймера 1 (TOV1) устанавливается, когда счетчик изменяет направление счета в точке \$0000. Прерывание по переполнению таймера 1 работает как при нормальном режиме работы таймера/счетчика, т. е. оно выполняется, если установлен флаг TOV1 и разрешены соответствующие прерывания. То же самое касается флага совпадения и прерывания по совпадению.

### Сторожевой таймер

Сторожевой таймер работает от отдельного встроенного генератора, работающего на частоте 1 МГц (это типовое значение частоты для питания 5 В). Управляя предварительным делителем сторожевого таймера, можно задавать интервал сброса таймера от 16 до 2048 мс. Команда WDR сбрасывает сторожевой таймер. Для работы сторожевого таймера можно выбрать одно из восьми значений частоты, что позволяет в широких пределах изменять время между исполнением команды WDR и сбросом процессора. При отработке периода работы сторожевого таймера, если не поступила команда WDR, AT90S2313 сбрасывается, выполнение программы продолжается с вектора сброса.

Для предотвращения нежелательного отключения сторожевого таймера, для его запрещения должна выполняться определенная последовательность, которая описана при рассмотрении регистра WDTCR.

Бит	7	6	5	4	3	2	1	0	
\$21 (\$41)	—	—	—	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
Чт./зап.	R	R	R	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.35. Регистр управления сторожевым таймером — WDTCR

**Биты 7...5** — в AT90S2313 зарезервированы и всегда читаются как 0.

**Бит 4 — WDTOE** — разрешение выключения сторожевого таймера. При очистке бита WDE этот бит должен быть установлен, иначе работа сторожевого таймера не прекращается. Через четыре такта после установки этого бита он аппаратно сбрасывается.

**Бит 3 — WDE** — разрешение сторожевого таймера. Если бит установлен, работа сторожевого таймера разрешена, если бит сброшен — запрещена. Сброс бита производится только в том случае, если бит WDTOE установлен в 1.

**Биты 2...0 — WDP2...0** — биты предварительного делителя сторожевого таймера. Если работа сторожевого таймера разрешена, эти биты определяют предварительный коэффициент деления для сторожевого таймера. В табл. 2.12 приведены различные значения установок предварительного делителя и соответствующие им временные интервалы при напряжении питания  $V_{cc} = 5 \text{ В}$ .

Для запрещения включенного сторожевого таймера должна исполняться следующая процедура:

1. Одной командой записать 1 в WDTOE и WDE. Единица в WDE должна записываться даже в том случае, если этот бит был установлен перед началом процедуры остановки таймера.

2. В течение следующих четырех тактов процессора необходимо записать в WDE логический 0, при этом работа сторожевого таймера запрещается.

Таблица 2.12. Установки предварительного делителя сторожевого таймера

WDP2	WDP1	WDP0	Период времени, мс	WDP2	WDP1	WDP0	Период времени, мс
0	0	0	16	1	0	0	256
0	0	1	32	1	0	1	512
0	1	0	64	1	1	0	1024
0	1	1	128	1	1	1	2048

## 2.7. Чтение и запись в энергонезависимую память

Регистры доступа к энергонезависимой памяти (EEPROM) расположены в пространстве ввода/вывода.

Время записи лежит в диапазоне 2,5...4 мс и зависит от напряжения питания. Если программа пользователя производит запись в энергонезависимую память, должны быть предприняты некоторые меры предосторожности. При использовании в источнике питания конденсаторов большой емкости напряжение питания нарастает и спадает достаточно медленно. Это приводит к тому, что процессор некоторое время работает при напряжении питания ниже минимума, достаточного для нормальной работы схем тактирования. При этом он может совершать нежелательные переходы, попадая на части программы, производящие запись в EEPROM. В таких случаях для защиты содержимого EEPROM необходимо использовать внешние схемы, формирующие сигнал сброса при уменьшении напряжения питания. При использовании внешнего супервайзера питания (например, K1171СП47 — для него граница срабатывания составляет 4,7 В), подающего на вывод Reset напряжение низкого уровня при напряжении питания ниже допустимого, практически можно быть уверенным в том, что содержимое памяти EEPROM не будет портиться независимо от емкости конденсатора на выходе источника питания.

При записи или чтении EEPROM процессор приостанавливается на 2 машинных цикла до начала выполнения следующей команды.

Бит	7	6	5	4	3	2	1	0	
\$1E (\$3E)	—	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEAR
Чт./зап.	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.36. Регистр адреса EEPROM — EEAR

**Бит 7** — в AT90S2313 зарезервирован и всегда читается как 0.

**Биты 6...0** — EEAR6...0 — адрес EEPROM. Адресный регистр EEPROM задает адрес в 128-байтовом пространстве EEPROM. Байты данных EEPROM адресуются линейно в диапазоне 0...127.

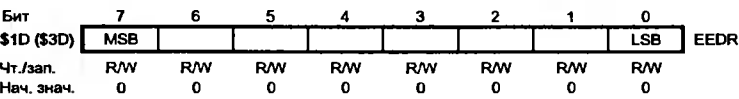


Рис. 2.37. Регистр данных EEPROM — EEDR

**Биты 7...0 — EEDR7...0** — Данные EEPROM. При записи регистр EEDR содержит данные, которые должны записываться в EEPROM по адресу, находящемуся в регистре EEAR. После операции чтения в этот регистр записываются данные, прочитанные из EEPROM, по адресу, заданному в регистре EEAR.

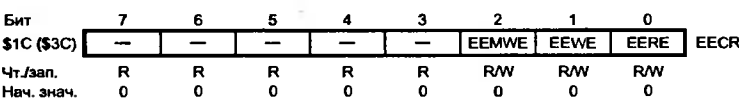


Рис. 2.38. Регистр управления EEPROM — EECR

**Биты 7...3** — в AT90S2313 зарезервированы и всегда читаются как 0.

**Бит 2 — EEMWE** — управление разрешением записи. Этот бит определяет, будут ли записаны данные при установке EEWE. Если бит EEMWE установлен, при установке EEWE данные записываются по выбранному адресу EEPROM. Если этот бит сброшен, установка EEWE не имеет эффекта. После программной установки этот бит сбрасывается аппаратно через четыре такта процессора.

**Бит 1 — EEWE** — разрешение записи в EEPROM. Сигнал EEWE является стробом записи в EEPROM. После установки правильных адреса и данных для записи в EEPROM необходимо установить бит EEWE. При записи 1 в бит EEWE должен быть установлен бит EEMWE, тогда происходит запись в EEPROM. Для записи в EEPROM должна соблюдаться следующая последовательность:

1. Ждать обнуления EEWE.
2. Записать адрес в EEAR.
3. Записать данные в EEDR.
4. Установить в 1 бит EEMWE.

5. Не позже чем через 4 такта после установки EEMWE установить EEWE.

После того, как время записи истечет (примерно 2,5 мс для  $V_{CC} = 5$  В и 4 мс для  $V_{CC} = 2,7$  В), бит EEWЕ очищается аппаратно. Программист может проверять этот бит и ожидать его установки в ноль перед тем, как записывать следующий байт. При установке EEWЕ микроконтроллер останавливается на два цикла перед исполнением следующей команды.

**Бит 0 — EERE** — разрешение чтения из EEPROM. Сигнал EERE является стробом чтения из EEPROM. После установки нужного адреса в регистре EEAR должен быть установлен бит EERE. После того как бит EERE будет аппаратно очищен, затребованные данные будут находиться в регистре EEDR. Чтение EEPROM занимает одну команду и не требует отслеживания бита EERE. После установки бита EERE микроконтроллер останавливается на 4 цикла перед тем, как будет выполнена следующая команда. Перед чтением пользователь должен проверить состояние бита EEWЕ. Если изменять содержимое регистров EEPROM во время операции записи в EEPROM, запись в ячейку памяти прерывается и результат операции записи становится неопределенным.

## 2.8. Универсальный асинхронный приемопередатчик

Микроконтроллер AT90S2313 имеет встроенный универсальный асинхронный приемопередатчик (UART).

Основные характеристики UART:

- генерация произвольных значений скорости;
- высокая скорость даже при низких тактовых частотах;
- 8 или 9 битов данных;
- фильтрация шума;
- определение переполнения;
- детектирование ошибки кадра;
- определение неверного стартового бита;
- три отдельных прерывания — завершение передачи, очистка регистра передачи и завершение приема.

Передача данных

Структурная схема узла передачи данных UART показана на рис. 2.39.

Передача данных инициируется записью передаваемых данных в регистр ввода/вывода данных UART — UDR. Данные пересылаются из UDR в сдвиговый регистр передатчика, когда:

- новый символ записывается в UDR после того, как был выдвинут стоповый бит для предыдущего символа. При этом сдвиговый регистр загружается сразу;
- новый символ записывается в UDR до того, как выдвинут стоповый бит для предыдущего символа. При этом сдвиговый регистр записывается сразу после того, как будет выдвинут стоповый бит предыдущего символа.

При этом в регистре состояния UART — USR устанавливается бит UDRE — признак очистки регистра данных. Когда этот бит уста-

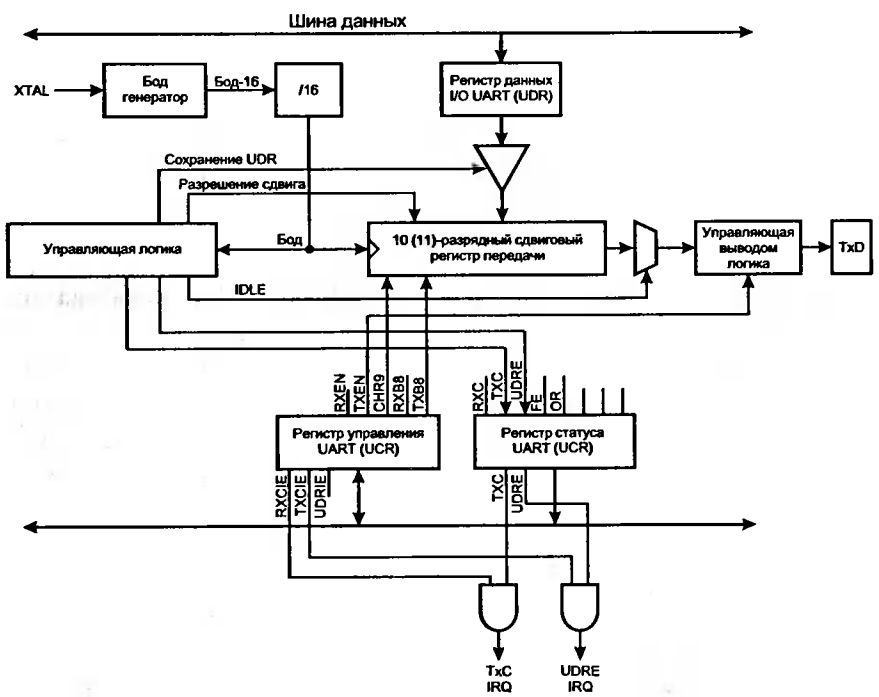


Рис. 2.39. Структурная схема узла передачи данных UART

новлен, UART готов к приему следующего символа. При перезаписи UDR в 10(11)-разрядный сдвиговый регистр бит 0 сдвигового регистра обнуляется (стартовый бит), а бит 9 или 10 устанавливается (стоповый бит). Если выбрано 9-битовое слово данных (установлен бит CHR9 в регистре UCR), бит TXB8 из UCR переписывается в 9-й бит сдвигового регистра передатчика.

После тактового импульса, следующего с частотой передачи, стартовый бит выдвигается на вывод TXD. Затем выдвигаются данные, начиная с младшего бита. После того как выдвинут стоповый бит, в сдвиговый регистр загружаются новые данные, если они были записаны в UDR во время передачи. При загрузке устанавливается бит UDRE. Если до выдвижения стопового бита в регистр UDR не поступают новые данные, UDRE остается установленным до последующей записи UDR. Если новые данные не поступили и на выводе TXD появляется стоповый бит, в регистре USR устанавливается флаг окончания передачи — TXC.

Установка бита TXEN в UCR разрешает работу передатчика. При очистке бита TXEN вывод PD1 можно использовать для ввода/вывода данных. Если бит TXEN установлен, передатчик UART подключен к выводу PD1 независимо от установки бита DDD1 в регистре DDRD.

### Прием данных

Структурная схема узла приема данных UART показана на рис. 2.40.

Логическая схема приемника обрабатывает сигнал на выводе RxD с частотой в 16 раз больше скорости передачи (для обработки одного бита принимаемой последовательности производится 16 выборок входного сигнала). В состоянии ожидания одна выборка логического нуля интерпретируется как спадающий фронт стартового бита, после чего запускается последовательность обнаружения стартового бита. Если в первой выборке сигнала обнаружен нулевой отсчет, приемник обрабатывает 8, 9 и 10 выборки сигнала на выводе RxD. Если хотя бы две из трех выборок равны логической единице, стартовый бит считается шумом и приемник ждет следующего перехода из 1 в 0.

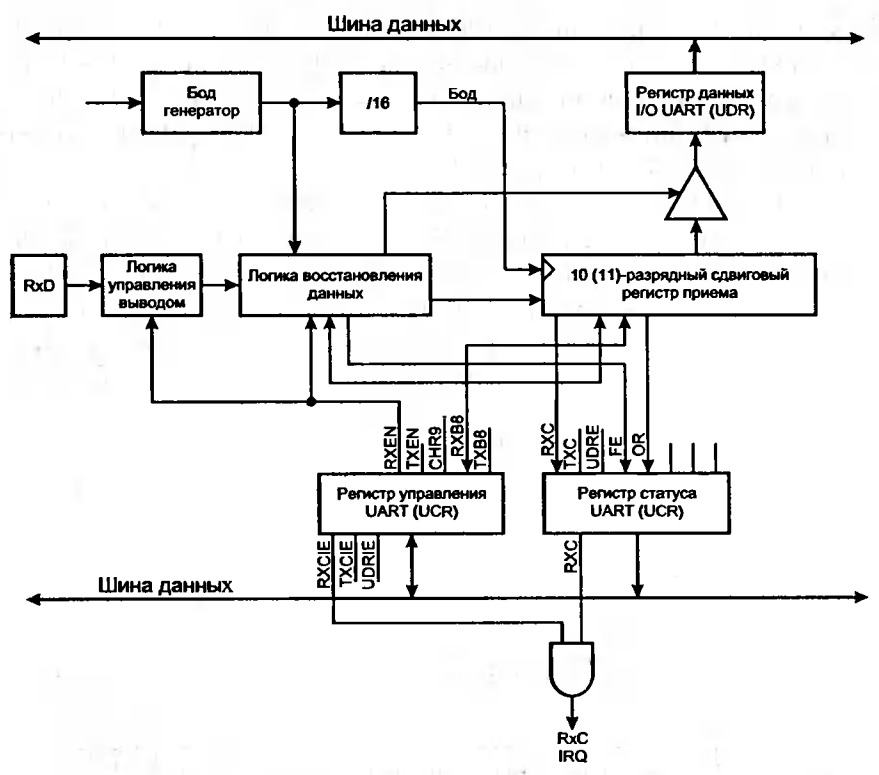


Рис. 2.40. Структурная схема узла приема данных UART

Если обнаружен стартовый бит, начинается обработка битов данных. Решение об уровне данных также производится по 8, 9 и 10 выборкам входного сигнала, уровень входного сигнала определяется по равенству двух выборок. После того как уровень данных определен, данные вдвигаются в сдвиговый регистр приемника.

Для определения стопового бита хотя бы две из трех выборок входного сигнала должны быть равны 1. Если это условие не выполняется, в регистре USR устанавливается флаг ошибки кадра FE. Перед чтением данных из регистра UDR пользователь должен проверять бит FE для обнаружения ошибок кадра.

Независимо от принятия правильного стопового бита по окончании приема символа принятые данные переписываются в UDR и устанавливается флаг RXC в регистре USR. Физически регистр UDR

состоит из двух отдельных регистров, один используется для передачи данных, другой — для приема. При чтении UDR происходит доступ к регистру приемника, при записи — к регистру передатчика. При обмене 9-битовыми данными 9-й бит принятых данных записывается в бит RXB8 регистра UCR.

Если при приеме символа из регистра UDR не был прочитан предыдущий символ, в регистре UCR устанавливается флаг переполнения — OR. Установка этого бита означает, что последний принятый байт данных не переписывается из сдвигового регистра в регистр UDR и будет потерян. Бит OR буферирован и обновляется при чтении правильных данных из UDR. Таким образом, пользователь всегда может проверить состояние OR после чтения UDR и обнаружить происшедшее переполнение.

При сбросе бита RXEN в регистре UCR прием данных запрещается. При этом вывод PD0 можно использовать для ввода/вывода общего назначения. При установке RXEN приемник подключен к выводу PD0 независимо от состояния бита DDD0 в регистре DDRD.

### Управление UART

Бит	7	6	5	4	3	2	1	0	
\$0C (\$2C)	MSB							LSB	UDR
Чт./зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.41. Регистр ввода/вывода UART (UDR)

Бит	7	6	5	4	3	2	1	0	
\$0B (\$2B)	RXC	TXC	UDRE	FE	OR	—	—	—	USR
Чт./зап.	R	R/W	R	R	R	R	R	R	
Нач. знач.	0	0	1	0	0	0	0	0	

Рис. 2.42. Регистр состояния UART (USR)

Регистр USR доступен только для чтения, в нем хранится информация о состоянии UART.

**Бит 7 — RXC** — прием завершен. Этот бит устанавливается в 1, когда принятый символ переписывается из сдвигового регистра приемника в регистр UDR. Бит устанавливается независимо от обнаружения ошибки кадра. Если установлен бит RXCIE в регистре UCR, при установке бита выполняется прерывание по завершении приема

символа. RXC сбрасывается при чтении UDR. При использовании приема данных по прерыванию обработчик прерывания должен читать регистр UDR для сброса RXC, иначе при выходе из прерывания оно будет вызвано снова.

**Бит 6 — TXC** — передача завершена. Этот бит устанавливается в 1, если символ из сдвигового регистра передатчика (включая стоповый бит) передан, а в регистр UDR не были записаны новые данные. Этот флаг особенно полезен при полудуплексной связи, когда передающее устройство должно перейти в режим приема и освободить линию связи сразу по окончании передачи. Если установлен бит TXIE в регистре UCR, при установке TXC выполняется прерывания по окончании передачи. TXC сбрасывается аппаратно при выполнении соответствующего вектора прерывания. Кроме того, бит можно сбросить, записав в него 1.

**Бит 5 — UDRE** — регистр данных UART пуст. Этот бит устанавливается в 1, когда данные, записанные в UDR, переписываются в регистр сдвига передатчика. Установка этого бита означает, что передатчик готов принять следующий символ для передачи. Если установлен бит UDRIE в регистре UCR, при установке этого бита выполняется прерывание окончания передачи. Бит UDRE сбрасывается при записи регистра UDR. При использовании передачи, управляемой прерыванием, подпрограмма обслуживания прерывания должна записывать UDR, чтобы сбросить бит UDRE, иначе при выходе из прерывания оно будет вызвано снова. При сбросе этот бит устанавливается в 1, чтобы показать готовность передатчика.

**Бит 4 — FE** — ошибка кадра. Этот бит устанавливается при обнаружении условия ошибки кадра, т. е. если стоповый бит принятого байта равен 0. Бит FE сбрасывается при приеме единичного стопового бита.

**Бит 3 — OR** — переполнение. Этот бит устанавливается при обнаружении условия переполнения, т. е. в том случае, когда символ из регистра UDR не был прочитан до того, как заполнился сдвиговый регистр приемника. Этот бит буферизован, т. е. остается установленным до тех пор, пока из регистра UDR не будут прочитаны правильные данные. Бит OR сбрасывается, когда принятые данные переписываются в UDR.

**Биты 2...0** — в AT90S2313 зарезервированы и всегда читаются как 0.

Бит	7	6	5	4	3	2	1	0	
\$0A (\$2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UCR
Чт./зап.	R/W	R/W	R/W	R/W	R/W	R/W	R	W	
Нач. знач.	0	0	0	0	0	0	1	0	

Рис. 2.43. Регистр управления UART (UCR)

**Бит 7 — RXCIE** — разрешение прерывания по окончании приема. Если RXCIE = 1, установка бита RXC в регистре USR приводит к выполнению прерывания по окончании приема (при условии, что прерывания разрешены).

**Бит 6 — TXCIE** — разрешение прерывания по окончании передачи. Если этот бит установлен, установка бита TXC в USR приводит к выполнению прерывания по окончании передачи (при условии, что прерывания разрешены).

**Бит 5 — UDRIE** — прерывание по очистке регистра данных последовательного порта. Если этот бит установлен, установка бита UDRE в USR приводит к выполнению прерывания по очистке регистра данных UART (при условии, что прерывания разрешены).

**Бит 4 — RXEN** — разрешение приемника. При установке этого бита разрешается работа приемника UART. Если приемник выключен, флаги TXC, OR и FE не устанавливаются. Если эти флаги установлены, сброс RXEN не очищает их.

**Бит 3 — TXEN** — разрешение передатчика. При установке этого бита разрешается работа передатчика UART. При запрещении работы передатчика во время передачи символа он продолжает работать, пока не будет очищен сдвиговой регистр и не будет передан символ, помещенный в UDR.

**Бит 2 — CHR9** — 9-битовые послышки. Если этот бит установлен, принимаемые и передаваемые символы имеют длину 9 битов. Для передачи и приема 9-го символа используются биты RXB8 и TXB8 соответственно. 9-й бит можно использовать как дополнительный стоповый бит или как признак четности.

**Бит 1 — RXB8** — 8 битов принимаемых данных. Если установлен бит CHR9, сюда записывается 9-й бит принятых данных.

**Бит 0 — TXB8** — 8 битов передаваемых данных. Если установлен бит CHR9, отсюда берется 9-й бит передаваемых данных.

### Генератор скорости передачи

Генератор скорости передачи — это делитель частоты, который генерирует скорости в соответствии с нижеприведенным выражением:

$$BAUD = F_{ck} / (16 * (UBRR + 1)).$$

Здесь: BAUD — скорость передачи (бод); Fck — частота тактового генератора процессора; UBRR — содержимое регистра скорости передачи UART.

В следующей таблице приведены значения регистра UBRR и процентное отклонение от стандартной скорости передачи для стандартных частот кварцевых генераторов.

Таблица 2.13

Скорость	1 МГц	Ошибки, %	1,8432 МГц	Ошибки, %	2 МГц	Ошибки, %	2,4576 МГц	Ошибки, %
2400	UBRR = 25	0,2	UBRR = 47	0,0	UBRR = 51	0,2	UBRR = 63	0,0
4800	UBRR = 12	0,2	UBRR = 23	0,0	UBRR = 25	0,2	UBRR = 31	0,0
9600	UBRR = 6	7,5	UBRR = 11	0,0	UBRR = 12	0,2	UBRR = 15	0,0
14400	UBRR = 3	7,8	UBRR = 7	0,0	UBRR = 8	3,7	UBRR = 10	3,1
19200	UBRR = 2	7,8	UBRR = 5	0,0	UBRR = 6	7,5	UBRR = 7	0,0
28800	UBRR = 1	7,8	UBRR = 3	0,0	UBRR = 3	7,8	UBRR = 4	6,3
38400	UBRR = 1	22,9	UBRR = 2	0,0	UBRR = 2	7,8	UBRR = 3	0,0
57600	UBRR = 0	7,8	UBRR = 1	0,0	UBRR = 1	7,8	UBRR = 2	12,5
76800	UBRR = 0	22,9	UBRR = 1	33,3	UBRR = 1	22,9	UBRR = 1	0,0
115200	UBRR = 0	84,3	UBRR = 0	0,0	UBRR = 0	7,8	UBRR = 0	25,0

Скорость	3,2768 МГц	Ошибки, %	3,6864 МГц	Ошибки, %	4 МГц	Ошибки, %	4,608 МГц	Ошибки, %
2400	UBRR = 84	0,4	UBRR = 95	0,0	UBRR = 103	0,2	UBRR = 119	0,0
4800	UBRR = 42	0,8	UBRR = 47	0,0	UBRR = 51	0,2	UBRR = 59	0,0
9600	UBRR = 20	1,6	UBRR = 23	0,0	UBRR = 25	0,2	UBRR = 29	0,0
14400	UBRR = 13	1,6	UBRR = 15	0,0	UBRR = 16	2,1	UBRR = 19	0,0
19200	UBRR = 10	3,1	UBRR = 11	0,0	UBRR = 12	0,2	UBRR = 14	0,0
28800	UBRR = 6	1,6	UBRR = 7	0,0	UBRR = 8	3,7	UBRR = 9	0,0
38400	UBRR = 4	6,3	UBRR = 5	0,0	UBRR = 6	7,5	UBRR = 7	6,7
57600	UBRR = 3	12,5	UBRR = 3	0,0	UBRR = 3	7,8	UBRR = 4	0,0
76800	UBRR = 2	12,5	UBRR = 2	0,0	UBRR = 2	7,8	UBRR = 3	6,7
115200	UBRR = 1	12,5	UBRR = 1	0,0	UBRR = 1	7,8	UBRR = 2	20,0

Скорость	7,3728 МГц	Ошиб- ки, %	8 МГц	Ошиб- ки, %	9,216 МГц	Ошиб- ки, %,	11,059 МГц	Ошиб- ки, %
2400	UBRR = 191	0,0	UBRR = 207	0,2	UBRR = 239	0,0	UBRR = 287	0,0
4800	UBRR = 95	0,0	UBRR = 103	0,2	UBRR = 119	0,0	UBRR = 143	0,0
9600	UBRR = 47	0,0	UBRR = 51	0,2	UBRR = 59	0,0	UBRR = 71	0,0
14400	UBRR = 31	0,0	UBRR = 34	0,8	UBRR = 39	0,0	UBRR = 47	0,0
19200	UBRR = 23	0,0	UBRR = 25	0,2	UBRR = 29	0,0	UBRR = 35	0,0
28800	UBRR = 15	0,0	UBRR = 16	2,1	UBRR = 19	0,0	UBRR = 23	0,0
38400	UBRR = 11	0,0	UBRR = 12	0,2	UBRR = 14	0,0	UBRR = 17	0,0
57600	UBRR = 7	0,0	UBRR = 8	3,7	UBRR = 9	0,0	UBRR = 11	0,0
76800	UBRR = 5	0,0	UBRR = 6	7,5	UBRR = 7	6,7	UBRR = 8	0,0
115200	UBRR = 3	0,0	UBRR = 3	7,8	UBRR = 4	0,0	UBRR = 5	0,0

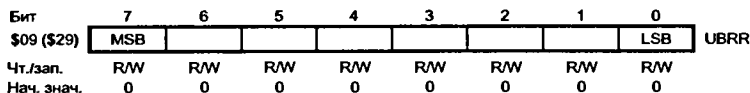


Рис. 2.44. Регистр скорости передачи (UBRR)

Это 8-разрядный регистр, который задает скорость передачи последовательного порта в соответствии с выражением, приведенным выше.

## 2.9. Аналоговый компаратор

Структурная схема аналогового компаратора изображена на рис. 2.45.

Аналоговый компаратор сравнивает входное напряжение на положительном входе PB0 (AIN0) и отрицательном входе PB1 (AIN1). Когда напряжение на положительном входе больше напряжения на отрицательном, устанавливается бит АСО (Analog Comparator Output). Выход аналогового компаратора можно установить на работу с функцией захвата таймера/счетчика1. Кроме того, компаратор может вызывать свое прерывание. Пользователь может установить срабатывание прерывания по нарастающему или спадающему фронту или по переключению.

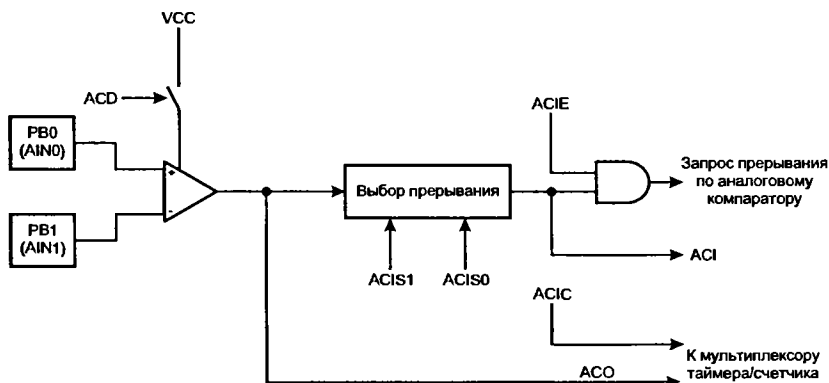


Рис. 2.45. Структурная схема аналогового компаратора

Бит	7	6	5	4	3	2	1	0	
\$08 (\$26)	ACD	—	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Чт./зап.	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	N/A	0	0	0	0	0	

Рис. 2.46. Регистр управления и состояния аналогового компаратора (ACSR)

**Бит 7 — ACD** — запрещение аналогового компаратора. Когда этот бит установлен, питание от аналогового компаратора отключается. Для отключения компаратора этот бит можно установить в любое время. Обычно это свойство используется, если критично потребление процессора в холостом режиме и восстановление работы процессора от аналогового компаратора не требуется. При изменении бита ACD прерывания от аналогового компаратора должны быть запрещены сбросом ACIE в регистре ACSR. В противном случае прерывание может произойти во время изменения бита.

**Бит 6** — в AT90S2313 зарезервирован и всегда читается как 0.

**Бит 5 — ACO** — выход аналогового компаратора. Бит ACO непосредственно «подключен» к выходу аналогового компаратора.

**Бит 4 — ACI** — флаг прерывания от аналогового компаратора. Этот бит устанавливается, когда переключение выхода компаратора совпадает с режимом прерывания, установленным битами ACIS1 и ACIS0. Программа обработки прерывания от аналогового компаратора выполняется, если установлен бит ACIE в 1 и установлен в 1 бит I в регистре состояния. ACI сбрасывается аппаратно при выполнении

соответствующего вектора прерывания. Другой способ очистить ACI — записать во флаг логическую единицу.

**Бит 3 — ACIE** — разрешение прерывания от аналогового компаратора. Когда установлен этот бит и бит I регистра состояния, прерывания от аналогового компаратора отрабатываются. Если бит ACIE = 0, прерывания запрещены.

**Бит 2 — ACIS** — захват по выходе аналогового компаратора. Если этот бит установлен, функция захвата таймера/счетчика1 управляется выходом аналогового компаратора. При этом выход компаратора подключается непосредственно к схеме обработки захвата, предоставляя удобные средства подавления шума и выбора фронта, предусмотренные прерыванием захвата по входу. Когда бит очищен, схема захвата и компаратор разъединены. Чтобы компаратор мог управлять функцией захвата таймера/счетчика1, должен быть установлен бит TICIE1 в регистре TIMCK.

**Биты 1, 0 — ACIS1, ACIS0** — выбор режима прерывания аналогового компаратора. Различные установки приведены в табл. 2.14.

**Таблица 2.14. Установки ACIS1/ACIS0**

ACIS1	ACIS0	Описание
0	0	Прерывание от компаратора по переключении выхода
0	1	Зарезервировано
1	0	Прерывание от компаратора по спадающему фронту выхода
1	1	Прерывание от компаратора по нарастающему фронту выхода

Примечание. При изменении битов ACIS1/ACIS0 прерывания от аналогового компаратора должны быть запрещены сбросом бита разрешения прерывания в регистре ACSR. Иначе прерывание может произойти в процессе изменения этих битов.

## 2.10. Порты ввода/вывода

### Порт В

Порт В 8-разрядный — двунаправленный для ввода/вывода.

Для обслуживания порта отведено три регистра: регистр данных PORTB (\$18, \$38), регистр направления данных — DDRB (\$17, \$37) и выводы порта В (\$16, \$36). Адрес выводов порта В предназначен

только для чтения, в то время как регистр данных и регистр направления данных — для чтения и записи.

Все выводы порта имеют отдельно подключаемые подтягивающие резисторы. Выходы порта В могут поглощать ток до 20 мА и непосредственно управлять светодиодными индикаторами. Если выводы PB0...PB7 используются как входы и замыкаются на землю, то при включенных внутренних подтягивающих резисторов выводы являются источниками тока. Дополнительные функции выводов порта В приведены в табл. 2.15.

Таблица 2.15. Альтернативные функции выводов порта В

Вывод	Альтернативная функция
PB0	AIN0 (положительный вход аналогового компаратора)
PB1	AIN1 (отрицательный вход аналогового компаратора)
PB3	OC1 (выход совпадения таймера/счетчика 1)
PB5	MOSI (вход данных для SPI)
PB6	MISO (выход данных для SPI)
PB7	SCK (вход тактовых импульсов SPI)

При использовании альтернативных функций выводов регистры DDRB и PORTB должны быть установлены в соответствии с описанием альтернативных функций. При использовании возможности внутрисхемного программирования микроконтроллера следует учитывать, что программатор использует для своей работы линии MOSI, MISO и SCK. Соответственно, устройства, подключенные к этим линиям, не должны мешать работе программатора.

Бит	7	6	5	4	3	2	1	0	
\$16 (\$38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Чт./зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.47. Регистр данных порта В — PORTB

Бит	7	6	5	4	3	2	1	0	
\$17 (\$37)	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0	DDRB
Чт./зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.48. Регистр направления данных порта В — DDRB

Бит	7	6	5	4	3	2	1	0	
\$16 (\$36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Чт./зап.	R	R	R	R	R	R	R	R	
Нач. знач.	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Рис. 2.49. Выводы порта В — PINB

PINB не является регистром, по этому адресу осуществляется доступ к физическим значениям каждого из выводов порта В. При чтении PORTB читаются данные из регистра-защелки, при чтении PINB читаются логические значения, соответствующие фактическому состоянию выводов порта.

### Порт В как порт ввода/вывода общего назначения

Все 8 битов порта В при использовании для ввода/вывода одинаковы.

Бит DDB<sub>n</sub> регистра DDRB выбирает направление передачи данных. Если бит установлен (т. е. равен единице), вывод сконфигурирован как выход. Если бит сброшен (т. е. равен нулю) — вывод сконфигурирован как вход. Если PORTB<sub>n</sub> установлен и вывод сконфигурирован как вход, включается КМОП, подтягивающий резистор. Для отключения резистора PORTB<sub>n</sub> должен быть сброшен или вывод должен быть сконфигурирован как выход.

Таблица 2.16. Влияние DDB<sub>n</sub> на выводы порта В

DDB <sub>n</sub>	PORTB <sub>n</sub>	Вход/выход	Подтягивающий резистор	Комментарий
0	0	Вход	Нет	Третье состояние (Hi-Z)
0	1	Вход	Есть	РВ <sub>n</sub> источник тока, если соединен с землей
1	0	Выход	Нет	Выход установлен в 0
1	1	Выход	Нет	Выход установлен в 1

Примечание: n = 7, 6...0 — номер вывода.

Альтернативные функции PORTB:

Бит 7 SCK — вход тактовой частоты для SPI.

Бит 6 MISO — выход данных для SPI.

Бит 5 MOSI — вход данных для SPI.

Бит 3 OC1 — выход совпадения. Этот вывод может быть сконфигурирован для внешнего вывода события совпадения таймера 1. Для этого бит DDB3 должен быть установлен в 1 (вывод сконфигурирован как выход).

Бит 1 AIN1 — отрицательный вход аналогового компаратора. Если этот вывод сконфигурирован как вход ( $DDB1 = 0$ ) и отключен внутренний подтягивающий резистор, этот вывод работает как отрицательный вход внутреннего аналогового компаратора.

Бит 0 AIN0 — положительный вход аналогового компаратора. Если этот вывод сконфигурирован как вход ( $DDB0 = 0$ ) и отключен внутренний подтягивающий резистор, этот вывод работает как положительный вход внутреннего аналогового компаратора.

## Порт D

Для порта D зарезервированы 3 ячейки памяти: регистр PORTD \$12 (\$32), регистр направления данных — DDRD \$11 (\$31) и выводы порта D — PIND \$10 (\$30). Регистры данных и направления данных могут читаться/записываться, ячейка PIND — только для чтения.

Порт D — 7-разрядный двунаправленный со встроенными подтягивающими регистрами. Выходные буферы порта могут поглощать ток до 20 мА. Если выводы используются как входы и на них подан низкий уровень, то при подключении подтягивающих резисторов они являются источниками тока. Некоторые из выводов порта имеют альтернативные функции, показанные в табл. 2.17.

Если выводы порта используются для обслуживания альтернативных функций, они должны быть сконфигурированы на ввод/вывод в соответствии с описанием функций.

**Таблица 2.17. Альтернативные функции порта D**

Вывод	Альтернативная функция
PD0	RXD (вход данных UART)
PD1	TXD (выход данных UART)
PD2	INT0 (вход внешнего прерывания 0)
PD3	INT1 (вход внешнего прерывания 1)
PD4	T0 (внешний вход таймера/счетчика 0)
PD5	T1 (внешний вход таймера/счетчика 1)
PD6	ICP (вход захвата таймера/счетчика 1)

Бит	7	6	5	4	3	2	1	0	
\$12 (\$32)	—	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Чт./зап.	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.50. Регистр данных порта D — PORTD

Бит	7	6	5	4	3	2	1	0	
\$11 (\$31)	—	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Чт./зап.	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. знач.	0	0	0	0	0	0	0	0	

Рис. 2.51. Регистр направления данных порта B — DDRB

Бит	7	6	5	4	3	2	1	0	
\$10 (\$30)	—	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Чт./зап.	R	R	R	R	R	R	R	R	
Нач. знач.	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Рис. 2.52. Выводы порта B — PINB

PIND не является регистром, по этому адресу осуществляется доступ к физическим значениям каждого из выводов порта D. При чтении PORTD читаются данные из регистра-защелки, при чтении PIND читаются логические значения, соответствующие фактическому состоянию выводов порта.

### Порт D как порт ввода/вывода общего назначения

Бит DDDn регистра DDRD выбирает направление передачи данных. Если бит установлен, вывод сконфигурирован как выход. Если бит сброшен — вывод сконфигурирован как вход. Если PORTDn установлен и вывод сконфигурирован как вход, включается КМОП подтягивающий резистор. Для отключения резистора PORTDn должен быть сброшен или вывод должен быть сконфигурирован как выход.

Таблица 2.18. Влияние DDDn на выводы порта D

DDDn	PORTDn	Вход/выход	Подтягивающий резистор	Комментарий
0	0	Вход	Нет	Третье состояние (Hi-Z)
0	1	Вход	Да	PВп источник тока, если соединен с землей
1	0	Выход	Нет	Выход установлен в 0
1	1	Выход	Нет	Выход установлен в 1

Примечание: n = 6...0 — номер вывода.

**Альтернативные функции порта D**

- Бит 6 ICP — вход захвата таймера/счетчика 1. Подробнее см. описание таймера.
- Бит 5 T1 — тактовый вход таймера/счетчика 1. Подробнее см. описание таймера.
- Бит 4 T0 — тактовый вход таймера/счетчика 0. Подробнее см. описание таймера.
- Бит 3 INT1 — вход внешних прерываний 1. Подробнее см. описание прерываний.
- Бит 2 INT0 — вход внешних прерываний 0. Подробнее см. описание прерываний.
- Бит 1 TXD — выход передатчика UART. Если разрешена работа передатчика UART, независимо от состояния DDRD1 этот вывод сконфигурирован как выход.
- Бит 0 RXD — выход приемника UART. Если разрешена работа приемника UART, независимо от состояния DDRD0 этот вывод сконфигурирован как выход. Когда UART использует вывод для приема данных, единица в PORTD0 подключает встроенный подтягивающий резистор.

**2.11. Программирование памяти**

**Программирование битов блокировки памяти**

Микроконтроллер AT90S2313 имеет два бита блокировки, которые могут быть оставлены незапрограммированными или программироваться, при этом достигаются свойства, приведенные в табл. 2.19.

**Таблица 2.19. Биты блокировки**

Режим	Биты блокировки		Тип защиты
	LB1	LB2	
1	1	1	Защита не установлена
2	0	1	Дальнейшее программирование Флэш-памяти и EEPROM запрещено
3	0	0	Как режим 2, но запрещено и чтение

Примечание: биты блокировки стираются только при полном стирании памяти.

## Биты конфигурации (Fuse bits)

В AT90S2313 предусмотрено два бита конфигурации — SPIEN и FSTRT. Когда бит SPIEN = 0, разрешен режим последовательного программирования. По умолчанию бит SPIEN = 0.

Когда бит FSTRT = 0, используется укороченное время запуска. По умолчанию этот бит FSTRT = 1. Можно заказывать микросхемы с предварительно запрограммированным битом.

Эти биты недоступны при последовательном программировании и не изменяются при стирании памяти.

## Код устройства

Все микроконтроллеры фирмы Atmel имеют 3-байтовый сигнатурный код, по которому идентифицируется устройство. Этот код можно прочитать в параллельном и последовательном режимах. Это три байта, размещенные в отдельном адресном пространстве, и для AT90S2313 имеют следующие значения:

1. \$000: \$1E — код производителя — Atmel.
2. \$001: \$91 — 2 кбайта флэш-памяти.
3. \$002: \$01 — при \$01=\$91 — микроконтроллер AT90S2313.

Если запрограммированы биты блокировки, байты сигнатуры в последовательном режиме не читаются.

## 2.12. Параметры микроконтроллера AT90S2313

Для правильного применения любой микросхемы, а особенно такой сложной, как микроконтроллер, имеет смысл при принятии решения о том или ином варианте схемы уточнить некоторые справочные данные. Чтобы у читателей была такая возможность, автор приводит здесь подробные технические данные на микроконтроллер AT90S2313. Параметры других микроконтроллеров этой серии очень близки к указанным.

### Максимально допустимые параметры

Рабочая температура	от $-55$ до $+125$ °C
Температура хранения	от $-65$ до $+150$ °C
Напряжение на любом выводе, кроме RESET	от $-1,0$ до $+7,0$ В
Максимальное рабочее напряжение	6,6 В
Постоянный ток через вывод порта	40,0 мА
Постоянный ток между VCC и GND	140,0 мА
Примечание. Выход параметров за пределы, указанные в таблице, может привести к нарушению работоспособности микросхемы. Это предельные значения параметров, рабочие параметры микросхемы приведены ниже. Удержание предельных значений на выводах микроконтроллера в течение длительного времени может привести к потере его работоспособности.	

### Характеристики по постоянному току

Обозн.	Параметр	Условия	Мин.	Тип.	Макс.	Ед.
VIL	Входное напряжение низкого уровня	(Кроме XTAL1)	$-0,5$	—	$0,3V_{CC}$ (1)	В
vil1	Входное напряжение низкого уровня	(XTAL1)	$-0,5$	—	$0,3V_{CC}$ (1)	В
vih	Входное напряжение высокого уровня	(Кроме XTAL1, RESET)	$0,6 V_{CC}$ (2)	—	$V_{CC}+0,5$	В
VIH1	Входное напряжение высокого уровня	(XTAL1)	$0,7 V_{CC}$ (2)	—	$V_{CC}+0,5$	В
VIH2	Входное напряжение высокого уровня	(RESET)	$0,85 V_{CC}$ (2)	—	$V_{CC}+0,5$	В
vol	Выходное напряжение низкого уровня (порты В, D)	$I_{ol} = 20$ мА, $V_{CC} = 5$ В $I_{ol} = 10$ мА, $V_{CC} = 3$ В	—	—	0,6 0,5	В
voh	Выходное напряжение высокого уровня (порты В, D)	$I_{oh} = -3$ мА, $V_{CC} = 5$ В $I_{oh} = -1,5$ мА, $V_{CC} = 3$ В	4,3 2,3	—	—	В
IIL	Входной ток утечки вывода I/O	$V_{CC} = 6$ В, низкий уровень	—	—	1,5	мкА
IIH	Входной ток утечки вывода I/O	Высокий уровень	—	—	980,0	нА
RRST	Подтягивающий резистор Reset		100,0	—	500,0	кОм

## Характеристики по постоянному току (продолжение)

Обозн.	Параметр	Условия	Мин.	Тип.	Макс.	Ед.
$r_i/o$	Подтягивающие резисторы линий ввода/вывода		35,0	—	120,0	кОм
ICC	Ток потребления от источника питания	Нормальный режим, $V_{CC} = 3 \text{ В}, 4 \text{ МГц}$	—	—	3,0	мА
		Idle режим $V_{CC} = 3 \text{ В}, 4 \text{ МГц}$			1,0	мА
ICC	Power-down Режим	Сторожевой таймер включен, $V_{CC} = 3 \text{ В}$	—	9,0	15,0	мкА
		Сторожевой таймер отключен, $V_{CC} = 3 \text{ В}$	—	< 1,0	2,0	мкА
VACIO	Напряжение смещения входа аналогового компаратора	$V_{CC} = 5 \text{ В}$ $V_{IN} = V_{CC}/2$	—	—	40,0	мВ
IACLK	Утечка по входу аналогового компаратора	$V_{CC} = 5 \text{ В}$ $V_{IN} = V_{CC}/2$	—50,0	—	50,0	нА
IACPD	Задержка аналогового компаратора	$V_{CC} = 2,7 \text{ В}$ $V_{CC} = 4,0 \text{ В}$	—	750,0 500,0	—	нс

## Примечания.

- В рабочем состоянии ток через выводы должен ограничиваться следующими условиями:  
максимальный ток через вывод — 20 мА — при состоянии 0!  
максимальный ток через все выводы — 80 мА.
- Минимальное напряжение для режима пониженного потребления — 2 В.

## Параметры внешнего тактового сигнала

Параметр	$V_{CC} = 2,7...6,0 \text{ В}$		$V_{CC} = 4,0...6,0 \text{ В}$		Ед. изм.
	min	max	min	max	
Частота кварца	0	4	0	10, МГц	
Период тактовой частоты	250		100		нс
Длительность «1»	100		40		нс
Длительность «0»	100		40		нс
Длительность фронта		1,6		0,5	
Длительность спада		1,6		0,5	мкс

### Варианты исполнения микроконтроллера

Частота	Напряжение питания	Маркировка	Корпус	Диапазон температур
4 МГц	2,7...6,0 В	AT90S2313-4PC	PDIP	Коммерческий (0...70 °C)
		AT90S2313-4SC	SOIC	
		AT90S2313-4PI	PDIP	Промышленный (-40...85 °C)
		AT90S2313-4SI	SOIC	
10 МГц	4,0...6,0 В	AT90S2313-10PC	PDIP	Коммерческий (0...70 °C)
		AT90S2313-10SC	SOIC	
		AT90S2313-10PI	PDIP	Промышленный (-40...85 °C)
		AT90S2313-10SI	SOIC	

## 2.13. Набор команд AT90S2313

### Арифметические и логические команды

Мнемоника	Описание	Действие	Флаги	Циклы
ADD Rd,Rr	Сложить два регистра	$Rd < Rd + Rr$	Z,C,N,V,H	1
ADC Rd,Rr	Сложить с переносом	$Rd < Rd + Rr + C$	Z,C,N,V,H	1
ADIW RdI,K	Сложить слово с константой	$Rdh, l < Rdh, l + K$	Z,C,N,V,S	2
SUB Rd,Rr	Вычесть два регистра	$Rd < Rd - Rr$	Z,C,N,V,H	1
SUBI Rd,K	Вычесть константу	$Rd < Rd - K$	Z,C,N,V,H	1
SBIW RdI,K	Вычесть слово с константой	$Rdh, l < Rdh, l - K$	Z,C,N,V,S	2
SBC Rd,Rr	Вычесть с переносом	$Rd < Rd - Rr - C$	Z,C,N,V,H	1
SBCI Rd,K	Вычесть с переносом	$Rd < Rd - K - C$	Z,C,N,V,H	1
AND Rd,Rr	Логическое И	$Rd < Rd \text{ AND } Rr$	Z,N,V	1
ANDI Rd,K	Логическое И	$Rd < Rd \text{ AND } K$	Z,N,V	1
OR Rd,Rr	Логическое ИЛИ	$Rd < Rd \text{ OR } Rr$	Z,N,V	1
ORI Rd,K	Логическое ИЛИ	$Rd < Rd \text{ OR } K$	Z,N,V	1
EOR Rd,Rr	Исключающее ИЛИ	$Rd < Rd \text{ XOR } Rr$	Z,N,V	1
COM Rd	Дополнение до 1	$Rd < \$FF - Rd$	Z,C,N,V	1

## Арифметические и логические команды (продолжение)

Мнемоника	Описание	Действие	Флаги	Циклы
NEG Rd	Дополнение до 2	$Rd < \$00 - Rd$	Z,C,N,V,H	1
SBR Rd,K	Установить бит(ы) в регистре	$Rd < Rd \text{ OR } K$	Z,N,V	1
CBR Rd,K	Сбросить бит(ы) в регистре	$Rd < Rd \text{ AND } (\text{FFh}-K)$	Z,N,V	1
INC Rd	Увеличить на 1	$Rd < Rd + 1$	Z,N,V	1
DEC Rd	Уменьшить на 1	$Rd < Rd - 1$	Z,N,V	1
TST Rd	Проверить на 0 или	$Rd < Rd \text{ AND } Rd$	Z,N,V	1
CLR Rd	Очистить регистр	$Rd < Rd \text{ XOR } Rd$	Z,N,V	1
SER Rd	Установить регистр	$Rd < \$FF$	None	1

## Команды ветвления

Мнемоника	Описание	Действие	Флаги	Циклы
RJMP k	Относительный переход	$PC < PC + k + 1$	None	2
LJMP	Переход по адресу (Z)	$PC < Z$	None	2
RCALL k	Относительный вызов подпрограммы	$PC < PC + k + 1$	None	3
ICALL	Вызов подпрограммы по адресу (Z)	$PC < Z$	None	3
RET	Выход из подпрограммы	$PC < \text{STACK}$	None	4
RETI	Выход из прерывания	$PC < \text{STACK}$	I <sup>1</sup>	4
CPSE Rd,Rr	Сравнить, пропуск, если равно	$\text{if}(Rd=Rr) \text{ PC} < PC + 2 \text{ или } 3$	None	1/2
CP Rd,Rr	Сравнить	$Rd - Rr$	Z,N,V,C,H	1
CPC Rd,Rr	Сравнить с переносом	$Rd - Rr - C$	Z,N,V,C,H	1
CPI Rd,K	Сравнить с константой	$Rd - K$	Z,N,V,C,H	1
SBRC Rr,b	Пропуск если бит в регистре сброшен	$\text{if}(Rr(b)=0) \text{ PC} < PC + 2 \text{ или } 3$	None	1/2
SBRS Rr,b	Пропуск, если бит в регистре установлен	$\text{if}(Rr(b)=1) \text{ PC} < PC + 2 \text{ или } 3$	None	1/2
SBIC P, b	Пропуск, если бит в регистре ввода/вывода сброшен	$\text{if}(P(b)=0) \text{ PC} < PC + 2 \text{ или } 3$	None	1/2

## Команды ветвления (продолжение)

Мнемоника	Описание	Действие	Флаги	Циклы
SBIS P, b	Пропуск, если бит в регистре ввода/вывода установлен	if(P(b)=1) PC<PC+2 или 3	None	1/2
BRBS s, k	Переход, если установлен флаг s	if(SREG(s)=1) PC<PC+k+1	None	
BRBC s, k	Переход, если сброшен флаг s	if(SREG(s)=0) PC<PC+k+1	None	1/2
BREQ k	Переход, если равно	if(Z=1) PC<PC+k+1	None	1/2
BRNE k	Переход, если неравно	if(Z=0) PC<PC+k+1	None	1/2
BRCS k	Переход, если установлен перенос	if(C=1) PC<PC+k+1	None	1/2
BRCC k	Переход, если сброшен перенос	if(C=0) PC<PC+k+1	None	1/2
BRSH k	Переход, если равно или больше	if(C=0) PC<PC+k+1	None	1/2
BRLO k	Переход, если меньше	if(C=1) PC<PC+k+1	None	1/2
BRMI k	Переход, если минус	if(N=1) PC<PC+k+1	None	1/2
BRPL k	Переход, если плюс	if(N=0) PC<PC+k+1	None	1/2
BRGE k	Переход, если больше или равно со знаком	if(N XOR V=0) PC<PC+k+1	None	1/2
BRLT k	Переход, если меньше нуля со знаком	if(N XOR V=1) PC<PC+k+1	None	1/2
BRHS k	Переход, если установлен флаг H	if (H=1) PC<PC+k+1	None	1/2
BRHC k	Переход, если сброшен флаг H	if (H=0) PC<PC+k+1	None	1/2
BRTS k	Переход, если установлен флаг T	if (H=1) PC<PC+k+1	None	1/2
BRTC k	Переход, если сброшен флаг T	if (H=0) PC<PC+k+1	None	1/2
BRVS k	Переход, если установлен флаг V (переполнение)	if (H=1) PC<PC+k+1	None	1/2
BRVC k	Переход, если сброшен флаг V	if (H=0) PC<PC+k+1	None	1/2
BRIE k	Переход, если разрешены прерывания	if(I=1) PC<PC+k+1	None	1/2
BRID k	Переход, если запрещены прерывания	if(I=1) PC<PC+k+1	None	1/2

## Команды пересылки

Мнемоника	Описание	Действие	Флаги	Циклы
MOV Rd,Rr	Пересылка между регистрами	$Rd < Rr$	None	1
LDI Rd,K	Загрузить константу	$Rd < K$	None	1
LD Rd,X	Загрузить регистр непосредственно	$Rd < (X)$	None	2
LD Rd,X+	Загрузить регистр непосредственно с постинкрементом	$Rd < (X), X < X+1$	None	2
LD Rd,-X	Загрузить регистр непосредственно с предварительным декрементом	$X < X-1, Rd < (X)$	None	2
LD Rd,Y	Загрузить регистр непосредственно	$Rd < (Y)$	None	2
LD Rd,Y+	Загрузить регистр непосредственно с постинкрементом	$Rd < (Y), Y < Y+1$	None	2
LD Rd,-Y	Загрузить регистр непосредственно с предварительным декрементом	$Y < Y-1, Rd < (Y)$	None	2
LDD Rd,Y+q	Загрузить регистр непосредственно со смещением	$Rd < (Y+q)$	None	2
LD Rd,Z	Загрузить регистр непосредственно	$Rd < (Z)$	None	2
LD Rd,Z+	Загрузить регистр непосредственно с постинкрементом	$Rd < (Z), Z < Z+1$	None	2
LD Rd,-Z	Загрузить регистр непосредственно с предварительным декрементом	$Z < Z-1, Rd < (Z)$	None	2
LDD Rd,Z+q	Загрузить регистр непосредственно со смещением	$Rd < (Z+q)$	None	2
LDS Rd,k	Загрузить из ОЗУ	$Rd < (k)$	None	3
ST X,Rr	Записать регистр непосредственно	$(X) < Rr$	None	2
ST X+,Rr	Записать регистр непосредственно с постинкрементом	$(X) < Rr, X < X+1$	None	2
ST -X,Rr	Записать регистр непосредственно с предварительным декрементом	$X < X-1, (X) < Rr$	None	2
ST Y,Rr	Записать регистр непосредственно	$(Y) < Rr$	None	2
ST Y+,Rr	Записать регистр непосредственно с постинкрементом	$(Y) < Rr, Y < Y+1$	None	2
ST -Y,Rr	Записать регистр непосредственно с пред. декрементом	$Y < Y-1, (Y) < Rr$	None	2

## Команды пересылки (продолжение)

Мнемоника	Описание	Действие	Флаги	Циклы
STD Y+q,Rr	Записать регистр непосредственно со смещением	$(Y+q)<Rr$	None	2
ST Z,Rr	Записать регистр непосредственно	$(Z)<Rr$	None	2
ST Z+,Rr	Записать регистр непосредственно с постинкрементом	$(Z)<Rr, Z<Z+1$	None	2
ST -Z,Rr	Записать регистр непосредственно с предварительным декрементом	$Z<Z-1, (Z)<Rr$	None	2
STD Y+q,Rr	Запись регистра непосредственно со смещением	$(Y+q)<Rr$	None	2
STS k,Rr	Записать в ОЗУ	$(k)<Rr$	None	3
LPM	Загрузка из памяти программ	$R0<(Z)$	None	3
IN Rd, P	Ввод из порта	$Rd<P$	None	1
OUT P, Rr	Вывод в порт	$P<Rr$	None	1
PUSH Rr	Записать в стек	$STACK<Rr$	None	2
POP Rr	Прочитать из стека	$Rr<STACK$	None	2

## Команды работы с битами

Мнемоника	Описание	Действие	Флаги	Циклы
SBI P,b	Установить бит в регистре ввода/вывода	$I/O(P,b)<1$	None	2
CBI P,b	Сбросить бит в регистре ввода/вывода	$I/O(P,b)<0$	None	2
LSL Rd	Логический сдвиг влево	$Rd(n+1)<Rd(n), Rd(0)<0$	Z,C,N,V	1
LSR Rd	Логический сдвиг вправо	$Rd(n)<Rd(n+1), Rd(7)<0$	Z,C,N,V	1
ROL Rd	Сдвиг влево через C	$Rd(0)<C, Rd(n+1)<Rd(n), C<Rd(7)$	Z,C,N,V	1
ROR Rd	Сдвиг вправо через C	$Rd(7)<C, Rd(n)<Rd(n+1), C<Rd(0)$	Z,C,N,V	1
ASR Rd	Арифметический сдвиг вправо	$Rd(n)<Rd(n+1), n=0...6$	Z,C,N,V	1

## Команды работы с битами (продолжение)

Мнемоника	Описание	Действие	Флаги	Циклы
SWAP Rd	Обмен байтлов	$Rd(3-0) < Rd(7-4)$ $Rd(7-4) < Rd(3-0)$	None	1
BSET s	Установить флаг	$SREG(s) < 1$	SREG(s)	1
BCLR s	Сбросить флаг	$SREG(s) < 0$	SREG(s)	1
BST Rr,b	Запомнить бит в T	$T < Rr(b)$	T	1
BLD Rd,b	Прочитать бит из T	$Rd(b) < T$	None	1
SEC	Установить перенос	$C < 1$	C	1
CLC	Сбросить перенос	$C < 0$	C	1
SEN	Установить флаг N	$N < 1$	N	1
CLN	Сбросить флаг N	$N < 0$	N	1
SEZ	Установить флаг Z	$Z < 1$	Z	1
CLZ	Сбросить флаг Z	$Z < 0$	Z	1
SEI	Разрешить прерывания	$I < 1$	I	1
CLI	Запретить прерывания	$I < 0$	I	1
SES	Установить флаг S	$S < 1$	S	1
CLS	Сбросить флаг S	$S < 0$	S	1
SEV	Установить флаг V	$V < 1$	V	1
CLV	Сбросить флаг V	$V < 0$	V	1
SET	Установить флаг T	$T < 1$	T	1
CLT	Сбросить флаг T	$T < 0$	T	1
SEN	Установить флаг H	$H < 1$	H	1
CLH	Сбросить флаг H	$H < 0$	H	1
NOP	Нет операции		None	1
SLEEP	Останов		None	3
WDR	Сброс сторожевого таймера		None	1

## Глава 3.

# Особенности использования микроконтроллеров семейства AVR

Какая самая простая работающая схема на микроконтроллере? Микроконтроллер, источник питания, элементы, обеспечивающие работу тактового генератора. На рис. 3.1 показана такая схема.

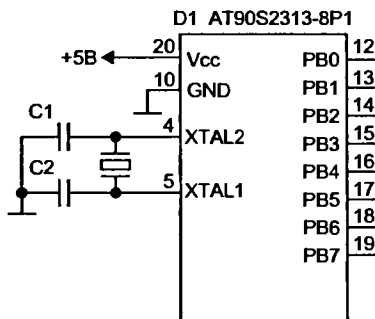


Рис. 3.1. Простейшая схема на микроконтроллере

### 3.1. Источник питания

Источник питания схемы на микроконтроллерах — очень важный узел. Ни одна схема не может работать без источника питания. На принятие решения о выборе типа источника питания влияет множество факторов, в зависимости от создаваемой конструкции. В первую очередь, устройство может питаться от батарей или от сети переменного тока 220 В.

В некоторых случаях даже тогда, когда рядом есть сеть переменного тока, имеет смысл применять батарейное питание — если потребление устройства очень мало и батарей хватит на длительный срок работы. Это позволяет сделать схему проще и легче (не нужен сетевой трансформатор и т. д.).

## Батарейное питание

Выпускается много видов батарей. Они бывают различных форм и размеров. При выборе батареи для питания своей схемы следует руководствоваться следующими параметрами.

1. Емкость батареи — это очень важный параметр, измеряется он в мА·ч или А·ч (миллиампер-час или ампер-час). Он определяет длительность работы конструкции без замены батарей. Чем емкость больше, тем дольше будет работать схема. Обычно чем больше емкость батареи, тем больше ее размеры и масса.

2. Напряжение батареи.

3. Срок хранения батареи.

4. Рабочая температура батарей. Особенно важно учитывать влияние температуры в том случае, если конструкция должна работать при низких или высоких температурах (например, на улице).

5. Размеры, форма и вес батареи.

## Питание от сети

Для питания от сети можно применить обыкновенные стабилизаторы напряжения например КР142ЕН5А (аналог зарубежной микросхемы 7805). Эта микросхема позволяет получить стабилизированное напряжение величиной 5 В при токе примерно до 1 А. Если такой большой ток не требуется, можно применить стабилизатор типа 78L05, позволяющий получить ток до 100 мА. Вообще стабилизаторы могут быть самыми различными, главное, чтобы они обеспечивали нужное напряжение и величину тока для работы схемы.

Перспективный вариант построения сетевых источников питания — импульсные преобразователи напряжения. Они позволяют ощутимо снизить массу источника питания и увеличить его КПД. Но это отдельная большая тема, и в этой книге она рассматриваться не будет.

## Питание от линий портов

В некоторых случаях, если схема потребляет очень маленький ток, возможно питание ее от информационных линий портов. Например, если устройство подключается к порту LPT1 и использует только 3 линии, остальные линии, работающие на вывод, могут использо-

ваться как источник питания (естественно, для этого на них нужно вывести значение 1).

Аналогично можно использовать линию RTS порта RS-232. Для ее использования следует собрать простейший параметрический стабилизатор напряжения на стабилитроне. Параллельно стабилитрону следует подсоединить конденсатор емкостью не менее 10 мкФ.

**Внимание!** Принципиальный момент — независимо от типа источника питания как можно ближе к выводам питания микроконтроллера следует подсоединить керамический (еще лучше — танталовый) конденсатор емкостью 0,01...0,1 мкФ — для подавления помех по цепи питания. Кстати, такой конденсатор следует ставить рядом с каждой цифровой микросхемой.

## 3.2. Внешние элементы тактового генератора

### Использование кварцевого резонатора

Использование кварцевого резонатора наиболее распространенный способ включения внешней схемы тактового генератора. Эта схема требует дополнительно два конденсатора емкостью от 22 до 33 пФ, чтобы облегчить запуск тактового генератора.

На рис. 3.2 показана схема подключения кварцевого резонатора к микроконтроллеру.

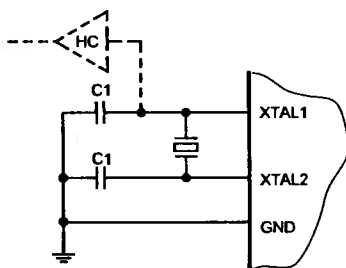


Рис. 3.2. Подключение кварцевого резонатора к микроконтроллеру

### Использование встроенного RC-генератора

В некоторых типах микроконтроллеров AVR имеется встроенный RC-тактовый генератор (например, AT90S1200, AT90S2343, Tiny22). Его включением управляет бит RCEN. В микроконтроллере

AT90S1200 встроенный RC-генератор отключен, его можно включить только пользуясь параллельным программатором. Имеется вариант микроконтроллера AT90S1200A, в котором заранее включен встроенный RC-генератор. Микроконтроллеры AT90S2343 и Tiny22 поставляются с уже включенным RC-генератором.

Следует учитывать, что стабильность частоты RC-генератора намного ниже, чем кварцевого, поэтому не следует применять его в случаях, когда нужно точно отмерять интервалы времени.

### 3.3. Цепь сброса

Используется для задержки запуска микроконтроллера при включении питания, что нужно для его правильного запуска, а также для ручного перезапуска микроконтроллера нажатием на кнопку.

Простейшая схема цепи сброса показана на рис. 3.3.

Если в схеме используется встроенная электрически перепрограммируемая память данных микроконтроллера, следует использовать внешнюю схему, удерживающую вывод Reset на низком уровне до тех пор, пока напряжение не поднимется до необходимого для нормальной работы микроконтроллера уровня. Эта схема должна работать как при увеличении напряжения (т. е. уровень на выводе Reset станет высоким только после достижения напряжением питания нужного уровня), так и при снижении напряжения питания ниже допустимого уровня. Такая схема обычно называется супервайзер питания. На рис. 3.4 показана схема одного из вариантов подключения супервайзера питания.

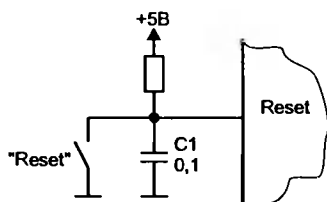


Рис. 3.3. Простейшая цепь сброса

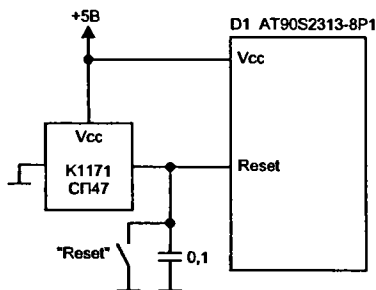


Рис. 3.4. Схема подключения супервайзера питания

# **Глава 4.**

## **Программы и инструменты**

### **4.1. Ассемблер**

Как известно, наиболее эффективные программы получаются при использовании языка Ассемблер. Стоит, правда, отметить, что при этом также увеличивается сложность и время разработки программы. Для микроконтроллеров семейства AVR имеется свободно распространяемый транслятор ассемблера — `wavrasm`. Также одновременно с транслятором ассемблера устанавливается программа для отладки программ на языке ассемблера для микроконтроллеров семейства AVR. Однако она заметно уступает свободно распространяемому фирмой Atmel отладчику AVR Studio, поэтому ее мы рассматривать не будем.

В этой главе читатели познакомятся с описанием транслятора ассемблера `wavrasm`. Этот транслятор преобразует команды в виде текстовых мнемоник (специальных сокращений) в закодированные в виде чисел-кодов команды микроконтроллеров. Полученный код можно использовать для работы совместно с отладчиком для проверки правильности работы программы. Также в результате работы транслятора можно получить так называемый файл прошивки, который используется программатором для занесения программы в память программ микроконтроллера.

Транслятор ассемблера работает в среде Microsoft Windows 3.11, Microsoft Windows 95/98 и NT. Кроме того, имеется версия ассемблера, работающая из командной строки MS-DOS. Она устанавливается одновременно с версией для Windows. Версия для Windows имеет встроенный полноэкранный текстовый редактор и справочную систему на английском языке.

В этом описании отсутствует описание системы команд микроконтроллеров, так как в главе 2 имеется таблица команд, и описаны способы адресации.

Это описание предполагает, что транслятор wavtasm правильно установлен на компьютер, на котором происходит работа.

### Начало работы

Запустите транслятор ассемблера. С помощью команды меню File>>Open откройте файл tutor.asm. В результате выполнения этой команды исходный текст программы на ассемблере будет загружен в программу и показан на экране. На рис. 4.1 показан примерный вид экрана.

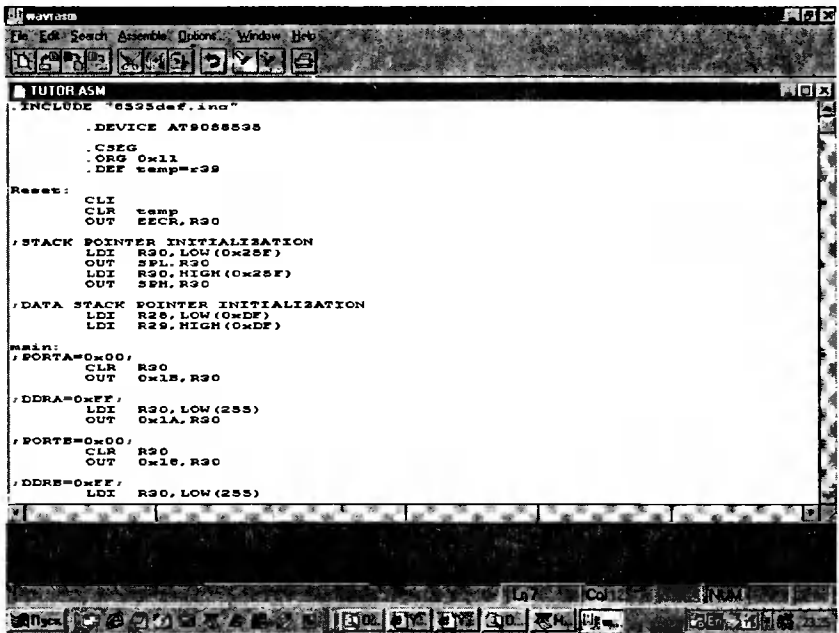


Рис. 4.1. Окно транслятора с загруженной программой

### Ассемблирование первой программы

После того как вы просмотрели текст, выберите команду меню Assemble. После этого появится второе окно (окно сообщений), которое содержит сообщения об ошибках. Окно сообщений будет распо-

лагаться поверх окна с исходным текстом программы на ассемблере, поэтому удобно предпринять некоторые действия по настройке расположения окон. Перейдите в окно с исходным текстом программы (просто щелкнув левой кнопкой мышки в любом месте текста программы) и выберите команду меню Windows>>Tile Horizontal. Кроме того, полезно увеличить размер окна с исходным текстом программы и уменьшить окно сообщений. Для этого следует переместить верх окна сообщений с помощью мышки ниже. В результате на экране должна получиться картинка, похожая на изображенную на рис. 4.2.

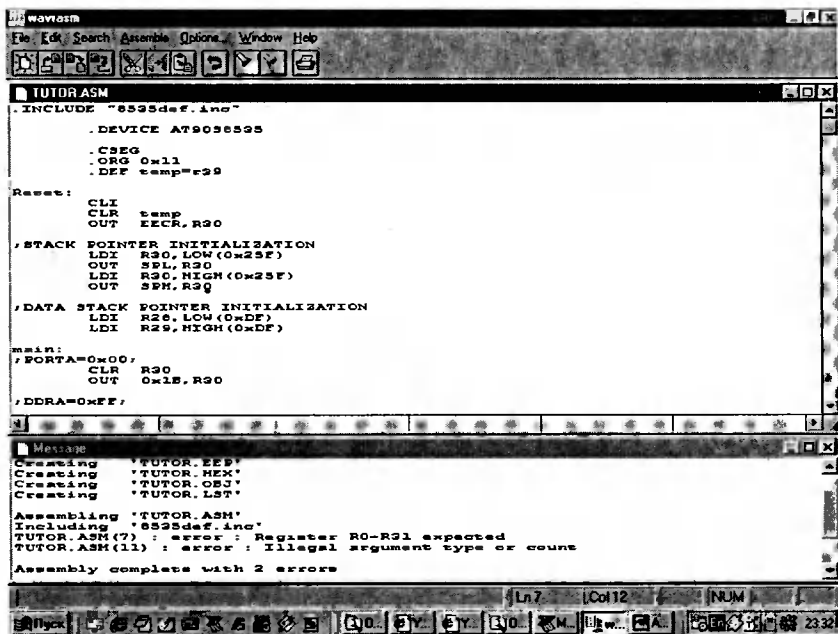


Рис. 4.2. Вид экрана транслятора ассемблера с двумя окнами

## Поиск и исправление ошибок

Глядя на окно сообщений, можно сделать вывод, что в процессе трансляции программы были обнаружены ошибки. В окне сообщений ошибки показываются следующим образом: в строке подряд идут название файла, в скобках номер строки, в которой обнаружена

ошибка, и, наконец, краткое текстовое сообщение о характере ошибки. Естественно, ошибку следует найти и исправить. Щелкните левой кнопкой мышки на первом сообщении об ошибке в окне сообщений (которая находится на 7-й строке). Обратите внимание, что в окне с исходным текстом на строке 7 появится вертикальная красная линия. Сообщение об ошибке говорит о том, что как имена регистров можно использовать только имена r0...r31. Это верно, так как микроконтроллеры семейства AVR имеют 32 регистра общего назначения, а в строке 7 программы указано имя регистра r39, которого не существует. Фотография окна программы с описанной ситуацией показана на рис. 4.3.

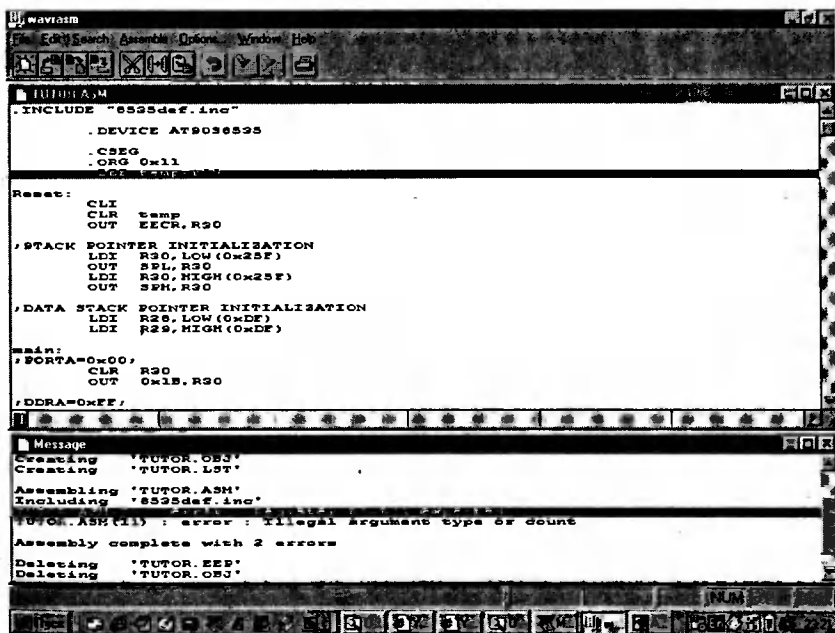


Рис. 4.3. Окно программы с выделенной строкой с ошибкой

Двойной щелчок на сообщении ошибки в окне сообщений приводит к тому, что окно текстового редактора исходного текста становится активным, а курсор устанавливается в начало строки с ошибкой. Исправьте r39 на r19.

Ниже в окне сообщений показано еще сообщение об ошибке. Щелкните левой кнопкой мышки по следующему сообщению об ошибке.

Сообщение *Illegal argument type or count* говорит о том, что что-то неправильно в аргументах команды. Обратите внимание, что один из аргументов — тот самый, который мы только что исправили. Просмотрев все сообщения об ошибках, можно прийти к выводу, что все остальные ошибки были связаны с первой.

Чтобы определить, все ли ошибки исправлены, имеет смысл снова запустить трансляцию программы. Если ошибки остались, следует их исправить. Если ошибок больше нет, в окне сообщений появится сообщение об успешном завершении трансляции.

### Формат программы на ассемблере

Программа на ассемблере представляет собой текстовый файл, который состоит из мнемоник — символьных обозначений команд микроконтроллера, меток и директив.

Любая строка может начинаться с метки — строки из символов и (или) цифр, заканчивающейся двоеточием.

Метки используются для обозначения текущей строки некоторым именем (меткой) для использования в командах условного или безусловного перехода, а также для обозначения участка в памяти для обращения к данным.

Строка исходного текста может иметь один из следующих видов:

1. [метка:] директива [аргументы директивы] [комментарий]
2. [метка:] мнемоника команды [аргументы команды] [комментарий]
3. Комментарий
4. Пустая строка

Комментарии всегда начинаются с символа «;».

Элементы, заключенные в квадратные скобки, могут отсутствовать. Текст, расположенный после символа «точка с запятой» до конца строки, полностью игнорируется ассемблером. Использование ме-

ток, мнемоник команд микроконтроллера и директив ассемблера подробнее будет рассмотрено чуть позднее.

### Примеры записи строк:

```
label1: .EQU var1=100      ; Директива определения символического  
                                ; имени var1, эквивалентного записи "100"  
      .EQU var2=200        ; Определение имени var2, соответствующего "200"  
  
test:  rjmp test           ; Бесконечный цикл (мнемоника команды микроконтроллера)  
                                ; Пустая строка
```

Обратите внимание, что не играет никакой роли, в каких местах расположены метки, команды ассемблера и директивы, важен только их порядок.

## Команды микроконтроллера

Транслятор ассемблера позволяет использовать в тексте программы мнемоники (обозначения команд микроконтроллера), полностью совпадающие с их названием в системе команд микроконтроллера.

Команды микроконтроллеров семейства AVR делятся на несколько групп:

- арифметические и логические;
- команды условных и безусловных переходов;
- команды передачи данных;
- команды для работы с битами.

Краткое описание команд микроконтроллера можно найти в главе 2.

Для транслятора ассемблера нет разницы, какими буквами написаны слова, т. е. `rjmp` и `RJMP` совершенно равнозначны. Однако для удобства понимания программы рекомендуется все мнемоники и метки записывать строчными (маленькими) буквами, а директивы прописными (большими).

## Директивы транслятора ассемблера

Транслятор ассемблера поддерживает достаточно много директив. Директивы не транслируются в программу для микроконтроллера. Они используются для указания транслятору ассемблера данных

о расположении программы в памяти микроконтроллера, определения макросов и т. д.

Ниже приведен перечень директив транслятора ассемблера:

**BYTE** — резервирует 1 байт для использования в качестве переменной;

**CSEG** — сегмент программ;

**DB** — определяет байт-константу;

**DEF** — определяет символическое имя для регистра;

**DEVICE** — задает тип целевого микроконтроллера;

**DSEG** — сегмент данных;

**DW** — определяет слово-константу;

**ENDMACRO** — конец определения макроса;

**EQU** — сопоставляет символному имени арифметическое выражение;

**ESEG** — сегмент EEPROM;

**EXIT** — выйти из файла (конец текста программы);

**INCLUDE** — загрузить исходный текст из другого файла;

**LIST** — включить генерацию листинга;

**LISTMAC** — включить печать содержимого макросов в листинге;

**MACRO** — начать определение макроса;

**NOLIST** — выключить генерацию листинга;

**ORG** — установить расположение;

**SET** — сопоставить символу выражение.

**Обратите внимание, что все директивы должны начинаться с точки.**

**.BYTE** — резервирует место (или несколько мест) размером 1 байт для переменной.

Директива **BYTE** резервирует один байт в памяти SRAM для реализации переменной. Для того чтобы иметь возможность обращаться к этой переменной, перед директивой **BYTE** должна стоять метка. Директива имеет один параметр — количество байтов для резервирования. Директива может использоваться только для резервирования места в памяти данных (смотри директивы **CSEG**, **DSEG** и **ESEG**).

**Синтаксис:**

Метка: .BYTE числовое выражение

**Примеры:**

```
.DSEG
var1: .BYTE 1          ; Резервируем 1 байт для переменной var1
table: .BYTE tab_size  ; Резервируем tab_size байт

.CSEG
ldi r30, low(var1)     ; Загружаем младший байт Z-регистра
ldi r31, high(var1)    ; Загружаем старший байт Z-регистра
ld r1, Z                ; Загрузить содержимое переменной var1 в r1
```

**.CSEG — сегмент кода.**

Директива CSEG определяет начало сегмента кода (программ). В исходном тексте программы может быть несколько сегментов кода. Транслятор ассемблера в процессе компиляции программы объединяет все сегменты кода в один. Директива BYTE не может быть использована в сегменте кода. Если в программе нет явного указания названия сегмента, по умолчанию считается, что это сегмент кода. Директива CSEG не имеет никаких параметров. Сегмент кода имеет свой счетчик слов. Директива ORG может быть использована для размещения кода или констант в определенном программистом месте памяти программ.

**Синтаксис:**

.CSEG

**Пример:**

```
.DSEG          ; Начало сегмента данных
vartab: BYTE 4 ; Резервируем 4 байта в SRAM

.CSEG          ; Начало сегмента кода
const: .DW 2    ; Запишем число 0x0002 в память программ
mov r1, r0      ; Что-нибудь сделаем
```

**.DB — определить байты-константы в памяти программ или EEPROM.**

Директива DB резервирует место в памяти программ или EEPROM. Для того чтобы иметь возможность обращаться к зарезервированному пространству, перед этой директивой следует ставить метку. Директива DB может быть расположена только в сегменте ко-

да или EEPROM. Параметрами директивы DB является список выражений.

Список выражений представляет собой одно или несколько выражений, разделенных запятыми. Каждое выражение может быть равно числу от -128 до 255. Если выражение представляет собой отрицательное число, оно будет помещено в память программ или EEPROM в дополнительном коде.

Если директива DB расположена в сегменте кода и имеет больше, чем одно выражение в списке параметров, выражения упаковываются таким образом, что два байта располагаются в одном слове памяти программ. Если число выражений нечетное, последнее выражение будет помещено в отдельное слово памяти программ, даже если после директивы DB следом расположена еще одна директива DB.

### Синтаксис:

Метка: DB список выражений

### Пример:

```
.CSEG
const: DB 0, 255, 0b01011100, -128, 0xaa
.ESEG
eeconst: DB 0xff
```

### .DEF — назначить регистру символьное имя.

Директива DEF позволяет назначить регистру символьное имя, что позволяет сделать программу гораздо понятнее и нагляднее. Можно назначить одному регистру несколько символьных имен. Символьное имя регистра может быть переопределено в последующем тексте программы.

### Синтаксис:

.DEF символьное имя = регистр

### Пример:

```
.DEF temp=r16
.DEF ior=r0

.CSEG
ldi temp, 0xf0      ; Загрузить в регистр temp число 0xf0
in ior, 0x3f        ; Прочитать содержимое SREG и записать в регистр ior
eor temp, ior       ; Исключающее ИЛИ между регистрами temp и ior
```

**.DEVICE** — определяет тип целевого микроконтроллера.

Директива **DEVICE** позволяет программисту указать, на каком микроконтроллере будет выполняться программа. Если в тексте программы указана эта директива, транслятор ассемблера будет проверять текст программы на наличие недопустимых операций (например, не поддерживаемых выбранным микроконтроллером). В случае попытки использования большего размера SRAM или EEPROM памяти, чем имеется у выбранного микроконтроллера, также будет выдано предупреждение. Если директива **DEVICE** отсутствует в тексте программы, разрешены все команды семейства микроконтроллеров AVR, а размеры памяти не проверяются.

Синтаксис:

```
.DEVICE AT9DS1200 | AT90S2313 | AT9DS2323 | AT90S2343 | AT90S4414  
          | AT90S8515 | ATMEGA103
```

*Примечание.* Появляются новые модели микроконтроллеров, поэтому при необходимости использовать более новый микроконтроллер следует самостоятельно отслеживать разрешенные инструкции в тексте программы (соответственно не применяя директиву **DEVICE**) либо воспользоваться обновленной версией транслятора ассемблера.

Пример:

```
.DEVICE AT90S1200 ; Используется микроконтроллер AT90S1200
```

```
.CSEG
```

```
push r30 ; Эта запись вызовет сообщение о том, что  
          ; выбранное устройство не поддерживает эту  
          ; инструкцию
```

**.DSEG** — сегмент данных.

Директива **DSEG** определяет начало сегмента данных. В исходном тексте программы на ассемблере может быть несколько сегментов данных. В процессе трансляции все они будут объединены в один. Обычно сегмент данных содержит только директивы **BYTE** с метками. Сегмент данных имеет свой счетчик байтов. Директива **.ORG** может быть использована для расположения переменных в конкретных местах SRAM. Директива **DSEG** не имеет параметров.

Синтаксис:

```
.DSEG
```

**Пример:**

```
.DSEG                                ; Начало сегмента данных
var1:.BYTE 1                        ; Резервируем 1 байт для переменной var1
table:.BYTE tab_size                ; Резервируем tab_size байт

.CSEG
ldi r30,low(var1)                   ; Загружаем младший байт Z-регистра
ldi r31,high(var1)                  ; Загружаем старший байт Z-регистра
ld r1,Z                             ; Загрузить содержимое переменной var1 в r1
```

**.DW — определение слов-констант в памяти программ или EEPROM.**

Директива DW резервирует место в памяти программ или EEPROM. Для того чтобы иметь возможность обращаться к зарезервированному пространству, перед этой директивой следует ставить метку. Директива DW должна быть расположена в сегменте кода или EEPROM. Параметрами директивы DW является список выражений.

Список выражений представляет собой одно или несколько выражений, разделенных запятыми. Каждое выражение может быть равно числу от -32768 до 65535. Если выражение представляет собой отрицательное число, оно будет помещено в память программ или EEPROM в дополнительном коде.

**Синтаксис:**

Метка: .DW список выражений

**Пример:**

```
.CSEG
varlist:.DW 0, 56255, 0b0101110011101011, -12128, 0xaaFF
.ESEG
eevar: .DW 0xFF17
```

**.ENDMACRO — конец описания макрокоманды.**

Директива ENDMACRO определяет конец описания макрокоманды. Директива не имеет параметров. Для получения информации о макрокомандах смотри директиву MACRO.

**Синтаксис:**

.ENDMACRO

**Пример:**

```
.MACRO SUBI16                        ; Начало определения макрокоманды
subi r16,low(@0)                    ; Вычитаем младший байт
```

```
sbc r17,high(@0)      ; Вычитаем старший байт
.ENDMACRO              ; Конец определения макрокоманды
```

### **.SET — присвоить символьному обозначению выражение.**

Директива SET присваивает символьному обозначению значение выражения. В дальнейшем это символьное обозначение может быть использовано в выражениях. Присвоенное значение — константа. В дальнейшем тексте программы это символьное выражение не может быть переопределено или изменено.

#### **Синтаксис:**

```
.EQU <символьное обозначение> = <выражение>
```

#### **Пример:**

```
.EQU io_offset = 0x23
.EQU porta = io_offset + 2

.CSEG                ; Начало сегмента кода
clr r2               ; Очистить регистр r2
out porta,r2         ; Записать в порт A
```

### **.ESEG — EEPROM сегмент.**

Директива ESEG определяет начало EEPROM сегмента. В исходном тексте программы может быть несколько EEPROM сегментов. Транслятор ассемблера в процессе компиляции программы объединяет все EEPROM сегменты в один. Директива BYTE не может быть использована в EEPROM сегменте. Директива ESEG не имеет никаких параметров. Сегмент EEPROM имеет свой счетчик байтов. Директива .ORG может быть использована для размещения кода или констант в определенном программистом месте памяти EEPROM.

#### **Синтаксис:**

```
.ESEG
```

#### **Пример:**

```
.DSEG                ; Начало сегмента данных
vartab:.BYTE 4       ; Резервируем 4 байта в SRAM

.ESEG
eevar:.DW 0xff67      ; Инициализируем одно слово в EEPROM

.CSEG                ; Начало сегмента кода
const:.DW 2           ; Запишем число 0x0002 в память программ
mov r1,r0             ; Что-нибудь сделаем
```

### **.EXIT — конец текста программы.**

Директива EXIT указывает транслятору ассемблера, что следует завершить трансляцию программы. При отсутствии этой директивы транслятор ассемблера работает до тех пор, пока исходный файл не закончится (EOF). Если директив EXIT встречается в файле, включаемом в текст директивой INCLUDE, транслятор ассемблера продолжит работу со строки, следующей после соответствующей директивы INCLUDE.

#### **Синтаксис:**

.EXIT

#### **Пример:**

.EXIT ; Завершить обработку этого файла

### **.INCLUDE — вставить файл.**

Директива INCLUDE указывает транслятору ассемблера на необходимость вставить в исходный текст программы другой файл. Реально при обработке этой директивы транслируется файл, указанный в директиве INCLUDE, после завершения его обработки (при достижении конца файла или директивы .EXIT) продолжается обработка основного файла. Вложенные файлы, в свою очередь, могут иметь директиву INCLUDE. Для облегчения понимания можно представить себе, что в текст программы вместо директивы INCLUDE вставляется соответствующий файл.

#### **Синтаксис:**

.INCLUDE "имя файла"

#### **Пример:**

	; Файл iodefs.asm
.EQU sreg=0x3f	; Регистр статуса
.EQU sphigh=0x3e	; Старший байт стека
.EQU splow=0x3d	; Младший байт стека
	; Файл incdemo.asm
.INCLUDE "iodefs.asm"	; Включить в текст программы файл iodefs.asm
in r0,sreg	; Прочитать содержимое регистра статуса

**.LIST — включить генерацию листинга.**

Директива LIST включает генерацию листинга. Ассемблер генерирует листинг, содержащий исходный текст на ассемблере, адреса и коды операций. По умолчанию генерация листинга включена. В комбинации с директивой NOLIST можно организовать печать только нужных фрагментов программы.

**Синтаксис:**

.LIST

**Пример:**

.NOLIST	; Отключить генерацию листинга
.INCLUDE "macro.inc"	; Включаемые файлы не будут
.INCLUDE "const.def"	; показаны в листинге
.LIST	; Включить генерацию листинга

**.LISTMAC — включить раскрытие макрокоманд.**

Директива LISTMAC указывает транслятору ассемблера на необходимость показа в листинге содержимого макрокоманд. По умолчанию в листинге показывается только название макрокоманды.

**Синтаксис:**

.LISTMAC

**Пример:**

.MACRO MACX	; Определить макрокоманду
add r0,@0	
eor r1,@1	
.ENDMACRO	; Конец определения макрокоманды
.LISTMAC	

MACX r2,r1	; Вызов макрокоманды. В листинге будет
	; показан ее текст

**.MACRO — начало определения макрокоманды.**

Директива MACRO указывает транслятору ассемблера на начало определения макрокоманды. Параметром директивы MACRO является имя определяемой макрокоманды. В дальнейшем при обнаружении в тексте программы имени макрокоманды транслятор ассемблера будет фактически заменять это имя на содержание макрокоманды. Макрокоманда может иметь до 10 параметров. Эти параметры

имеют фиксированные имена: @0...@9. При вызове макрокоманды параметры должны быть представлены в виде списка, разделенного запятыми. Определение макрокоманды завершается директивой ENDMACRO.

При определении новой макрокоманды нельзя использовать другие макрокоманды (т. е. нельзя использовать вложенные макрокоманды).

Макрокоманда должна быть определена в тексте программы до того, как ее используют.

По умолчанию в листинге генерируется только вызов макрокоманды. Чтобы получить в листинге содержимое макрокоманд, следует использовать директиву LISTMAC. Текст макрокоманды в листинге помечен символом «+».

**Синтаксис:**

```
.MACRO
```

**Пример:**

```
.MACRO SUBI16                ; Начало определения макрокоманды
    subi    r16,low(@0)      ; Вычитаем младший байт
    sbci    r17,high(@0)     ; Вычитаем старший байт
.ENDMACRO                    ; Конец определения макрокоманды

.CSEG                        ; Начало сегмента кода
SUBI16 9x1234,r16,r17        ; Вычесть 0x1234 из r17:r16
```

*Примечание:* r17:r16 в данном случае — пара регистров, содержащая 16-разрядное число.

**.NOLIST — включить генерацию листинга.**

Директива NOLIST выключает генерацию листинга. Ассемблер генерирует листинг, содержащий исходный текст на ассемблере, адреса и коды операций. По умолчанию генерация листинга включена. В комбинации с директивой LIST можно организовать печать только нужных фрагментов программы.

**Синтаксис:**

```
.NOLIST
```

**Пример:**

```
.NOLIST                    ; Отключить генерацию листинга
```

```

INCLUDE "macro.inc"      ; Включаемые файлы не будут
INCLUDE "const.def"      ; показаны в листинге
LIST                     ; Включить генерацию листинга

```

### **.ORG — установка значения счетчика расположения.**

Директива ORG присваивает абсолютное значение счетчику. Параметром директивы является значение, которое должно быть присвоено счетчику. При использовании директивы ORG в сегменте данных будет определено значение, указывающее расположение в оперативной памяти SRAM. При использовании директивы ORG в сегменте кода будет определено значение, указывающее расположение в памяти программ. При использовании директивы ORG в сегменте EEPROM будет определено значение, указывающее расположение в памяти EEPROM.

Если перед директивой расположена метка (на этой же строке), метка получит значение параметра директивы. Значение по умолчанию для сегмента кода и EEPROM равно 0, а для SRAM — 32 (так как регистры занимают пространство с 0 до 31). Обратите внимание, что для EEPROM и SRAM отсчитываются байты, в то время как в памяти программ — слова.

#### **Синтаксис:**

.ORG выражение

#### **Пример:**

```

.DSEG                      ; Начало сегмента данных (SRAM)
.ORG 0x37                  ; Установить адрес SRAM 37H
    variable:.BYTE 1      ; Резервировать 1 байт по адресу 37H SRAM

.ESEG                      ; Начало сегмента EEPROM
.ORG 0x20                  ; Установить значение счетчика расположения
    eever:.DW 0xf77a      ; Инициализировать слово в памяти EEPROM

.CSEG
.ORG 0x10                  ; Установить счетчик на значение 0x10

mov r0,r1                  ; Эта команда будет расположена в памяти
                           ; программ по адресу 0x10

```

### **.SET — присвоить символьному обозначению выражение.**

Директива SET присваивает символьному обозначению значение выражения. В дальнейшем это символьное обозначение может быть

использовано в выражениях. В дальнейшем тексте программы это символическое выражение может быть изменено.

### Синтаксис:

.SET символическое обозначение = выражение

### Пример:

```
.SET io_offset = 0x23
.SET porta = io_offset + 2

.CSEG          ; Начало сегмента кода
clr r2         ; Очистить регистр r2
out porta,r2   ; Записать в порт A
```

## Выражения

Ассемблер позволяет использовать в тексте программы выражения. Они могут содержать операнды, операции и функции. Все выражения имеют разрядность 32 бита.

### Операнды

Могут быть использованы следующие операнды:

- определенные программистом метки, имеющие значение счетчика, в зависимости от места своего расположения;
- переменные, определенные с помощью директивы SET;
- константы, определенные с помощью директивы EQU;
- целые константы:
  - десятичные (по умолчанию): 10, 255,
  - шестнадцатеричные (два вида записи): 0x0a, \$0a, 0xff, \$ff,
  - двоичные: 0b00001010, 0b11111111;
- коды символов ASCII: 'A', 'a';
- строки ASCII (без нуля в конце строки): «String»;
- PC — текущее значение счетчика команд в памяти программ.

### Функции

LOW(выражение) — возвращает младший байт выражения;  
HIGH(выражение) — возвращает старший байт выражения;  
BYTE2(выражение) — возвращает 2 байта выражения;  
BYTE3(выражение) — возвращает 3 байта выражения;

BYTE4(выражение) — возвращает 4 байта выражения;  
LWRD(выражение) — возвращает биты 0—15 выражения;  
HWRD(выражение) — возвращает биты 16—31 выражения;  
PAGE(выражение) — возвращает биты 16—21 выражения;  
EXP2(выражение) — возвращает  $2^{\wedge}$  выражения;  
LOG2(выражение) — возвращает целую часть  $\log_2$ (выражение).

### Операции

Ассемблер поддерживает различные операторы, описанные ниже. При их использовании можно применять скобки.

#### Логическое НЕ

Обозначение: !

Описание: унарный оператор, возвращает 1, если выражение равно нулю, и 0, если выражение было не равно нулю.

Приоритет: 14.

Пример: `ldi r16,!0xf0` ; Загрузить в r16 0x00

#### Побитовое НЕ

Обозначение: ~

Описание: унарный оператор, который возвращает исходное выражение со всеми инвертированными битами.

Приоритет: 14.

Пример: `ldi r16,~0xf0` ; Загрузить в r16 0x0f

#### Унарный минус

Обозначение: —

Описание: возвращает число с измененным на противоположный знаком.

#### Умножение

Обозначение: \*

Описание: возвращает результат умножения двух чисел.

Приоритет: 13.

Пример: `ldi r30,label*2` ; Загрузить в регистр r30 label\*2

#### Деление

Обозначение: /

**Описание:** возвращает целую часть от деления левого параметра на правый.

**Приоритет:** 13.

**Пример:** `ldi r30,label/2` ; Загрузить в регистр r30 label/2

### **Сложение**

**Обозначение:** +

**Описание:** возвращает сумму двух чисел.

**Приоритет:** 12.

**Пример:** `ldi r30,c1+c2` ; Загрузить в регистр r30 c1+c2

### **Вычитание**

**Обозначение:** -

**Описание:** возвращает результат вычитания правого числа из левого.

**Приоритет:** 12.

**Пример:** `ldi r17,c1-c2` ; Загрузить в регистр r30 c1-c2

### **Сдвиг влево**

**Обозначение:** <<

**Описание:** возвращает значение левого числа, сдвинутое влево на число раз, равное правому числу.

**Приоритет:** 11.

**Пример:** `ldi r17,1<<3` ; Загружает в регистр r17 число 1,  
; сдвинутое влево на 3 бита

### **Сдвиг вправо**

**Обозначение:** >>

**Описание:** возвращает значение левого числа, сдвинутое вправо на число раз, равное правому числу.

**Приоритет:** 11.

**Пример:** `ldi r17,1>>2` ; Загружает в регистр r17 число 1,  
; сдвинутое вправо на 2 бита

### **Меньше**

**Обозначение:** <

**Описание:** возвращает 1, если первое число меньше второго, иначе — 0.

**Приоритет:** 10.

**Пример:** `ori r18,bitmask*(c1<c2)+1`

**Меньше или равно**

Обозначение: &lt;=

Описание: возвращает 1, если первое число меньше второго или равно ему, иначе — 0.

Приоритет: 10.

Пример: `ori r18, bitmask*(c1<=c2)+1`**Больше**

Обозначение: &gt;

Описание: возвращает 1, если первое число больше второго, иначе — 0.

Приоритет: 10.

Пример: `ori r18, bitmask*(c1>c2)+1`**Больше или равно**

Обозначение: &gt;=

Описание: возвращает 1, если первое число больше второго или равно ему, иначе — 0.

Приоритет: 10.

Пример: `ori r18, bitmask*(c1>=c2)+1`**Равно**

Обозначение: ==

Описание: возвращает 1, если первое число равно второму, иначе — 0.

Приоритет: 9

Пример: `andi r19, bitmask*(c1==c2)+1`**Не равно**

Обозначение: !=

Описание: возвращает 1, если первое число не равно второму, иначе — 0.

Приоритет: 9.

Пример: `.SET flag=(c1!=c2)`**Побитовое И**

Обозначение: &amp;

Описание: возвращает результат побитной операции «И» между операндами.

Приоритет: 8.

Пример: `ldi r18, High(c1&c2)`

### **Побитовое исключающее ИЛИ**

Обозначение: `^`

Описание: возвращает результат побитной операции «исключающее ИЛИ» между операндами.

Приоритет: 7.

Пример: `ldi r18, Low(c1^c2)`

### **Побитовое ИЛИ**

Обозначение: `|`

Описание: возвращает результат побитной операции «ИЛИ» между операндами.

Приоритет: 6.

Пример: `ldi r18, Low(c1|c2)`

### **Логическое И**

Обозначение: `&&`

Описание: возвращает 1, если оба выражения не равны нулю, иначе — 0.

Приоритет: 5.

Пример: `ldi r18, Low(c1&&c2)`

### **Логическое ИЛИ**

Обозначение: `||`

Описание: возвращает 0, если оба выражения равны нулю, иначе — 0.

Приоритет: 4.

Пример: `ldi r18, Low(c1||c2)`

## **Описание программы WAVRASM**

Здесь опишем специфические особенности применения транслятора ассемблера WAVRASM.

### **Открытие файла программы**

Теоретически нет ограничений на количество одновременно открытых файлов исходных текстов. Размер каждого файла не должен

превышать примерно 28 Кб. Для работы с файлами большего размера следует использовать версию ассемблера, работающую из командной строки MS-DOS — avrasm. Также можно разбить всю программу на несколько файлов и объединить их с помощью директивы INCLUDE.

Для каждого открытого файла создается окно с его текстом.

Для создания нового файла следует выполнить команду меню File>>New (быстрая комбинация клавиш: Alt-F N). Для открытия существующего файла следует выполнить команду меню File>>Open (быстрая комбинация клавиш: Alt-F O).

### **Встроенный текстовый редактор**

#### **Перемещение по тексту программы**

Для перемещения по тексту программы можно пользоваться следующими командами:

- вправо — стрелка вправо;
- влево — стрелка влево;
- вверх — стрелка вверх;
- вниз — стрелка вниз;
- в начало строки — Home;
- в конец строки — End;
- в начало файла — Ctrl+Home;
- в конец файла — Ctrl+End.

#### **Редактирование текста**

Для редактирования текста следует пользоваться клавишами:

- вставить пробел — пробел;
- завершить строку — Enter;
- удалить символ слева от курсора — BackSpace;
- удалить символ справа от курсора — Del.

Для разбиения строки на две следует установить курсор на место разбиения и нажать Enter.

Для объединения двух строк следует установить курсор в начало второй строки и нажать клавишу BackSpace.

Выделение текста, операции копирования, перемещения и удаления осуществляются так же, как в любой программе для Windows.

### Установка опций программы

Некоторые установки транслятора ассемблера могут быть изменены. Для этого следует выполнить команду меню Options. Появится окно, подобное изображенному на рис. 4.4.

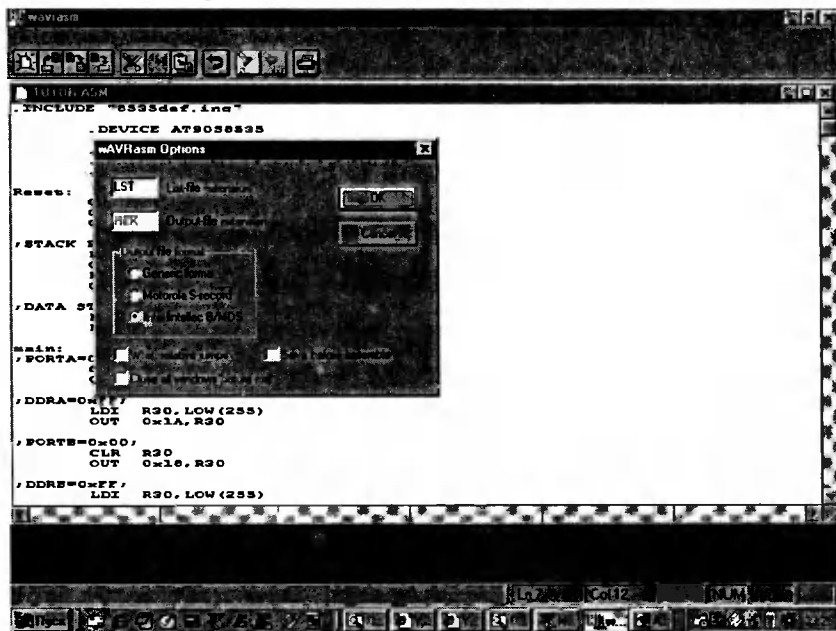


Рис. 4.4. Окно установки опций программы

В этом окне можно установить расширение файла, содержащего листинг программы и файла с оттранслированным кодом. Менять их не рекомендуется.

Также здесь можно указать, какого типа должен генерироваться выходной файл. Имеется три типа файлов: Generic, Motorola S-record и Intel HEX.

Обратите внимание, что объектный файл (который используется отладчиком) всегда имеет расширение obj. Также, если в программе инициализируются значения в памяти EEPROM, генерируется файл с расширением eep, используемый программатором для прошивки в микроконтроллер в процессе программирования. Этот файл генерируется в формате Generic.

Опция `Wrap relative jumps` — разрешить относительную адресацию переходов. Эта опция полезна для использования с микроконтроллерами, имеющими 4 К слов памяти программ.

Опция `Save before assemble` — сохранять исходный текст программы каждый раз перед ее ассемблированием.

### **Версия ассемблера для командной строки**

Одновременно с установкой транслятора ассемблера для Windows, устанавливается версия для работы из командной строки MS-DOS. Эта версия транслятора не имеет никаких ограничений по объему транслируемой программы (т. е., в отличие от версии для Windows, размер файла может быть больше 28 кбайтов).

Вызывается программа следующим образом:

```
avram [-m | -i | -g] input.asm output.lst output.hex
```

В результате выполнения программы будет прочитан файл `input.asm`, сгенерирован файл листинга `listfile.lst`, файл скомпилированного кода для загрузки в память программ микроконтроллера `output.hex` и объектный файл `*.obj`, используемый отладчиком.

#### **Назначение ключей программы**

`m` — генерировать файл кода в формате Motorola S-Record;

`i` — генерировать файл кода в формате Intel HEX;

`g` — генерировать файл кода в формате Generic.

По умолчанию генерируется файл кода в формате Generic.

Для выбора типа файла следует уточнить, с каким типом файлов может работать программатор, которым программа будет заноситься в микроконтроллер.

### **Формат файлов, генерируемых транслятором ассемблера**

#### **Формат Generic**

Рассматриваемый транслятор ассемблера может генерировать три различных типа файлов: Generic, Motorola S-Records и Intel HEX.

Рассмотрим один из этих форматов — Generic. Файлы этого формата — текстовые.

Структура файла очень проста. Каждая строка файла имеет вид:

Адрес: код операции.

Здесь «Адрес» — 6 цифр (24 бита) шестнадцатеричного числа, и «код операции» — 4 цифры (16 битов) шестнадцатеричного числа. «Адрес» определяет адрес в памяти программ, а «код операции» — содержимое памяти по указанному адресу.

В качестве примера рассмотрим программу на ассемблере:

; Демонстрация формата Generic

```
mov r0,r1
inc r1
call oursub
```

.org 0x50 ; Установить адрес в памяти программ на 0x50

```
oursub: add r1,r2
ret
```

В результате трансляции этой программы будет получен файл `gen_demo.gom` следующего содержания:

```
000000: 2c01
000001: 9413
000002: 940e
000003: 0050
000050: 0c12
000051: 9508
```

Обратите внимание на то, что команды, состоящие из двух слов, размещаются в двух строках файла.

Если в программе был определен сегмент EEPROM, то генерируется файл для прошивки в EEPROM. Расширение этого файла `eep`. Этот файл всегда генерируется в формате `Generic`.

### **Формат объектного файла**

Объектный файл содержит отладочную информацию и может быть использован отладчиком для проверки правильности работы программы.

Объектный файл имеет две части: `record` и `trailer`.

Заголовочная часть имеет следующий формат:

- смещение к исходным именам файлов (4 байта);
- смещение к объектным записям (4 байта);
- число байтов в каждой записи (1 байт);
- число имен файлов во второй части файла (1 байт);
- строку `AVR Object File\0` (0 означает, что строка завершена 0).

Длительность записей составляет 9 байтов каждая. Формат записи:

- адрес памяти программ (3 байта);
- код операции (2 байта);
- номер инструкции в исходном файле (1 байт, счет начинается с 0);
- номер строки в исходном файле (2 байта, счет начинается с 1);
- индикатор макроса (1 байт, 1 — если макрос, иначе — 0).

И наконец, trailer часть имеет следующий формат:

- имена файлов (заканчивающиеся на 0, число файлов в заголовке);
- ASCII 0.

Для примера рассмотрим программу (файл obj\_demo.asm):

```
; Демонстрация объектного формата
```

```
.equ const1=0x15
```

```
.equ const2=0x40
```

```
.macro SWIN
```

```
    swap @0
```

```
    inc @0
```

```
.endmacro
```

```
start: ldi r16,const1
```

```
    SWIN r16
```

```
    ldi r16,const2
```

```
    SWIN r16
```

```
    rjmp start
```

```
.include "delay.asm" ; Включение другого файла на ассемблере
```

Включаемый файл (delay.asm):

```
delay: dec r16
```

```
    breq delay
```

```
    ret
```

После трансляции будет получен объектный файл. Файл — бинарный. Для удобства рассмотрения он был переведен в 16-ричный формат, а столбцы были раздвинуты.

Смещение:            Содержимое файла (в 16-ричном формате):

00000000:    00000074    Смещение к именам файлов

00000004:    0000001A

00000008:    09

00000009:    02

0000000A:    415652204F626A6563742046696C6500    Строка A0F

0000001A:    000000E10500000B00    Первая запись

00000023:    000001950200000C01

## 4.2. Компилятор языка C CodeVision AVR

CodeVisionAVR Max1241.c [C:\cvavr\1017eval\examples\Max1241\max1241.c]

File Edit Project Tools Settings Windows Help

Navigator

- CodeVision
  - Project
  - Not
  - mas
  - Other F

```

1  /*****
2  Digital voltmeter using an
3  Maxim MAX1241 ADC
4  connected to an AT90S8515
5  using the SPI bus
6  The measured voltage is transmitted
7  through the STK200 RS232 interface
8  Communication parameters: 9600 8N1
9
10 CodeVisionAVR C Compiler
11 (C) 1998-2000 Faval Haiduc, HF InfoTech S.R.L.
12
13 Chip type       : AT90S8515
14 Clock frequency : 4.000000 MHz
15 Memory model    : Small
16 Internal RAM size: 512
17 External RAM size: 0
18 Data Stack size : 128
19 *****/
20
21 MAX1241 connections to the AT90S8515
22
23 [MAX1241] [AT90S8515 DIP40]
24 1 VDD=5V
25 2 Vin
26 3 /SHEN - 1 FBO
27 4 RXP=5V
28 5 GND - 20 GND

```

Messages

**Рис. 4.5.** Внешний вид окна программы CodeVision AVR

Программа представляет собой 32-разрядное приложение для работы в операционных системах Windows 95, 98, NT4.0 и 2000.

Кросс-компилятор включает в себя почти все элементы, соответствующие стандарту ANSI. Также в компилятор включены дополнительные возможности, ориентированные на использование архитектурных особенностей микроконтроллеров этого семейства и встроенных систем в целом.

Объектные файлы COFF позволяют осуществлять отладку программ с просмотром содержимого переменных. Для этого следует применять свободно распространяемый фирмой Atmel ([www.atmel.com](http://www.atmel.com)) отладчик AVR Studio debugger версии 3.5 или более поздний.

Для отладки систем, использующих последовательную передачу данных, в графической оболочке имеется встроенный Terminal.

Кроме стандартных библиотек языка C, компилятор имеет библиотеки для работы с:

- ЖКИ индикаторами со встроенным контроллером;
- шиной I2C фирмы Philips;
- датчиком температуры LM75 фирмы National Semiconductor;
- часами реального времени PCF8563 и PC8583 фирмы Philips, DS1302 и DS1307 фирмы Dallas Semiconductor;
- однопроводным протоколом фирмы Dallas Semiconductor;
- датчиками температуры DS1820 и DS1822 фирмы Dallas Semiconductor;
- датчиком температуры/термостатом DS1621 фирмы Dallas Semiconductor;
- памятью EEPROM DS2430 и DS2433 фирмы Dallas Semiconductor;
- шиной SPI;
- управлением режимами пониженного потребления энергии;
- временными задержками.

Также в CodeVision имеется автоматический генератор программ, который позволяет в течение считанных минут получить готовый код для следующих функций:

- настройка доступа к внешней памяти;
- определение источника прерывания Reset;
- инициализация портов ввода/вывода;
- инициализация внешних прерываний;

- инициализация таймеров/счетчиков;
- инициализация сторожевого таймера;
- инициализация UART;
- инициализация аналогового компаратора;
- инициализация встроенного АЦП;
- инициализация интерфейса SPI;
- инициализация поддерживаемых библиотеками CodeVision микросхем, работающих с однопроводным интерфейсом и шиной I<sup>2</sup>C;
- инициализация модуля ЖКИ-индикатора со встроенным контроллером.

Среда CodeVision AVR включает в себя программное обеспечение для работы с совместимым с платой STK200 программатором. После компиляции исходной программы на языке C полученный код может быть непосредственно запрограммирован в микроконтроллер. Этот программатор использует всего четыре сигнала: MOSI, MISO, SCK, RESET. Известно много простейших программаторов, соединяющихся с последовательным или параллельным портом IBM совместимого персонального компьютера.

Читателям предлагается на примере простейшей схемы познакомиться с применением демонстрационной версии компилятора языка C CodeVisionAVR. Для этого придется изготовить простой совместимый с STK200 кабель внутрисхемного программирования и несложную схему с микроконтроллером AT90S2313. В итоге проделанную работу можно будет проверить, запустив простую программу на языке C.

### **Изготовление кабеля для внутрисхемного программирования «STK200/300»**

На рис. 4.6 показана электрическая принципиальная схема кабеля. Микросхема 74HC244 представляет собой буфер с тремя состояниями на выходах, что позволяет избежать влияния кабеля на схему после программирования микроконтроллера, не отключая кабеля.

Адаптер получил свое название от комплектующихся им отладочных плат фирмы Atmel для быстрого начала работы с микроконтроллерами At90s8515 и Atmega103 соответственно. На самом деле приведенная схема соответствует одновременно обоим адаптерам, в

ней присутствуют перемычки для определения наличия как адаптера STK200 (выводы 2-12 разъема X1), так и STK300 (выводы 3-11).

Разводка колодки X2 на приведенной схеме соответствует принятой фирмой Atmel для производимых ею плат.

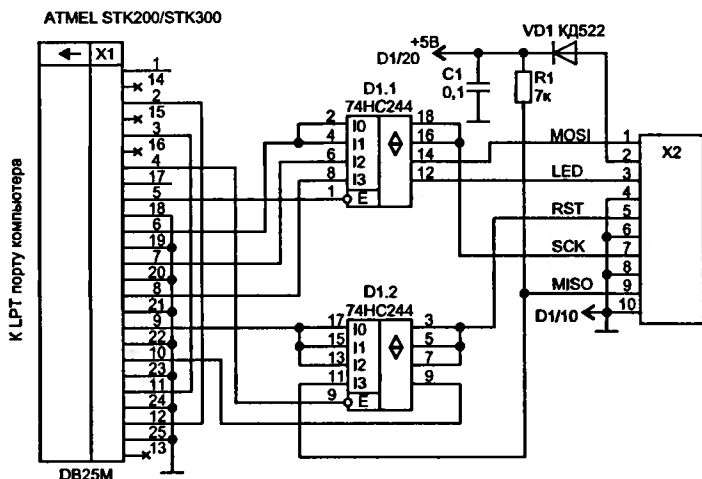


Рис. 4.6. Простой кабель для внутрисхемного программирования

### Простая демонстрационная схема на микроконтроллере AT90S8535

Чтобы проверить эту схему в работе, вам нужно иметь только один микроконтроллер AT90S2313, источник питания +5 В и кварцевый резонатор частотой 4 МГц. Кроме этого, потребуется несколько пассивных элементов — конденсаторов и резисторов. Электрическая принципиальная схема устройства изображена на рис. 4.7. Для управления светодиодом в схеме использован всего один вывод микроконтроллера — PD1.

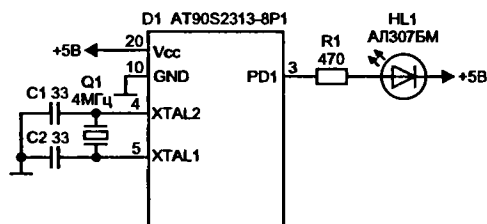


Рис. 4.7. Простейшая схема с микроконтроллером AT90S2313

### Пример программы

Для проверки работы кабеля внутрисхемного программирования и демонстрационной схемы предлагается несложная программа, которая заставляет мигать светодиод. Оттранслированный код для этой программы можно найти на прилагаемом к книге компакт диске в файле `blink.hex`.

Текст программы:

```
// blink.c

#include <90s2313.h>
#include <delay.h>

void main()
{
    // инициализация порта D
    DDRA=0xff;        // Порт D работает на вывод
    PORTD=0x00;

    while(1)
    {
        PORTD.1=0;
        delay_ms(1000);
        PORTD.1=1;
        delay_ms(1000);
    } // while(1)

} // main
```

### Использование встроенного программатора CodeVision

Простейший способ работы с прорамматором — использование команды меню **Project**, подменю **CONFIGURE**. Если осуществить указанные на рис. 4.8 установки, оттранслированный код программы будет загружен в микроконтроллер непосредственно после успешной компиляции.

Перед работой следует указать тип используемого программатора. На рис. 4.9 показан вид окна выбора типа программатора.

После успешной трансляции должно появиться окно, подобное показанному на рис. 4.10. Для занесения программы в микроконтроллер, следует нажать кнопку «**Program**».

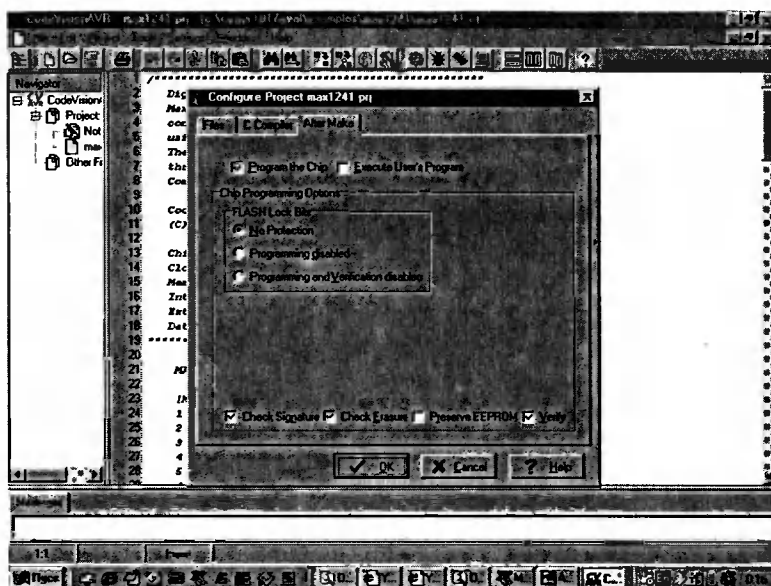


Рис. 4.8. Окно настройки проекта

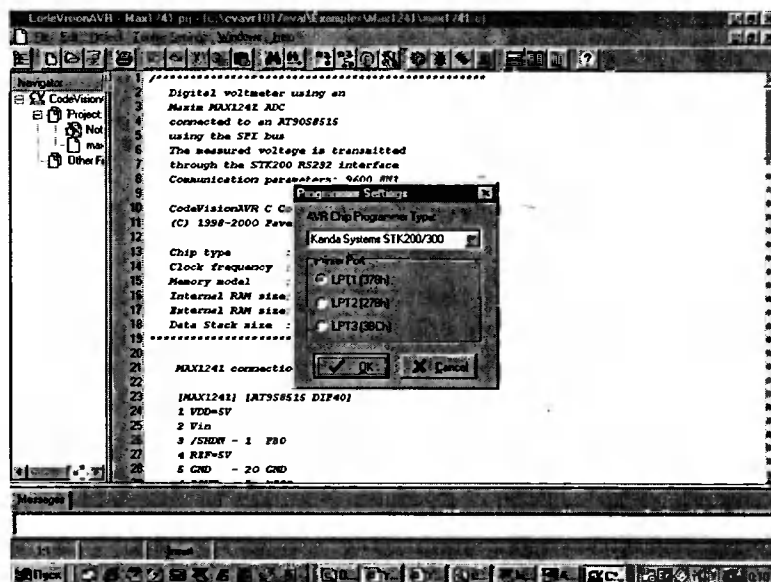


Рис. 4.9. Окно выбора типа программатора

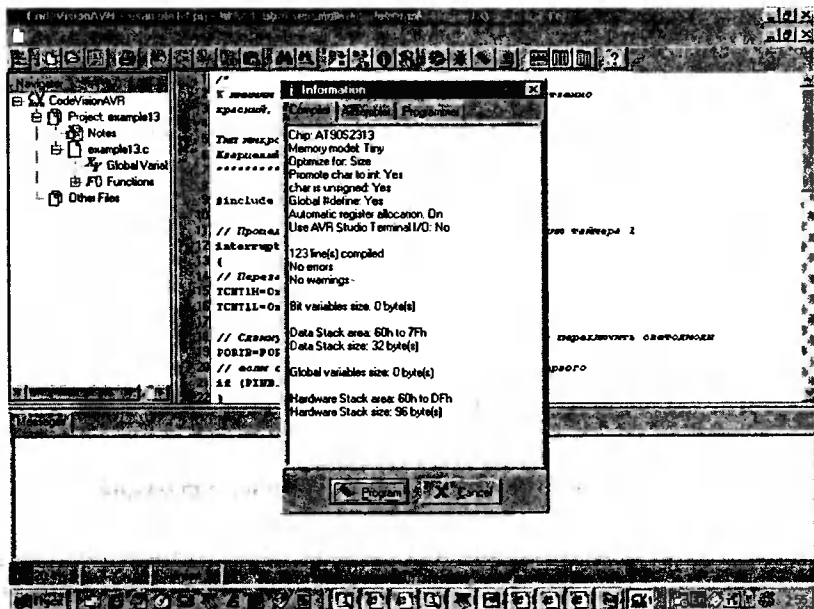


Рис. 4.10. Окно, появляющееся после успешной трансляции программы

## Примеры программ для компилятора CodeVision AVR C

### Процедуры работы со встроенным АЦП AT90S8535 без прерываний

Текст программы:

```
// В данном примере определяются две функции для работы с АЦП:
// void InitADC(void)           инициализация АЦП
// int ReadADC(unsigned char) чтение значения напряжения на заданном входе

#include <i08535.h>

void InitADC(void)
{
    ADMUX = 0;           // выбрать вход номер 0
    ADCSR = 0x0C;        // включить АЦП и запустить первое "пустое" преобразование
}

int ReadADC(unsigned char channel)
{
    int i;

    ADMUX = channel;      // Выбрать номер входа
    ADCSR |= 0x40;        // Начать преобразование
```

```

while (!(ADCSR & 0x10));           // Проверка завершения преобразования

ADCSR |= 0x10;                     // Очистка бита "Преобразование завершено" при
                                   // помощи записи в него "1"

i = ADCL;                          // Чтение младших 8 битов ПЕРВЫМИ
i += (int)ADCH << 8;               // Чтение старших 2 битов, умножение их на 256
                                   // и сложение с младшим байтом

return i;
}

void main(void)
{
    unsigned int temp;
    InitADC();                      // Инициализация АЦП
    temp=ReadADC(0);                // Измерить напряжение на нулевом входе АЦП (линия PA0 порта A)
}

```

### ***Пример вызова написанных на ассемблере функций из C программы***

Материал взят из демонстрационной версии компилятора Code-VisionAVR C Compiler, автором которого является Pavel Haiduc, HP InfoTech S.R.L.

#### **Текст программы:**

```

// Определение функции на ассемблере. Эта функция возвращает a+b+c
#pragma warn- // Запретить предупреждения
int sum_abc(int a, int b, unsigned char c) {
    #asm
        ldd r30,y+3           ;R30=LSB a
        ldd r31,y+4           ;R31=MSB a
        ldd r26,y+1           ;R26=LSB b
        ldd r27,y+2           ;R27=MSB b
        add r30,r26           ;(R31,R30)=a+b
        adc r31,r27
        ld r26,y              ;R26=c
        clr r27                ;Преобразование с типа unsigned char в тип int
        add r30,r26            ;(R31,R30)=(R31,R30)+c
        adc r31,r27
    #endasm
}
#pragma warn+ // Разрешить предупреждения

void main(void) {
    int r;
    // Теперь вызовем функцию и сохраним результат в r
    r=sum_abc(2,4,6);
}

```

### *Некоторые пояснения.*

Компилятор передает параметры функции с помощью стека данных. Первым он передаст целый параметр *a*, затем *b* и в завершение *c* типа `unsigned char`. При каждой передаче регистровая пара *Y* увеличивается на размер параметра (4 для типа `long`, 2 для `int`, 1 для `char`).

В случае параметров, состоящих из нескольких байтов, первым передается старший байт. Как вы видите, стек растет вниз. После того как все параметры функции были записаны в стек (`pushed`), регистр *Y* указывает на последний параметр *c*, поэтому мы можем прочитать его значение в *R26*, воспользовавшись командой `ld r26,y`.

Параметр *b* был записан в стек перед *c*, поэтому он находится по более высокому адресу в стеке данных. Мы можем прочитать его значение, воспользовавшись командами `ldd r27,y+2` (старший байт) и `ldd r26,y+1` (младший байт).

Старший байт был записан в стек первым, поэтому он находится по более высокому адресу.

Параметр *a* был записан в стек перед *b*, поэтому он находится по более высокому адресу в стеке данных. Мы можем прочитать его значение, воспользовавшись командами `ldd r27,y+4` (старший байт) и `ldd r26,y+3` (младший байт).

Старший байт был записан в стек первым, поэтому он находится по более высокому адресу.

Функции возвращают свои значения в следующих регистрах:

*R30* для типов `char` & `unsigned char`;

*R30*, *R31* для типов `int` & `unsigned int`;

*R30*, *R31*, *R22*, *R23* для типов `long` & `unsigned long`.

Поэтому наша функция должна вернуть ее результат в регистрах *R30*, *R31*.

После возвращения из функции компилятор автоматически генерирует код, освобождающий стек от параметров функции, поэтому можно не задумываться об этом.

Директива `#pragma warn` запрещает компилятору генерировать предупреждения о том, что функция не возвращает результат.

Это необходимо, потому что компилятору не известно, что мы делаем в нашей написанной на ассемблере функции.

**Использование встроенного  
EEPROM**

Материал взят из демонстрационной версии компилятора Code-VisionAVR C Compiler, автором которого является Pavel Haiduc, HP InfoTech S.R.L.

**Текст программы:**

```
// Контроллер: AT90S2313
// Модель памяти: TINY
// Размер стека данных: 64 bytes

flash char f[]="This is a test";
#pragma warn-
eeprom char e[16];

#pragma warn+
char r[16];

void main (void)
{
char flash *ptr_to_flash;
char eeprom *ptr_to_eeprom;
char *ptr_to_ram;

// Копировать строку f из FLASH в
// строку e в EEPROM
ptr_to_flash=f;
ptr_to_eeprom=e;
while (*ptr_to_flash)
    *ptr_to_eeprom++=*ptr_to_flash++;

// Копировать строку e из EEPROM в
// строку r в оперативной памяти
ptr_to_eeprom=e;
ptr_to_ram=r;
while (*ptr_to_eeprom)
    *ptr_to_ram++=*ptr_to_eeprom++;

// Стоп (бесконечный цикл)
while (1);
}
```

**Работа с клавиатурой  
4 x 4**

Материал взят из демонстрационной версии компилятора Code-VisionAVR C Compiler, автором которого является Pavel Haiduc, HP InfoTech S.R.L.



```

// Инициализация использованного таймера TIMERO
#define INIT_TIMER0 TCNT0=0x100L-F_XTAL/64L/500L
#define FIRST_COLUMN 0x80
#define LAST_COLUMN 0x10

typedef unsigned char byte;
// Здесь в виде бита сохраняется состояние каждой нажатой клавиши,
// бит 0 будет KEY0, бит 1 KEY1, ...
unsigned keys;
// Буфер ЖКИ-индикатора
char buf[33];

// Прерывание по таймеру TIMER 0 каждые 2 мс
interrupt [TIMO_OVF] void timer0_int(void)
{
    static byte key_pressed_counter=20;
    static byte key_released_counter,column=FIRST_COLUMN;
    static unsigned row_data,crt_key;
    // Перезагрузить таймер TIMERO
    INIT_TIMER0;
    row_data<<=4;
    // Получить группу из 4 клавиш в переменной row_data
    row_data|=~KEYIN&0xf;
    column>>=1;
    if (column==(LAST_COLUMN>>1))
    {
        column=FIRST_COLUMN;
        if (row_data==0) goto new_key;
        if (key_released_counter) --key_released_counter;
        else
        {
            if (--key_pressed_counter==9) crt_key=row_data;
            else
            {
                if (row_data!=crt_key)
                {
                    new_key:
                    key_pressed_counter=10;
                    key_released_counter=0;
                    goto end_key;
                };
            }
            if (!key_pressed_counter)
            {
                keys=row_data;
                key_released_counter=20;
            };
        };
    };
    end_key:;
}

```

```
    row_data=0;
};

// Выбрать следующий столбец, входы будут притянуты к 5 В
KEYOUT=~column;
}

// Проверить, были ли нажаты клавиши
unsigned inkey(void)
{
    unsigned k;
    if (k=keys) keys=0;
    return k;
}

void init_keypad(void)
{
    DDAD=0xf0;
    INIT_TIMER0;
    TCCR0=3;
    TIMSK=2;
    #asm("sei")
}

main() {
    unsigned k;
    init_keypad();
    lcd_init(16);
    lcd_putsf("CVAVR Keypad");
    // Читать состояние клавиш и индцировать код клавиши
    while (1)
    {
        lcd_gotoxy(0,1);
        if (k=inkey())
        {
            sprintf(buf, "Key code=%Xh", k);
            lcd_puts(buf);
        }
        else lcd_putsf("NO KEY   ");
        delay_ms(500);
    }
}
```

**Работа с алфавитно-цифровым ЖК-индикатором  
2 × 16**

Материал взят из демонстрационной версии компилятора Code-VisionAVR C Compiler, автором которого является Pavel Haiduc, HP InfoTech S.R.L.

Использован ЖК-индикатор со встроенным контроллером, подсоединенный к порту PORTC следующим образом:

	ЖКИ			PORTC
1	GND	—	9	GND
2	+5V	—	10	VCC
3	VLC	—		LCD HEADER Vo
4	RS	—	1	PC0
5	RD	—	2	PC1
6	EN	—	3	PC2
11	D4	—	5	PC4
12	D5	—	6	PC5
13	D6	—	7	PC6
14	D7	—	8	PC7

### Текст программы:

```
// ЖК-индикатор подсоединен к выходам порта PORTC
// смотри файл lcd.h в директории..\inc
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm

// Включить в состав программы описания и процедуры для работы с ЖК-индикатором
#include <lcd.h>

void main(void)
{
// Инициализировать ЖК-индикатор для работы
// с 2 строками по 16 символов
lcd_init(16);

// Перейти на 2-ю строку ЖК-индикатора
lcd_gotoxy(0, 1);

// Отобразить сообщение
lcd_putsf("Hello world");

// Остановиться (бесконечный цикл)
while (1);
}
```

**Использование определенных пользователем символов при работе с ЖК-индикатором со встроенным контроллером**

Материал взят из демонстрационной версии компилятора CodeVisionAVR C Compiler, автором которого является Pavel Haiduc, HP InfoTech S.R.L.

Использован алфавитно-цифровой ЖК-индикатор. Соединения между ЖК-индикатором и платой со схемой должны быть как можно короче.

### Текст программы:

```
// Включить в программу определения и процедуры для работы со ЖК-индикатором
// в плате STK200/300
#include <lcdstk.h>

typedef unsigned char byte;

// Таблица для определенного пользователем символа
// стрелка, указывающая на верхний правый угол
flash byte char0[8]={
0b0000000,
0b0001111,
0b0000011,
0b0000101,
0b0001001,
0b0010000,
0b0100000,
0b1000000};

// Функция, использованная для определения заданного пользователем символа
void define_char(byte flash *pc, byte char_code)
{
byte i, a;
a=(char_code<<3)|0x40;
for (i=0; i<8; i++) lcd_write_byte(a++, *pc++);
}

void main(void)
{
// Инициализация ЖКИ для работы
// с 2 строками по 16 символов в строке
lcd_init(16);

// Определить символ 0
define_char(char0, 0);

// Переключиться в режим записи в память отображения (Display RAM)
lcd_gotoxy(0, 0);
// Отобразить определенный пользователем символ
lcd_putsf("User defined\nchar 0:");
// Отобразить определенный пользователем символ 0
lcd_putchar(0);

// Стоп (бесконечный цикл)
while (1);
}
```

**Бегущий огонь на светодиодах**

Материал взят из демонстрационной версии компилятора CodeVisionAVR C Compiler, автором которого является Pavel Haiduc, HP InfoTech S.R.L.

8 светодиодов соединены с выходами порта PORTB и +5 В через резисторы сопротивлением 270 Ом, ограничивающие ток. Аноды светодиодов соединены с линией +5 В.

**Текст программы:**

```
// Определение регистров ввода-вывода для AT90S8515
#include <90s8515.h>

// Частота кварцевого резонатора [Гц]
#define xtal 4000000
// Частота переключения светодиодов [Гц]
#define fmove 2

// Включить светодиод на линии 0 порта PORTB
unsigned char led_status=0xfe;

// Процедура обработки прерывания по переполнению таймера TIMER1
// вызывается каждые 0,5 с

interrupt [TIM1_OVF] void timer1_overflow(void)
{
    // preset again TIMER1
    TCNT1=0x10000-(xtal/1024/fmove);
    // move the LED
    led_status<<=1;
    led_status|=1;
    if (led_status==0xff) led_status=0xfe;
    // turn on the LED
    PORTB=led_status;
}

void main(void)
{
    // Инициализация портов ввода-вывода
    // Все выходы порта PORTB настроены на вывод информации
    DDRB=0xff;
    // Включить первый светодиод
    PORTB=led_status;

    // Инициализация таймера TIMER1
    // Таймер TIMER1 отсоединен от линии OC1
    // не включен режим широтно-импульсной модуляции PWM
    TCCR1A=0;
    // Частота таймера TIMER1 xtal/1024
```

```

TCCR1B=5;
// Предустановим значение таймера TIMER1
TCNT1=0x10000-(xtal/1024/fmove);
// Очистить флаги прерываний по таймеру TIMER1
TIFR=0;
// Разрешить прерывания по переполнению таймера TIMER1
TIMSK=0x80;
// Все остальные виды прерываний запрещены
GIMSK=0;

// Разрешение глобальных прерываний
#asm
    sei
#endasm

// Бесконечный цикл, при этом возможно возникновение прерываний
// по переполнению таймера TIMER1
while (1);
}

```

### **Цифровой вольтметр с использованием АЦП фирмы Maxim типа MAX1241**

Измеренное значение передается с помощью интерфейса RS232.  
Параметры интерфейса: 9600 8N1.

Материал взят из демонстрационной версии компилятора Code-VisionAVR C Compiler, автором которого является Pavel Haiduc, HP InfoTech S.R.L.

Частота кварцевого резонатора: 4,000 МГц.

Подсоединение MAX1241 к AT90S8515.

	MAX1241			AT90S8515-8PI
1	VDD=5V			
2	Vin	—		
3	/SHD	—	1	PB0
4	REF=5V	—		
5	GND	—	20	GND
6	DOUT	—	7	MISO
7	/CS	—	2	PBI
8	SCLK	—	8	SCK

Программатор после программирования контроллера должен быть отсоединен.

## Текст программы:

```

#include <90s8515.h>

// Стандартная библиотека ввода/вывода
#include <stdio.h>

// Библиотека SPI-функций
#include <spi.h>

// Библиотека для формирования задержек
#include <delay.h>

// опорное напряжение для MAX1241 [мВ]
#define VREF 5000

// Определение управляющих сигналов для MAX1241
#define NSHDN PORTB.0
#define NCS PORTB.1
#define DOUT PINB.6

union adcu
{
    unsigned char byte[2];
    unsigned int word;
};

// Произвести одно аналого-цифровое преобразование и
// вернуть его результат
unsigned int max1241_read(void)
{
    union adcu adc_data;
    // Вывести микросхему MAX1241 из режима shutdown
    NSHDN=1;
    // Подождать 5 мкс для приведения MAX1241 в рабочее состояние
    delay_us(5);
    // Теперь выбрать чип для начала преобразования
    NCS=0;
    // Ждать, пока преобразование не завершится
    // DOUT будет равен 0 в процессе преобразования
    while (DOUT==0);
    // DOUT=1 -> преобразование завершено
    // Прочитать младший байт (MSB)
    adc_data.byte[1]=spi(0);
    // Прочитать старший байт (LSB)
    adc_data.byte[0]=spi(0);
    // Снять сигнал выбора чипа
    NCS=1;
    // Перевести АЦП в режим shutdown
    NSHDN=0;
    // Форматировать результат преобразования и возврат результата
    return (adc_data.word>>3)&0xffff;
}

```

```

void main(void)
{
// Переменная для хранения результата преобразования
unsigned n;

// Инициализация портов ввода/вывода
// Port A
DDRA=0x00;
PORTA=0x00;

// Port B
// Линия /SS установлена на вывод информации
// на уровне 1, это необходимо для работы
// SPI-порта в режиме мастер
DDRB=0xA3;
PORTB=0x12;

// Port C
DDRC=0x00;
PORTC=0x00;

// Port D
DDRD=0x00;
PORTD=0x00;

// Инициализация UART (интерфейс RS-232)
// Параметры интерфейса: 8 битов данных, 1 стоп бит, без проверки четности
// приемник UART: отключен
// передатчик UART: включен
UCR=0x08;
// Скорость передачи данных: 9600
UBRR=0x19;

// Инициализация SPI
// Режим работы SPI: мастер
// Тактовая частота SPI: 1000 000 кГц

// Порядок передачи данных: младший бит первый
SPCR=0x50;

putsf("MAX1241 Demo using the CodeVisionAVR C Compiler");
putsf("*****\n");

// Произвести аналого-цифровое преобразование и передать результаты
// через интерфейс RS-232
while (1)
{
n=max1241_read();
printf("MAX1241-> N=%4u U=%4umV\r\n",n,(unsigned) ((long) n*VREF/4096));
// 0.3 sec. delay
delay_ms(300);
};
}

```

**Использование ЖК-индикатора МТ10Т7-7**

Микроконтроллер: АТ90S8535-8PI.

Кварцевый резонатор: 4 МГц.

Подсоединение индикатора к порту Port C:

MAX1241	PORTC	AT90S8535 DIP 40
WR1	= 5	27
A0	= 4	26
DB0	= 0	22
DB1	= 1	23
DB2	= 2	24
DB3	= 3	25

**Текст программы:**

```
void delay(void)
{
    asm("del1: ldi r24,0x01");
    asm("d1: dec r24");
    asm("brne d1");
    asm("dec r25");
    asm("brne del1");
}

void STROB_WR1(void)
{
    asm("sbi 0x18,5");
    delay();
    asm("cbi 0x18,5");
}

void STROB_ADR(void)
{
    asm("cbi 0x18,4");
    delay();
    asm("sbi 0x18,5");
    delay();
    asm("cbi 0x18,5");
    delay();
    asm("sbi 0x18,4");
}

void Set_Bus(unsigned char A)
{
    if ((A&0x01)==0x01) asm("sbi 0x18,0");
                        else asm("cbi 0x18,0");
    if ((A&0x02)==0x02) asm("sbi 0x18,1");
```

```
        else asm("cbl 0x18,1");

if ((A&0x04)==0x04) asm("sbl 0x18,2");
        else asm("cbl 0x18,2");

if ((A&0x08)==0x08) asm("sbl 0x18,3");
        else asm("cbl 0x18,3");
}

void Init_LCD(void)
{
unsigned char temp;
Set_Bus(0x0f);
STROB_ADR();
Set_Bus(0x01);
STROB_WR1();

Set_Bus(0);
STROB_ADR();
for (temp=0;temp<20;temp++) STROB_WR1();
}

unsigned char code7(unsigned char code)
{
switch(code)
{
case 0: return 0xee;
case 1: return 0x60;
case 2: return 0x2f;
case 3: return 0x6d;
case 4: return 0xe1;
case 5: return 0xcd;
case 6: return 0xcf;
case 7: return 0x68;
case 8: return 0xef;
case 9: return 0xed;
case 10: return 0;
}
}

return 0;
}

void out(unsigned char num)
{
num=code7(num);
Set_Bus(num);
STROB_WR1();
delay();
asm("swap %num");
Set_Bus(num);
STROB_WR1();
delay();
}
```

```

void display(unsigned int N,unsigned int N1, unsigned int N2)
{
    NN N1N1N1N1 N2N2N2N2
    Set_Bus(0);
    STROB_ADR();

    out((unsigned int)(N/10)-10*(unsigned int)(N/100));
    out(N-10*(unsigned int)(N/10));

    out((unsigned int)(N1/1000)-10*(unsigned int)(N1/10000));
    out((unsigned int)(N1/100)-10*(unsigned int)(N1/1000));
    out((unsigned int)(N1/10)-10*(unsigned int)(N1/100));
    out(N1-10*(unsigned int)(N1/10));

    out((unsigned int)(N2/1000)-10*(unsigned int)(N2/10000));
    out((unsigned int)(N2/100)-10*(unsigned int)(N2/1000));
    out((unsigned int)(N2/10)-10*(unsigned int)(N2/100));
    out(N2-10*(unsigned int)(N2/10));
}

#include <io8535.h>

void main()
{
    DDRB = 0xFF;      /* Port C настроен на вывод */
    PORTB = 0xff;     /* Все линии порта C = 1 */
    Init_LCD();
    display(0,1,7);
    while(1);
}

```

### ***Динамическая индикация***

Динамическая индикация с применением прерывания по переполнению таймера Timer 0. Вывод на индикатор происходит каждые 65 мс. Прерывания по таймеру 1 происходят с периодом примерно 1 с. При этом на единицу увеличивается значение на индикаторе. При достижении значения 20 происходит обнуление значения для вывода.

Микроконтроллер: AT90S2313.

Кварцевый резонатор: 4 МГц.

Текст программы:

```

#include <90s2313.h>
#include "HG.h"

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
    // Reinitialize Timer's 0 value

```

```
TCNT0=0xFF;
// Place your code here
HG12();
}

// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
// Reinitialize Timer's 1 value
TCNT1H=0xf0;
TCNT1L=0xbe;
// Place your code here
N++;
if (N==20) N=0;
}

// Declare your global variables here

void main(void)
{
unsigned int temp,temp1,temp2;

// Input/Output Ports initialization
// Port B
PORTB=0x00;
DDRB=0xFF;

// Port D
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 3906 kHz
// Mode: Output Compare
// OCD output: Disconnected
TCCR0=0x05;
TCNT0=0xFF;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 3,906 kHz
// Mode: Output Compare
// OC1 output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x05;
TCNT1H=0xf0;
TCNT1L=0xbe;
OCR1H=0x00;
OCR1L=0x11;
```

```

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
GIMSK=0x00;
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x82;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;

// Global enable interrupts
#asm("sei")

init_HG();

N=0;

while (1)
{
};
}

```

### **Звуковой генератор на частоту 1000 Гц**

К нулевой линии порта В подсоединен транзисторный ключ, управляющий динамической головкой.

Микроконтроллер: AT90S2313.

Кварцевый резонатор: 4 МГц.

```

#include <90s2313.h>

// Процедура обработки прерывания Timer 1 по переполнению
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
// Перезагрузить значение Timer 1
TCNT1H=0xff;
TCNT1L=0xfd;
if (PINB.0==0) PORTB.0=1; else PORTB.0=0;
}

void main(void)
{
// Инициализация портов ввода/вывода
// Port B
DDRB=0xFF;
PORTB=0x00;

// Port D

```

```

PORTD=0x00;
DDRD=0x00;

// Инициализация Timer 1
// Clock source: System Clock
// Clock value: 3906 kHz
// Mode: Output Compare
// OC1 output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x05;
TCNT1H=0xff;
TCNT1L=0xfd;
OCR1H=0x00;
OCR1L=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x80;

PORTB=0x00;

while (1);

}

```

### **Модель светофора**

К линиям 0, 1 и 2 порта В подсоединены соответственно красный, желтый и зеленый светодиоды.

Микроконтроллер: AT90S2313.

Кварцевый резонатор: 4 МГц.

Текст программы:

```

#include <90s2313.h>

// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
// Reinitialize Timer's 1 value
TCNT1H=0xb3;
TCNT1L=0xb5;
// Place your code here
PORTB=PORTB<<1;
if (PINB.2==1) PORTB=0x01;
}

// Declare your global variables here

void main(void)
{
// Declare your local variables here

```

```
// Input/Output Ports initialization
// Port B
DDRB=0xFF;
PORTB=0x00;

// Port D
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 3906 kHz
// Mode: Output Compare
// OC1 output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x05;
TCNT1H=0xb3;
TCNT1L=0xb5;
OCR1H=0x00;
OCR1L=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0xB0;

PORTB=0x01;

while (1);

}
```

### 4.3. Компилятор Imagecraft C

Очень интересен для рассмотрения C компилятор фирмы ImageCraft. Этот оптимизирующий компилятор достаточно компактен: дистрибутив имеет объем всего 2,5 Мбайт. После установки на диск компилятор занимает чуть больше 4 Мбайт. Несмотря на скромные размеры, он позволяет создавать вполне серьезные программы. Интегрированная среда разработки (IDE) работает в среде Windows 95/98/NT и включает в себя текстовый редактор с полноценной поддержкой русского языка и подсветкой синтаксиса программ, и менеджер проекта. Компилятор поддерживает работу практически со всеми AVR-микроконтроллерами, в том числе с новыми ATmega161/163 и FPSLIC AT94. Для микросхем семейства ATtiny и

AT90S1200, не имеющих внутреннего ОЗУ, имеется пакет ICC Tiny AVR. Опции компиляции можно выбрать из стандартного набора или установить самостоятельно, выбрав в поле «тип микросхемы» — custom device. Компилятор позволяет получить стандартный HEX-файл для загрузки в микросхему, и файл в формате COFF, необходимый для работы с отладчиком AVR Studio. Компилятор совместим со стандартом ANSI C. В исходном тексте можно использовать вставки на ассемблере.

Библиотеки, поставляемые в комплекте с компилятором, включают достаточно большое количество функций, в том числе специализированных, ориентированных на специфические особенности микроконтроллеров, такие как доступ к ЭСПЗУ, АЦП и SPI-интерфейсу. В интегрированной среде имеется достаточно подробно написанный help-файл, практически полностью повторяющий содержание книги, входящей в комплект поставки компилятора. Есть очень полезная возможность обратиться за помощью непосредственно на фирму ImageCraft, написав письмо по адресу [info@imagecraft.com](mailto:info@imagecraft.com). В отличие от многих других производителей программного обеспечения, эта фирма достаточно оперативно отвечает на вопросы пользователей. Несомненным преимуществом работы с компилятором ImageCraft C является возможность полноценной работы до приобретения C-компилятора. Последняя демо-версия пакета доступна на сайте производителя программы по адресу <http://www.imagecraft.com>, причем эта версия максимально дружелюбна: пользователь может работать с исходными текстами неограниченной длины и компилировать файл для любой микросхемы. Единственным ограничением является время работы — 30 дней. Его достаточно для разработки даже крупного проекта. По истечении месяца пользователь должен принять решение: или переформатировать винчестер, чтобы еще раз установить демо-версию, либо приобрести официальный дистрибутив и право на техническую поддержку. Демонстрационная версия этого компилятора имеется на компакт-диске, прилагаемом к книге.

Ниже приведен пример простейшей программы для компилятора Imagecraft. Эта программа заставляет мигать восемь светодиодов, подключенных к порту В.

## Текст программы:

```
// Микроконтроллер: AT90S2313

#include <io2313.h>
#include <macros.h>

// Инициализация портов ввода/вывода
void init_ports(void)
{
    DDRA = 0xFF;           // Порт A работает на вывод
    PORTA = 0xFF;

    DDRD = 0x00;           // Порт D работает на ввод
    PORTD = 0x00;
}

// Временная задержка
void delay()
{
    unsigned char a, b;

    for (a = 1; a; a++)
        for (b = 1; b; b++);
}

void main(void)
{
    init_ports();           // Инициализация портов

    PORTB = 0xff;           // Погасили все светодиоды
    delay();

    PORTB = 0x00;           // Зажгли все светодиоды
    delay();
}
```

## 4.4. Компилятор языка C AVR GCC

Этот компилятор принципиально отличается от описанных выше тем, что он бесплатно распространяется, но при этом не имеет вообще никаких ограничений. Дело в том, что первоначальный вариант компилятора существовал (и существует) для операционной системы Линукс, практически, это — тот же компилятор, адаптированный для работы в среде Windows. По этой причине им несколько непривычно пользоваться, но он имеет достаточно неплохие характеристики.

Нижеприведенные программы идут в комплекте с свободно распространяемым компилятором AVR GCC.

**Мигание светодиодами**

Автор: Volker Oth.

Мигает светодиодами на плате STK200.

Текст программы:

```
#include <io.h>

typedef unsigned char u08;

int main(void)
{
    u08 led, i, j, k;

    outp(0xff, DDRB);          /* Все выходы порта В работают на вывод */

    led = 1;                   /* Инициализировать начальное состояние */

    for (;;) {
        outp(~led, PORTB);    /* Инвертировать выход. 0 - светодиод включен */
        led <=<= 1;           /* К следующему светодиоду */
        if (!led)              /* Переполнение: снова начать с линии В0 */
            led = 1;

        for (i=0; i<255; i++)   /* Цикл формирования временной задержки */
            for (j=0; j<255; j++) /* Вложенный цикл формирования временной задержки */
                k++;            /* Произвольное действие */
    }
}
```

**Мигание светодиодами с использованием таймера 0**

Автор: Volker Oth.

Мигает светодиодами, подключенными к порту В под управлением таймера 0.

Текст программы:

```
#include <io.h>
#include <interrupt.h>
#include <signal.h>

unsigned char led;

SIGNAL(SIG_OVERFLOW)          /* Обработчик прерывания переполнения таймера 0 */
{
    outp(~led, PORTB);        /* Инвертировать выходные линии. 0 - светодиод горит */
    led <=<= 1;                /* К следующему светодиоду */
    if (!led)                  /* Переполнение: начать снова с линии В0 */
        led = 1;
}
```

```

outp(0, TCNT0);          /* Сбросить таймер, для возможности повторного */
                          /* прерывания */
}

int main(void)
{
    outp(0xff, DDRB);      /* Все выводы порта В работают на вывод */

    outp((1<<TOIE0), TMSK); /* Разрешить прерывание по переполнению таймера 0 */
    outp(0, TCNT0);        /* Сбросить (обнулить) TCNT0 */
    outp(5, TCCR0);        /* Включить предварительное деление СК/1024 */

    led = 1;               /* Инициализация начального состояния светодиодов */
    sei();                 /* Разрешить прерывания */
    for (;;) {}            /* Бесконечный цикл */
}

```

### ***Иллюстрация использования внешних прерываний INT0 и INT1 и препротессора***

Автор: Volker Oth.

Текст программы:

```

#include <io.h>
#include <signal.h>
#include <interrupt.h>

#ifdef AVR_ATmega103
#define AVR_MEGA 1
#else
#ifdef AVR_ATmega603
#define AVR_MEGA 2
#else
#ifdef AVR_ATmega161
#define AVR_MEGA 3
#else
#define AVR_MEGA 0
#endif
#endif
#endif

typedef unsigned char u08,

SIGNAL(SIG_INTERRUPT0)    /* Обработчик внешнего прерывания int0 */
{
    register u08 led = inp(PORTB);

    if (led & 1)
        led &= ~0x0f;
    else
        led |= 0x0f;
}

```

```

    outp(led, PORTB);                /* Зажечь светодиоды */
}

SIGNAL(SIG_INTERRUPT1)              /* Обработчик внешнего прерывания int1 */
{
    register u08 led = inp(PORTB);

    if (led & 0x80)
        led &= ~0xf0;
    else
        led |= 0xf0;
    outp(led, PORTB);                /* Зажечь светодиоды */
}

int main(void)
{
    outp(0xff, DDRB);                /* Все линии порта B на вывод (светодиоды) */
    outp(0x00, DDRD);                /* Все линии порта D на ввод (кнопки) */
#ifdef AVR_MEGA
    outp((1<<INT0)|(1<<INT1), EIMSK); /* Разрешить внешние прерывания int0, int1 */
#else
    outp((1<<INT0)|(1<<INT1), GIMSK); /* Разрешить внешние прерывания int0, int1 */
    outp((1<<ISC01)|(1<<ISC10)|(1<<ISC11), MCUCR);
                                     /* По слэду: int0, по нарастанию: int 1 */
#endif
    sei();                           /* Разрешить прерывания */
    for (;;) {}                      /* Бесконечный цикл */
}

```

### **Иллюстрация применения UART**

Автор: Volker Oth.

Иллюстрация применения UART. Работает совместно с программой terminal или подобной.

Формат UART: 9600 бод, 8 битов, 1 стоп-бит, без проверки четности.

Текст программы:

```

#include <io.h>
#include <interrupt.h>
#include <signal.h>

#define F_CPU      4000000          /* 4 МГц */
#define UART_BAUD_RATE  9600        /* 9600 бод */

#define UART_BAUD_SELECT (F_CPU/(UART_BAUD_RATE*161)-1)

typedef unsigned char u08;
typedef      char s08;

```

```

typedef unsigned short u16;
typedef      short s16;

static volatile u08 *uart_data_ptr;
static volatile u08 uart_counter;

SIGNAL(SIG_UART_TRANS)          /* Обработчик прерывания UART txd готов */
{
    uart_data_ptr++;

    if (--uart_counter)
        outp(*uart_data_ptr, UDR);    /* Записать байт в буфер данных */
}

SIGNAL(SIG_UART_RECV)           /* Обработчик прерывания "прием завершен" */
{
    register char led;

    led = inp(UDR);               /* Прочитать байт из буфера данных UART */
    outp(~led, PORTB);            /* Отобразить полученный байт на светодиодах, */
                                /* Подключенных к порту В */
}

void uart_send(u08 *buf, u08 size)
{
    if (!uart_counter) {          /* Записать первый байт в буфер данных */
        uart_data_ptr = buf;
        uart_counter = size;
        outp(*buf, UDR);
    }
}

void uart_init(void)             /* Инициализировать UART */
{
    /* разрешить прерывания RxD/TxD */
    outp((1<<RXCIE)|(1<<TXCIE)|(1<<RXEN)|(1<<TXEN), UCR);
    /* установить скорость */
    outp((u08)UART_BAUD_SELECT, UBRR);
}

int main(void)
{
    outp(0xff, DDRB);             /* Все линии порта В на вывод */
    outp(0x00, PORTB);           /* Зажечь светодиоды */

    uart_init();
    sei();                        /* Разрешить прерывания */

    for (;;) {                   /* Бесконечный цикл */
        uart_send("Serial Data from AVR received###", 32);
    }
}

```

**Работа с EEPROM и UART**

Автор: Volker Oth.

Читает и записывает EEPROM. При возникновении прерывания UART «передача завершена», содержимое EEPROM пересылается на компьютер. После получения байта от компьютера прерывание «прием завершен» отображает полученный байт на светодиодах и сохраняет его в EEPROM. Формат UART: 9600 бод, 8 битов, 1 стоп-бит, без проверки четности.

Текст программы:

```
#include <io.h>
#include <interrupt.h>
#include <signal.h>
#include <eeprom.h>

#define F_CPU      4000000          /* 4 МГц */
#define UART_BAUD_RATE  9600       /* 9600 бод */

#define EEPROM_SIZE (E2END+1)
#define UART_BAUD_SELECT (F_CPU/(UART_BAUD_RATE*161)-1)

typedef unsigned char u08;
typedef      char s08;
typedef unsigned short u16;
typedef      short s16;

u16 read_counter;
u16 write_counter;

SIGNAL(SIG_UART_RECV)              /* Обработчик прерывания "прием завершен" */
{
    register u08 ee_write;

    ee_write = inp(UDR);             /* Прочитать байт из буфера данных UART */
    outp(~ee_write, PORTB);          /* Отобразить байт на светодиодах */
    eeprom_wb(write_counter, ee_write); /* Записать байт в EEPROM */
    if (++write_counter >= EEPROM_SIZE) /* Переполнение: установить смещение 0 */
        write_counter = 0;
}

SIGNAL(SIG_UART_TRANS)            /* Обработчик прерывания "передача завершена" */
{
    register u08 ee_read;

    ee_read = eeprom_rb(read_counter); /* Прочитать следующий байт из EEPROM */
    outp(ee_read, UDR);                /* Записать байт в буфер данных UART */
    if (++read_counter >= write_counter) /* Переполнение: начать с 1-го символа */
        read_counter = 0;
}
```

```

int main(void)
{
    outp(0xff, DDRB);                /* Все линии порта B на вывод */
    outp(0x00, PORTB);               /* Зажечь все светодиоды */

    /* Разрешить прерывания RxD/TxD */
    outp(((1<<RXCIE)|(1<<TXCIE)|(1<<RXEN)|(1<<TXEN), UCR);
    /* Установить скорость 9600 */
    outp(UART_BAUD_SELECT, UBRR);

    sei();                           /* Разрешить прерывания */

    read_counter = 0;                /* Начать читать с первого байта в EEPROM */
    write_counter = 0;               /* Начать запись с первого байта в EEPROM */
    outp('#', UDR);                  /* Записать 1-й байт в буфер данных UART */

    for (;;) {}                     /* Бесконечный цикл */
}

```

### ***Демонстрирует использование библиотеки вычислений с плавающей запятой***

Автор: Volker Oth.

Демонстрирует использование библиотеки вычислений с плавающей запятой. Выполняет 4 основных арифметических операции. Результаты преобразуются в 16-битовый формат с фиксированной точкой и передаются на настольный ПК с помощью UART. Результаты могут быть просмотрены с помощью программы terminal или подобной:

```

$0000 (start identifier)
$006b = 107 = 10.0*(7.5+3.2)
$002b = 43 = 10.0*(7.5-3.2)
$00f0 = 240 = 10.0*(7.5*3.2)
$0017 = 23 = 10.0*(7.5/3.2)

```

Текст программы:

```

#include <io.h>
#include <interrupt.h>
#include <signal.h>

#define F_CPU      4000000
#define UART_BAUD_RATE  9600
#define UART_BAUD_SELECT (F_CPU/(UART_BAUD_RATE*161)-1)

typedef unsigned char u08;
typedef      char s08;
typedef unsigned short u16;

```

```

typedef      short s16;

u08 uart_ready;
u08 *uart_data_ptr;
s08 uart_counter;
s16 result_buf[5];          /* Буфер результата */
float a_buf[2] = { 7. };

SIGNAL(SIG_UART_TRANS)     /* Обработчик прерывания uart txд готов*/
{
    uart_data_ptr++;
    uart_counter--;

    if (uart_counter>0)
        outp(*uart_data_ptr, UDR);    /* Записать байт в буфер данных */
    else
        uart_ready = 1;               /* Готов отсылать */
}

void uart_send(u08 *buf, u08 size) /* Послать буфер на uart */
{
    /* Записать первый байт в буфер данных */
    if (!uart_ready) return;
    uart_ready = 0;                  /* Не готов отсылать */
    uart_data_ptr = buf;
    uart_counter = size;
    outp(*buf, UDR);
}

void calc(float a, float b)
{
    result_buf[0] = 0;
    result_buf[1] = (a+b)*10.0;
    result_buf[2] = (a-b)*10.0;
    result_buf[3] = (a*b)*10.0;
    result_buf[4] = (a/b)*10.0;
}

int main(void)
{
    /* Разрешить прерывания RxD/TxD */
    outp((1<<RXCIE)|(1<<TXCIE)|(1<<TXEN),UCR);
    /* Установить скорость */
    outp((u08)UART_BAUD_SELECT, UBRR);

    uart_ready = 1;                /* Готов отсылать */
    sei();                         /* Разрешить прерывания */

    for (;;) {                    /* Бесконечный цикл */
        calc(7.5, 3.2);

        uart_send((u08*)result_buf, 5*sizeof(s16));
    }
}

```

**Простейшие приемы печати и чтения UART**

Автор: Volker Oth.

Назначение: демонстрирует простейшие приемы печати и чтения UART.

Текст программы:

```
#include "uart.h"
#include <progmem.h>

int main(void)
{
    u08 data;
    UART_Init();                /* Инициализация UART */

    PRINT("Hello World !");
    EOL();

    for (;;) {                  /* Бесконечный цикл */
        PRINT("Press any key...");
        EOL();

        data = UART_ReceiveByte();

        PRINT("You pressed '");
        UART_SendByte(data);
        PRINT("' which is 0x");
        UART_Printfu08(data);
        PRINT(" in hexadecimal.");
        EOL();
    }
}
```

**Получение доступа к данным в памяти программ**

Текст программы:

```
#include <io.h>
#include <progmem.h>

typedef unsigned char u08;

u08 __attribute__((progmem)) leds[]={0xff, 0xe7, 0xc3, 0x81, 0x00, 0x81, 0xc3, 0xe7};

int main(void)
{
    u08 i, j, k, l;

    outp(0xff, DDRB);          /* Все выходы порта B на вывод */

    for (;;) {
        for (l=0; l<sizeof(leds); l++) {
```

```

outp(PRG_RDB(&leds[1]), PORTB);

for (i=0; i<255; i++)          /* Цикл временной задержки */
    for(j=0; j<255; j++)        /* Вложенный цикл временной задержки */
        k++;                    /* Любая операция (чтобы "занять" процессор) */
    }
}
}

```

## 4.5. Программатор

Одним из самых необходимых устройств для работы с микроконтроллерами является программатор. Это специальный прибор, позволяющий оттранслированную программу занести непосредственно в память микроконтроллера. Без него вся работа будет только теорией.

Существуют различные виды программаторов. Различаются они по сложности, цене, программному обеспечению для работы. Для микроконтроллеров семейства AVR можно выделить наиболее удачные программаторы — AS1 производства фирмы Аргуссофт, и AVReAl — самодельный, сконструированный Александром Редчуком. Последний отличается исключительной простотой в изготовлении и качеством работы, не уступающим промышленным программаторам.

В этой главе читатели могут познакомиться со схемой программатора, которая может работать с программой AVReAl, и научиться пользоваться этим программатором.

Программа avgeal.exe с авторским описанием работы с ней находится на компакт-диске, прилагаемом к книге.

Программатор AVReAl может работать с различными схемами программаторов. Остановимся на схеме, совместимой с использованной в стартовой плате STK200 фирмы Atmel. Почему именно эта схема? Потому что она же может работать совместно с компилятором языка C CodeVision AVR.

Схема программатора показана на рис. 4.12.

Легко увидеть, что схема действительно предельно проста. Тем не менее она позволяет осуществлять все необходимые операции: проверять память микроконтроллера на наличие в ней программы, стирать ее содержимое, зашивать новую программу и т. д.

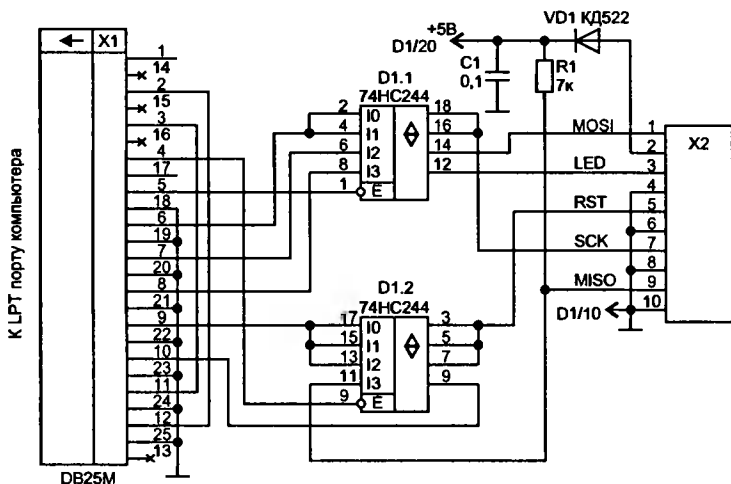


Рис. 4.12. Схема программатора ATMELEL STK200/STK300

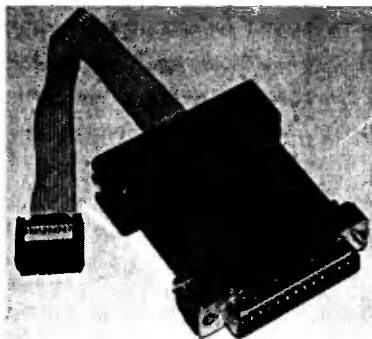


Рис. 4.13. Внешний вид программатора

Внешний вид собранного программатора показан на рис. 4.13.

Программатор работает с портом LPT1, 2 или 3. Если у вас порт LPT занят (например, принтером), можно купить на радиорынке отдельную плату с LPT-портом от старого компьютера, настроить ее на адрес порта LPT2 или 3 и подключать программатор к ней.

Следует иметь в виду, что длина проводов, соединяющих программатор и схему с микроконтроллером, не должна превышать 15...20 см. При большей длине начинают очень сильно влиять навод-

ки на провода и возникают ошибки при работе программатора. Для удобства работы следует приобрести удлинитель порта LPT и подключить программатор к нему.

В качестве разъема для программирования рекомендую разъем типа IDC10. На рис. 4.14 изображен внешний вид этого разъема и его цоколевка. Номера выводов на схеме программатора соответствуют

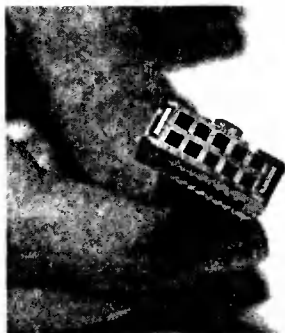


Рис. 4.14. Разъем IDC10

номерам на этом разъеме.

Следует стараться использовать на всех своих схемах с микроконтроллерами AVR один тип разъема программирования и его цоколевку — в этом случае можно будет пользоваться одним программатором для любых схем.

#### **Описание программатора avreal**

Автор программатора и настоящего описания: Александр Редчук.  
(E-mail: [real@real.kiev.ua](mailto:real@real.kiev.ua)).

AVReAl — программатор At90sXXXX через LPT.

Новые версии, описание и FAQ лежат на

<http://www.ln.com.ua/~real/avreal>;

<http://www.chat.ru/~avreal>.

Компилируется в следующих вариантах: 16-битовом DOS, для работы на процессорах начиная с 286; 32-битовом WIN32, для рабо-

ты необходимы файлы: для Windows 95/98 — DLportIO.DLL, для Windows NT — LportIO.DLL и DLportIO.sys.

Это файлы из комплекта DriverLINX от Scientific Software Tools, Inc. (<http://www.sstnet.com>). Его поставка достаточно громоздкая, с примерами работы на C и VisualBASIC, автоматической инсталляцией DLportIO.SYS для Windows NT и т. д. — итого файл port95nt.exe имеет размер 1,6 Мб.

В переупакованном виде p95nt.zip лежит у автора на <http://www.ln.com.ua/~real/avreal/p95nt.zip>.

Также существует вариант программатора для LINUX, но он еще недостаточно оттестирован.

Подключение LPT к чипу осуществляется через Altera ByteBlaster (ключ -ab) или через адаптер, совместимый с платами STK200/STK300 (ключ -as).

Подключение выводов процессоров:

Сигнал	Выводы микроконтроллеров			
	1200/2313	8535	8515	mega103/603
/RESET	1	9	9	20
SCK	19	8	8	11
MOSI	17	6	6	2
MISO	18	7	7	3
XTAL1	5	13	19	24

При использовании буферизованных адаптеров (Altera ByteBlaster, Atmel STK\*00) программа рассчитана на работу в схеме с «родными» питанием и кварцем. Возможна генерация XTAL1 по линии LED в STK\*00 и по дополнительной линии, выведенной на контакт 8 в ByteBlaster.

Программа использует файлы в формате INTEL HEX (avrasm -i). При этом она загружает файл прошивки не в массив, а в список. То есть различается ситуация «байт 0xFF в .hex» и «непомянутый байт». В результате есть возможность шить только то, что надо, остальное только проверять при необходимости (см. ключ -v+).

Для любителей патчить прямо по HEX-файлу: если не совпадает контрольная сумма строки, то задается вопрос: «игнорировать?». При ответе 'Y' (или 'y') эта и все остальные ошибки CSUM игнорируются (но сами символы контрольной суммы должны присутствовать, иначе программа завершит работу еще раньше по недопустимому символу в строке).

Для облегчения таких патчей (особенно «форматных» записей в EEPROM) реализована возможность, подсмотренная у программы от VMK@real.kiev.ua для программирования PIC через Tait-совместимое железо: перед анализом строки из нее убираются пробелы и табуляции. В итоге можно дать:

```
:08 0001 00 00 01 0203 0405 0 6 0 7 DB
```

Также игнорируются пустые строки и строки, начинающиеся с '#', последнее предназначено для помещения комментариев о версии непосредственно в HEX-файлы.

Проверяется верхний адрес в HEX-файле, если не помещается в опознанный чип, то никаких действий (кроме выдачи сообщений) не осуществляется. Проверяется перекрытие адресов записей в HEX-файле. При нахождении первого перекрытия производится выход из программы с указанием диапазона адресов перекрывающихся данных.

Для megal03 необходим расширенный формат HEX-файла (обычный не поддерживает объем больше 64 Кб). Запись прочитанных данных в файл всегда производится с дополнительными записями типа '04'-Extended Linear Address Record, как не имеющими различий. Чтение из HEX-файла записей типа '02' Extended Segment Address Record производится в соответствии с формированием их в AVRASM by Atmel (без предусмотренного документацией Intel сворачивания адреса смещения по модулю 64 Кб).

### **Вызов программы**

```
avreal [ключи] [[-с] имя_файла_кода [[-d] имя_файла_данных]]
```

или

```
avreal [ключи] -d имя_файла_данных (когда нет желания трогать код)
```

При пустой командной строке (ни имен, ни ключей) не делается никаких операций с портами/чипом. Если есть только имена файлов, но нет относящихся к ним ключей (-г -w -v), то файлы игнорируются. Подробнее про работу с файлами и про ключи -с, -d рассказано после описания остальных ключей. Признак ключа — символ '-' или '/', регистр букв не важен.

### Ключи

**-? (-h)** — выдать ключи и перечень поддерживаемых кристаллов.

**avreal +name -?**

(+name должен быть задан раньше ключа -?) выдает список fuses, поддерживаемых в режиме ISP для чипа name.

**+name** — установить тип кристалла, <name> задается без префикса 'At', например, 90s1200, 90s8515, mega103 (т. е. в том виде, в котором имена выдаются по /h, регистр букв не важен).

Обязательный ключ, никаких действий по умолчанию не производится.

Если задан только чип (больше нет ни ключей, ни имен файлов), то чип будет сброшен и выдана информация о нем (наличие, если есть доступные по ISP fuses — их состояние).

**-p<число1>[,<число2>]**

«число1», шестнадцатеричное.

set Port 1, 2, 3 — номер LPT-порта, >0x100 — базовый адрес порта, если задан номер — адрес из BIOS MEMORY (не работает для win32 и linux версий, там принято, что LPT1/2/3 это 378/278/3BC). Адрес в виде -p3BC и в виде -p0x3BC воспринимается как HEX число, если задан адрес — «сами попросили» — по умолчанию -p2 (LPT2).

«число2», с плавающей точкой, необязательный параметр, если указан, то не производится настройка на скорость порта, время обращения принимается равным <число2> микросекунд.

В win32 версии настройка не производится никогда, если не указано — считается, что обращение к порту не быстрее 1 мкс.

**-a<буква>** — группа ключей «адаптер»;

**-ab** — работа через Altera ByteBlaster;

- as — работа через адаптер Atmel STK\*00;
- aa — автоматический выбор ByteBlaster/STK;
- az — пытаться переводить выходы LPT в Z-состояние для «отключения» программатора от схемы («byte-blaster для ленивых»);
- ar — инверсия полярности сброса (например, для подачи его через резистор в базу присутствующего в схеме транзистора);
- ai# — установить время неактивного Reset после стирания равным #mS. Необходимо устанавливать большим времени задержки снятия сброса присутствующим в схеме с супервизором;
- ap — подавать питание чипа через свободные выходы данных LPT (несовместимо с -ab).

Рекомендуется поближе к чипу установить блокировочный конденсатор по питанию.

-o<число> задать частоту установленного кварца для настройки скорости SPI.

<число>=<частота в кГц>, по умолчанию 800 (0,8 МГц).

0 означает необходимость генерации XTAL из программы.

Работает со всеми адаптерами. У STK\*00 для генерации XTAL используется линия LED. Байт-бластер должен быть доработан, см. <http://www.ln.com.ua/~real/avreal/adapters.html#ABB>.

-n[<число>] — использовать последние 2 байта кода как счетчик стираний.

Если указано <число>, то для записи счетчика используется оно, а не инкрементированное прочитанное значение.

При -ewn если последние 2 байта заняты кодом, то -n игнорируется.

При -w если в чипе записан счетчик, а последние 2 байта заняты кодом, то производится насильное стирание.

-e[-] — стереть чип.

'-' задает сохранение содержимого EEPROM способом прочесть/стереть/записать. Даже для megaX03 применяется этот способ, если хочется пользоваться имеющимся fuse EESAVE, следует установить его (-fe) отдельным запуском avreal.

-1, -2, -3, -4, -5 — увеличить задержки на программирование в 1,5, 2, 3, 4, 5 раз соответственно (для программирования при пониженном напряжении).

-b — проверить на чистоту.

-r — прочитать из чипа в файл[ы].

-w[+] — записать в чип. Дополнительный '+' — если чип в этом задании не стирался и задана запись в EEPROM, то прописать FF в ячейки EEPROM, не заданные в HEX. Верификация этих ячеек на значение FF будет производиться, даже если был задан просто -v, а не -v+ (мы же эти ячейки писали).

-v[+] — верифицировать (только то, что есть в hex).

Дополнительный '+' вызывает проверку на FF «свободных» с точки зрения HEX-файла позиций.

-l[+]# -l# — установить уровень защиты # = 1 or 2.

-l# — то же самое, только lock биты прошиваются до зашивки fuses (необходимо для включения BODEN и залочки 90s4433 в одном цикле подачи питания).

-f<fuselist> — список fuses для тех чипов, в которых они шьются по ISP.

<fuselist> выглядит как fusename=value, fusename=value, alias, alias...

value — ШЕСТНАДЦАТЕРИЧНОЕ значение без префиксов и суффиксов.

Для однобитовых fuse добавлены значения ON и OFF (прошить и стереть, 0 и 1 соответственно).

alias'ы — это однобуквенные сокращения для fusename=value, соответствующие буквенным обозначениям в версиях avreal до 1.22.

При наличии «противоречивых» fusename=val и/или alias выдается сообщение об ошибке.

В целях совместимости с предыдущими версиями оставлена возможность перечисления alias-ов без разделения запятыми.

Перечень fusename с диапазоном value и комментарием, а также допустимые alias'ы для конкретного чипа можно получить при помощи avreal +chipname -?

Если работа с fuses по SPI поддерживается, то их состояние сообщается при любой операции с чипом, задание ключа -F необязательно.

OSCCAL (Tiny12, Tiny15, Mega163) рассматривается как Read-Only fuse ; -)

В случае наличия -F<fuselist> при -V производится верификация fuses.

Не упомянутые fuse остаются неизменными (кроме BLB для mega161, mega163, так как они могут быть стерты по -E).

AVReAl обеспечивает также расширенную поддержку osccal (см. ниже)

Группа BLB относится скорее к lock-битам, записать 1 поверх уже запрограммированной в 0 fuse невозможно. Если запрошена такая операция и чип не стирался, то программа завершает работу с соответствующим предупреждением.

Для tiny12, tiny15 поддерживается программирование SPIEN и RSTDSBL, будьте осторожны, SPIEN по умолчанию запрограммировано, ключ -fspien=1 (-fspien=off) \_сотрет\_ SPIEN и запретит дальнейшее низковольтное программирование. То же произойдет и при \_программировании\_ RSTDSBL (-frstdsbl=0 или -frstdsbl=0).

-% — выдавать по ходу дела дополнительную информацию (производимые действия и ответы чипа). Внутри .bat-файлов следует писать -%% (пожалуй, это был неудачный выбор — символ, имеющий особый смысл в bat-файлах, но меняться уже не будет).

-.! — делать, что велено, даже если чип не распознан (или распознан «не так», как задано в +<имя чипа>), а также, если не обнаружен указанный в командной строке адаптер. Попытка стереть нераспознанный (возможно, просто залоченный) чип при наличии команды стирания производится и без -.! после чего проводится повторное детектирование чипа.

Возможны осмысленные комбинации:

/vw — записать, проверить;

-e-wv+ — понятно;

-bw — проверить на чистоту, если не пустая — завершить работу, иначе записать.

После ключа `-d` идет имя файла данных, после `-c` — имя кода, в этом случае порядок файлов не важен. Пробел между ключами `-c/-d` и именами не обязателен, т. е. допустимо как `-cfoo.hex`, так и `-c foo.hex`.

Если дано два имени файла без ключей `-c` и `-d`, то первое имя — файл кода, второе — файл данных (EEPROM).

Если имя одно и нет ключей `-c/-d`, то этот файл содержит данные для кода, и если он содержит информацию после верхнего адреса FLASH для заданного кристалла, то эта информация используется для программирования EEPROM данных. То есть если, например, для 90s2313 hex-файл содержал данные в адресах от `0x800` до `0x87F`, то эти данные будут записаны в EEPROM по адресам `0x00-0x7F`.

Для Mega163 и Tiny12 поддерживаются особые формы ключей `-c` и `-d` для записи значения OSCCAL по адресу `offset` (шестнадцатеричное значение без префиксов и суффиксов). Байт (слово для `?`) по `offset` должен существовать в исходном HEX-файле. Используется адрес байта, а не слова, в том числе и при обращении ко flash коду.

`-d*osccal=offset` — как байт в EEPROM данных;

`-c*osccal=offset` — как байт во FLASH коде (под команду LPM);

`-c?osccal=offset` — врезать OSCCAL как часть команды LDI по `offset,offset+1`.

Например:

```
public osccal_ldi
...
osccal_ldi:
    ldi R16,0xFF
    out OSCCAL,R16
...
```

Далее смотрим по MAP-файлу значение `osccal_ldi`, например, оно вышло `0x120`

`avreal +tiny12 -ewv -c foo.hex -c?osccal=120`

Группа ключей может задаваться как одним аргументом (`-wv`)? так и раздельно (`-v -w`). Ключ с параметром (`-p -c -d`) может стоять в группе только последним. То есть допустимо, скажем, такое `:-) -wddata.hex -bv+c prog.hex -ep1`

Порядок ключей не важен, выполнение производится в порядке E B W V L.

Если проверка (B, V) дала отрицательный результат, то дальнейшая работа не производится. То есть -ebwvl2

Стереть, если стерлась — писать, если верифицировалась — за-  
ложить.

+tiny12 -w -c foo.hex -c\*osccal=1f3 -fcks=3,boden=0

Для tiny12 записать во флэш кода файл foo.hex, по адресу 0x1F3 занести прочитанное из чипа значение OSCCAL, запрограммировать fuse boden и установить fuse CKSEL в бинарное значение 0011.

*Возвращаемый код ошибки:*

0 — все заказанное сделалось;

10 — not blank при -B, несравнение при -V[+];

20 — невозможность осуществить запрошенную операцию HEX-кода или перечень данных слишком большой для распознанного чипа.

Запрошенное для записи состояние BLB нельзя записать, не стерев чип (а команды стирания не было);

30 — не распознан конкретный чип (бывает при защищенном чипе, поэтому при -E все равно делается попытка стереть, и если после этого тоже не распознан — программа завершает работу);

40 — адаптер не подключен (не удалось войти в программирование по алгоритму для At90s);

50 — ошибка при чтении (не найден, «не те» символы, не совпала контрольная сумма, ошибка чтения...) или записи (есть защищенный от перезаписи с таким же именем...) HEX-файла;

60 — недостаточно памяти для списков кода/данных;

70 — недопустимая командная строка (в том числе задан номер отсутствующего LPT).

При обнаружении ошибок просьба сообщать версию программы и очень желательно отослать копию расширенной выдачи программы по ключу -% (перенаправив выдачу в файл) по адресу real@real.kiev.ua.

Программы командной строки удобны для использования в какой-либо системе автоматизации проектирования программ. Большинство IDE имеют настраиваемое меню **TOOLS**, в которое можно вставить вызов программы и передать ей аргументы, например, имя текущего файла, имя проекта и др.

Многие программисты пользуются для сборки проекта утилитой **make** даже в том случае, если пакет имеет интегрированную оболочку, но содержит и компилятор/линкер командной строки.

Для использования **avreal** с такими программами необходимо один раз правильно сформировать строки для вызова **avreal** и поместить их в соответствующие позиции меню **TOOLS** используемого IDE или как команды соответствующих целей в **makefile**.

При использовании **avreal** для работы с кристаллами «вручную» неудобно каждый раз набирать всю командную строку. Для упрощения работы можно заготовить командные файлы с постоянно используемыми ключами и в командной строке задавать только название контроллера и имена файлов.

Ниже приведены примеры таких командных файлов, написанные в предположении, что:

- используется адаптер Atmel STK200 или STK300 (ключ **-as**), подключенный к порту LPT1 (ключ **-p1**). \*.bat-файлы должны находиться в одной папке (директории) с программой **avreal.exe**;
- тактовая частота контроллера не ниже 3,686 МГц (3686 кГц, ключ **-o3686**). Без этого ключа **avreal** будет предполагать, что тактовая частота может быть ниже, вплоть до 0,8 МГц, что приведет к существенному увеличению времени работы.

При использовании **avreal** для **win32** необходимо в командном файле указать **avreal32** либо переименовать **avreal32.exe** в **avreal.exe**.

-----  
erase.bat

```
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -e
@goto exit
:help
@echo erase.bat - стирание AVR-контроллера
@echo Вызов:
@echo erase cputype,
```

```
@echo где cruname - имя используемого процессора
:exit
```

```
-----
erasen.bat
```

```
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -en
@goto exit
:help
@echo erasen.bat - стирание AVR-контроллера
@echo с записью в верхние два байта flash-числа стираний
@echo Вызов:
@echo erasen cruname,
@echo где cruname - имя используемого процессора
:exit
```

```
-----
write.bat
```

```
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -w %2 %3
@goto exit
:help
@echo write.bat - запись AVR-контроллера
@echo Записывается flash кода и, если задан файл, EEPROM данных
@echo Вызов:
@echo write cruname codename dataname,
@echo где cruname - имя используемого процессора
@echo codename - имя файла кода
@echo dataname - имя файла данных (необязательный параметр)
:exit
```

```
-----
writed.bat
```

```
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -wd %2
@goto exit
:help
@echo writed.bat - запись AVR-контроллера
@echo Записывается только EEPROM данных
@echo Вызов:
@echo writed cruname dataname,
@echo где cruname - имя используемого процессора
@echo dataname - имя файла данных
:exit
```

```
-----
read.bat
```

```
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -r %2 %3
```

```
@goto exit
:help
@echo read.bat - чтение из AVR-контроллера
@echo Считывается flash кода и, если задан файл, EEPROM данных
@echo Вызов:
@echo read cруname codename dataname,
@echo где cруname - имя используемого процессора
@echo codename - имя файла кода
@echo dataname - имя файла данных (необязательный параметр)
:exit
```

```
-----
readd.bat
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -rd %2
@goto exit
:help
@echo readd.bat - чтение из AVR-контроллера
@echo Считывается только EEPROM данных
@echo Вызов:
@echo readd cруname dataname,
@echo где cруname - имя используемого процессора
@echo dataname - имя файла данных
:exit
```

```
-----
verify.bat
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -v %2 %3
@goto exit
:help
@echo verify.bat - верификация AVR-контроллера
@echo Проверяется flash кода и, если задан файл, EEPROM данных
@echo Вызов:
@echo verify cруname codename dataname,
@echo где cруname - имя используемого процессора
@echo codename - имя файла кода
@echo dataname - имя файла данных (необязательный параметр)
:exit
```

```
-----
verifyd.bat
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -vd %2
@goto exit
:help
@echo verifyd.bat - верификация AVR-контроллера
@echo Проверяется только EEPROM данных
@echo Вызов:
@echo verifyd cруname dataname,
```

```
@echo где cpubame - имя используемого процессора
@echo dataname - имя файла данных
:exit
```

-----

check.bat

```
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -bn
@goto exit
:help
@echo check.bat - проверка AVR-контроллера на чистоту (стертость)
@echo Если при стирании контроллера был использован файл erasen.bat,
@echo то будет сообщено число стираний контроллера, в противном
@echo случае будет сообщение "Number of erases not initialised",
@echo означающее, что верхние два байта flash тоже содержат 0xFF
@echo Вызов:
@echo verifyd cpubame dataname,
@echo где cpubame - имя используемого процессора
@echo dataname - имя файла данных
:exit
```

-----

lock.bat

```
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -l2
@goto exit
:help
@echo lock.bat - защита AVR-контроллера от чтения
@echo Вызов:
@echo lock cpubame
@echo где cpubame - имя используемого процессора
:exit
```

-----

lockv.bat

```
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -v12 %2 %3
@goto exit
:help
@echo lockv.bat - защита AVR-контроллера от считывания с предварительной
@echo верификацией (при несовпадении содержимого защита не устанавливается)
@echo Вызов:
@echo lockv cpubame codename dataname,
@echo где cpubame - имя используемого процессора
@echo codename - имя файла кода
@echo dataname - имя файла данных (необязательный параметр)
:exit
```

-----

fuses.bat

```
@if '%1'==' ' goto help
@avreal -as -p1 -o3686 +%1 -wvf%2
@goto exit
:help
@echo fuses.bat - запись fuses AVR-контроллера
@echo Вызов:
@echo fuses cprname fuselist
@echo где cprname - имя используемого процессора
@echo fuselist - перечисленные через запятую fuses, состояние
@echo которых следует изменить. Более подробно см.
@echo описание avreal
:exit
```

-----

### **Примечания.**

Имя используемого процессора задается так, как оно выглядит в списке поддерживаемых кристаллов при вызове *avreal -h*, например, *90s8515* или *mega103*. Допускается также указывать имена с префиксом 'AT', например, *at90s8515* или *atmega103*. Регистр букв значения не имеет.

Запись *fuses* достаточно произвести один раз, стирание кристалла их не стирает (за исключением *boot lock bits* в кристаллах с *boot sector во flash*, рассматриваемых программой *avreal* как *fuses*).

Не указанные в команде *fuses* не изменяются.

Записанное состояние *fuse* — '0', стертое — '1'. Для уменьшения вероятности ошибки следует пользоваться mnemonicскими обозначениями *ON* и *OFF* соответственно.

При записи *fuses* следует быть осторожным с изменением состояния *SPIEN* и *RSTDSDL* для тех кристаллов, у которых эти *fuse* доступны в режиме низковольтного программирования.

Контроллеры AVR поставляются с *SPIEN=0(ON)*, *RSTDSDL=1 (OFF)*. Эти значения необходимы для *low voltage ISP*.

### **Еще один вариант использования программатора**

Идея очень простая — обычно в процессе проектирования схемы и написания для нее программы тип чипа и название файла программы (т. е. и кода) выбираются один раз в начале работы и в дальнейшем изменяются очень редко.

В начале работы создается подобный файл с указанием типа кристалла (в этом файле *AT90S2313*) и имени файла, и при дальнейшей работе при необходимости запрограммировать кристалл просто запускается этот \*.bat-файл.

Кстати, запускать его можно не только из командной строки MS-DOS, но и из среды Windows 95/98. В последнем случае удобно сделать ярлык для этого файла.

При выполнении этого командного файла производятся следующие действия: проверяется наличие адаптера, совместимого с STK200, наличие подключенного микроконтроллера. Содержимое памяти микроконтроллера стирается, осуществляется проверка на успешное завершение операции стирания, после чего в кристалл записывается \*.hex-файл, указанный в командной строке (в этом примере test2313.hex):

```
avreal.exe +90s2313 -p378 -as -ebvw -c test2313.hex
```

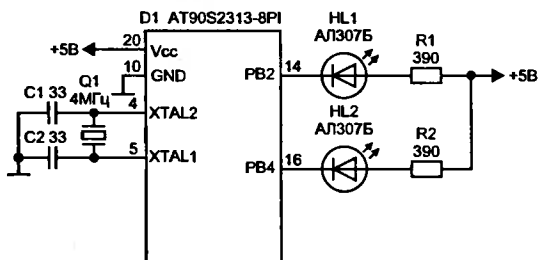


Рис. 4.15. Схема на микроконтроллере AT90S2313

Конкретный пример. Изготовьте простейшую конструкцию на микроконтроллере AT90S2313. Ее схема изображена на рис. 4.15.

Эта схема управляет двумя светодиодами, подсоединенными к линиям PB2 и PB4 микроконтроллера. Файл прошивки памяти программ для этой схемы имеется на компакт-диске, прилагаемом к книге. При правильной сборке и программировании схемы светодиоды должны поочередно мигать.

Для каждого проекта следует заводить отдельную папку (директорию), в которой располагать все файлы, относящиеся к проекту.

Итак, для того чтобы, например, занести код, находящийся в файле test2313.hex в память программ микроконтроллера AT90S2313, необходимо:

- 1) скопировать программу avreal.exe в папку проекта;
- 2) оттранслировать текст программы, чтобы получить файл \*.hex. В нашем случае файл test2313.hex имеется готовый на ком-

пакт-диске, прилагаемом к книге. Его следует скопировать его в папку проекта;

3) создать текстовый файл с именем `test2313.bat`, в котором должна быть строка

```
avreal.exe +90s2313 -p378 -as -ebvw -c test2313.hex
```

4) подключить программатор к порту LPT и разъему ISP (программирования) схемы на микроконтроллере AT90S2313;

5) включить питание схемы на микроконтроллере;

6) запустить файл `test2313.bat`.

Обратите внимание — файлы `avreal.exe`, `*.bat` (в нашем примере `test2313.bat`) и `*.hex` (в нашем случае `test2313.hex`) должны находиться в одной папке.

Если все было сделано верно, программа должна быть занесена в микроконтроллер, и светодиоды начнут мигать.

Если необходимо заносить код не только в память программ, но и в EEPROM, в конце строки в `*.bat`-файле следует добавить `-d` и название файла с прошивкой EEPROM.

## **Глава 5.**

# **Основные схемные решения интерфейсов**

### **5.1. Параллельные выходы**

Одним из наиболее простых, но одновременно и наиболее важных и частых применений параллельных портов микроконтроллера можно назвать управление различными устройствами. В данном случае речь пойдет об управлении типа «включить/выключить».

В качестве выходов параллельные порты могут применяться для управления реле, симисторами, светодиодными индикаторами и т. д.

#### **Управление светодиодами или оптронами**

Управление светодиодами — самое простое, что может встретиться при изготовлении схем на микроконтроллерах. Как известно, светодиоды потребляют достаточно маленький ток — в зависимости от типа светодиода этот ток может составлять от 3 до 20 мА. Рабочее напряжение светодиодов составляет примерно от 1,5 до 4 В.

Так как ток, который микроконтроллеры семейства AVR могут отдавать при напряжении «логический ноль» на выходной линии, может достигать 20 мА, можно управлять светодиодом просто подключив его к выходной линии порта последовательно с ограничивающим ток резистором. Второй вывод этой цепочки следует подсоединить к положительной линии питания.

Стоит обратить внимание на то, что подключать следует именно таким образом — при напряжении «логическая единица» микроконтроллер может отдавать гораздо меньший ток. А значит, его нельзя будет применить для управления светодиодом напрямую. Более подробно можно узнать величины допустимых токов, воспользовавшись фирменной документацией на микроконтроллеры.

Управлять светодиодом предельно просто: так как один его вывод подключен к положительному проводу питания, для того, чтобы он стал светиться (т. е. падение напряжения на нем стало достаточным для зажигания), нужно сформировать на втором выводе цепочки со светодиодом напряжение низкого уровня «0». Говоря проще, для того, чтобы зажечь светодиод, надо записать в выходной порт значение «0». Чтобы погасить — записать «1».

На рис. 5.1 изображена простая схема с двумя светодиодами.

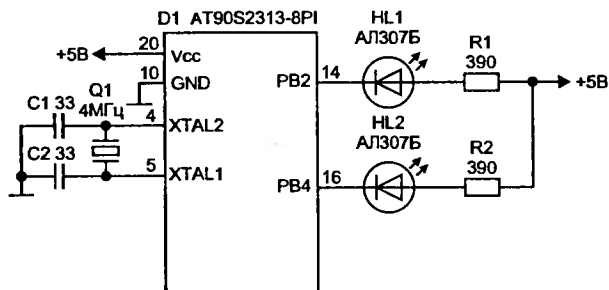


Рис. 5.1. Простейшая схема для управления двумя светодиодами

Таким же образом можно подсоединить и большее количество светодиодов — вплоть до того, что ко всем линиям портов ввода/вывода. Однако следует иметь в виду очень важный факт — хотя каждый выход микроконтроллера может управлять нагрузкой до 20 мА, общий потребляемый ток от всех линий портов ввода/вывода не должен превышать определенного значения. В зависимости от типа корпуса микроконтроллера и числа его линий портов ввода/вывода его величина может быть различной. Точно это значение можно узнать в фирменной документации на микроконтроллер.

Например, для микроконтроллера AT90S2313 имеются следующие ограничения: суммарный ток нагрузки при «0» на выходах не должен превышать 200 мА, причем суммарный ток линий D0—D5 не более 100 мА и суммарный ток линий B0—B7 и D6 также не должен превышать 100 мА. Легко увидеть, что если нагрузить все выходы по 20 мА, то можно превысить допустимый ток, что может повредить микросхему.

Аналогично можно управлять оптопарами, ведь по-существу, они представляют собой размещенные в одном корпусе напротив друг друга светодиод и фоточувствительный элемент — фоторезистор, фототранзистор, и т. д. Например, используя оптопару со встроенным фотосимистором можно управлять высоковольтной нагрузкой. При этом достигаются такие важные цели, как гальваническая развязка высоковольтных цепей и схемы управления, отсутствие искрового промежутка.

### Управление реле

Для питания обмотки реле требуется ток, превышающий 20 мА, поэтому напрямую подключить к микроконтроллеру его нельзя. Для управления реле, можно применять простейший усилитель — транзисторный ключ. На рис. 5.2 показан пример схемы с реле. Обратите внимание на наличие диода, подключенного параллельно обмотке реле — он нужен для защиты схемы от ЭДС самоиндукции, появляющейся в процессе коммутации обмотки.

Совершенно аналогично можно включать не реле, а какую-либо другую нагрузку, например, лампу накаливания и т. д.

В случае, если необходимо управлять большим числом реле, или других мощных нагрузок, удобно применять микросхемы ULN2003 или ULN2803. Эти микросхемы содержат соответственно, 7 и 8 транзисторных ключей на составных транзисторах (схема Дарлингтона). Они позволяют управлять нагрузкой до 500 мА при напряжении до

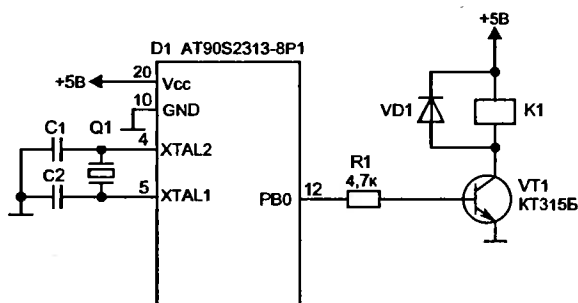


Рис. 5.2. Использование реле

50 В. При этом входы этих микросхем можно подключать непосредственно к линиям портов ввода/вывода микроконтроллера. Внутри микросхем уже имеется встроенный защитный диод, который можно подключать или отключать, осуществляя внешние соединения. На рис. 5.3 показан пример схемы с использованием микросхемы ULN2003.

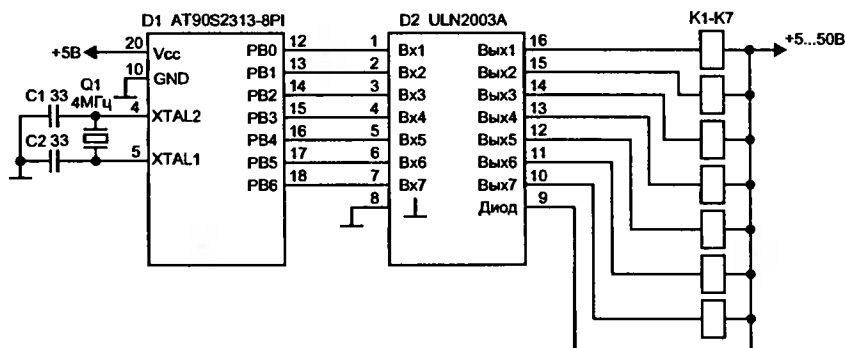


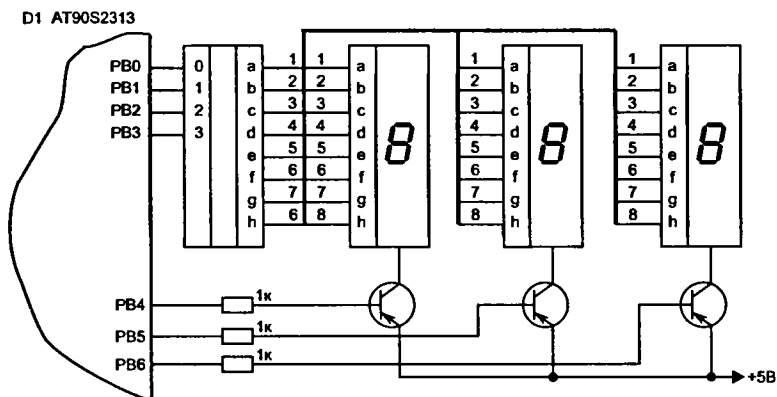
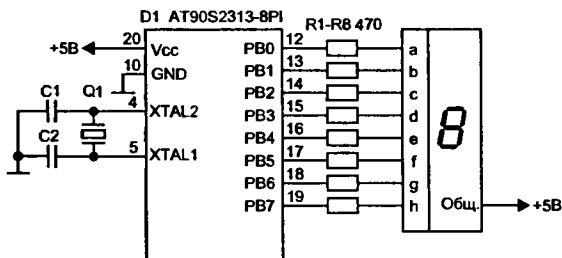
Рис. 5.3. Применение микросхемы ULN2003

Для включения нагрузки следует сформировать на соответствующем выводе микроконтроллера уровень «1». При этом ток, потребляемый от вывода порта микроконтроллера, не превышает допустимый, в то же время, как осуществляется управление достаточно мощной нагрузкой.

### Управление светодиодными цифровыми индикаторами

Так как светодиодные цифровые индикаторы, по-существу, представляют собой набор светодиодов специальной формы, расположенные так, чтобы при зажигании различных их комбинаций, получались цифры, управление ими принципиально не отличается от управления отдельными светодиодами. На рис. 5.4 изображен пример схемы управления семисегментным светодиодным индикатором.

Легко увидеть, что если потребуется управлять большим числом индикаторов, количества выводов портов ввода/вывода будет недостаточно. Для преодоления этого препятствия применяется динами-



ческая индикация. На рис. 5.5 показан пример схемы динамической индикации. Идея, лежащая в основе работы этой схемы очень проста — человеческий глаз достаточно инерционен, поэтому можно зажигать не все индикаторы одновременно, а только один из них, потом через короткое время другой и так далее. Так как переключение индикаторов происходит достаточно быстро, человеку кажется, что все индикаторы горят.

## 5.2. Параллельные входы

Параллельные входы обычно применяются для контроля состояния различных коммутационных элементов: кнопок, переключателей, блока контактов и т. д. Также можно проверять состояние неко-

торых видов датчиков, но при этом может потребоваться дополнительная схема, преобразующая состояние датчика к логическим уровням (например, уровень воды в баке ниже или выше определенной высоты и т. д.). Очень часто входы параллельных портов применяются для контроля состояния кнопок управления устройством.

### Кнопки и переключатели

Проверять состояние кнопок или выключателей достаточно просто. Достаточно подсоединить, например, кнопку одним выводом к общему проводу, а другим — ко входной линии порта ввода/вывода, настроенной для работы в режиме чтения. Также эта линия должна быть соединена через резистор сопротивлением примерно 4,7...100 кОм с проводом «+» питания. При большем сопротивлении меньше суммарный потребляемый ток.

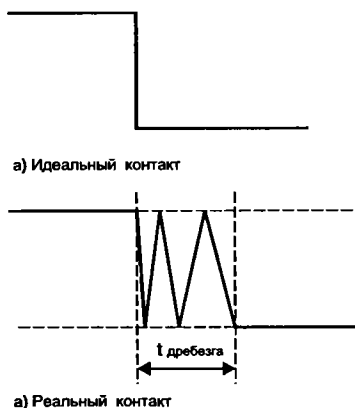


Рис. 5.6. Явление «дребезг контактов»

При разомкнутых контактах, на соответствующем выводе микроконтроллера будет «1», при замыкании контактов — «0».

Все механические выключатели имеют недостаток — при работе с ними наблюдается так называемый дребезг контактов, при котором при нажатии на кнопку происходит много замыканий и размыканий контактов из-за того, что они как правило, пружинят. Длительность периода дребезга зависит от качества контактов и обычно составляет

от 10 до 100 мс. Бороться с этим эффектом проще программным способом. На рис. 5.6 показаны графики, иллюстрирующие дребезг контактов, а на рис. 5.7 приведена простейшая схема с кнопкой.

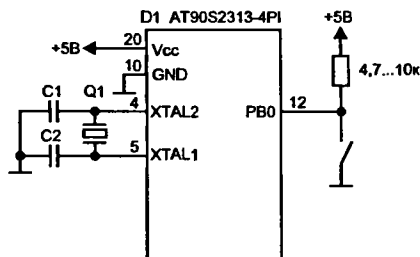


Рис. 5.7. Подключение кнопки к микроконтроллеру

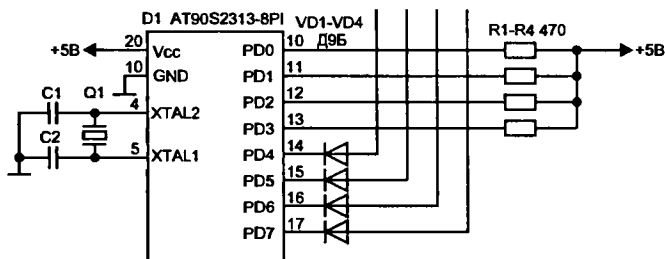


Рис. 5.8. Использование матричной клавиатуры

Для реализации большого числа кнопок управления используют матричную схему соединения клавиатуры. Пример подобной схемы изображен на рис. 5.8.

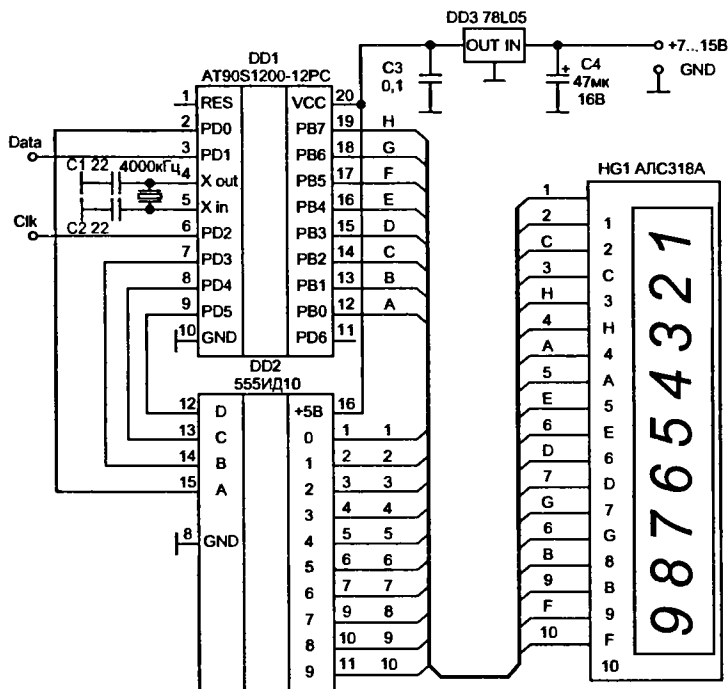
## **Глава 6.**

# **Практические примеры применения микроконтроллеров AVR**

### **6.1. 10-разрядный светодиодный индикатор на AT90S1200**

Данное устройство предназначено для замены ЖК-индикатора со встроенным контроллером, аналога HT1611, применяемого в телефонных аппаратах, на светодиодный индикатор с общим катодом. При применении десяти знаков ЖК-индикатора реализуется полная совместимость с HT1611 (за исключением режима часов и таймера). Частота мерцания индикатора близка к 100 Гц, что практически незаметно даже боковым зрением. Ток через сегмент ограничен 20 мА самим микроконтроллером, что позволяет выровнять свечение при разном количестве зажженных сегментов. При программировании контроллера есть возможность постоянно зажечь запятые в нужных разрядах и исключить неиспользуемый знак при применении 9-знакового ЖКИ (который показан на схеме).

Сигналы Clk и Data должны соответствовать питанию 5 В, т. е. логическая единица должна быть 3...5 В. После проверки правильности монтажа подаем и проверяем питание +5 В на 20-м выводе панельки под DD1. Вставляем микроконтроллер — на экране после включения питания появятся 1234567890, что записано в программе. Если нет — проверить колебания с частотой 4,000...4,005 МГц на выводе 4. При подаче внешних сигналов Clk и Data индицироваться будет входящая информация. Общее потребление — 65...250 мА. Микросхему AT90S1200-12PC можно заменить на AT90S1200-12PI, а также AT90S1200-4PC и AT90S1200-4PI. Светодиодный индикатор — АЛС318А, ТОТ — 3361 AG-1 (три штуки), от АОНа или любой другой с общим катодом, можно набрать из одиночных, сдвоенных, строенных и т. п.



**Рис. 6.1. Электрическая принципиальная схема**

Программа для AT90S1200: lcd.zip, находится на компакт-диске, прилагаемом к книге.

Автор: Дергаев Э. Ю., UA4NX (E-mail: ua4nx@ezmail.ru).

## 6.2. Управление синтезатором частоты радиостанции «Маяк»

Данное устройство позволяет управлять частотой радиостанции «Маяк» в диапазоне 144,5—146,0 МГц. В режиме репитера и антирепитера индицируется частота передачи. Программа хранит в энергонезависимой памяти 63 частоты каналов и одну VFO, включая репитерный разнос +600 кГц, антирепитерный разнос — 600 кГц, с шагом перестройки 25 кГц. Запись частот в каждую ячейку памяти гарантируется 100 000 раз. В режиме SCAN происходит сканирование с

53-го по 63-й канал памяти, в режиме DUAL — сканирование между любым каналом памяти и VFO. При понижении напряжения источника питания на индикаторе отображаются прочерки. При выключении питания или нажатии клавиши CLOCK индикатор переходит в режим часов. Подтверждением нажатия клавиш служит короткий звуковой сигнал высокого тона.

Принципиальная схема на рис. 6.2. Микроконтроллер DD1 AT90S2313 управляет работой синтезатора радиостанции, передавая в блок приемопередатчика на DD4 и DD5 последовательный код частоты, анализирует состояние клавиатуры, нажатие тангенты PTT, состояние шумоподавителя BUSY, напряжение питания радиостанции и передает информацию на жидкокристаллический 10-разрядный индикатор DD3 HT1611. Питание стабилизировано DD2 78L05, напряжение — 5 В. Для питания ЖКИ используется отдельный источник питания +1,5 В, (для установки и коррекции времени используются кнопки S1 и S2). Питание на DD4 и DD5 подается с блока синтезатора +9 В, туда же желательно установить и плату с ними. Вариант 1 соответствует синтезатору на трех K561IE11. Вариант 2 — синтезатор с дополнительной платой на K155PE3. Если у вас другой синтезатор, установите перемычками 125,000 МГц и подключите остальные шины по «весу». Сигналы с контроллера подаются по кабелю, соединяющему пульт управления и приемопередатчик, используются свободные A0, A1, A2 (последним можно продублировать «землю»). Сам контроллер встраивается в пульт управления, питание +13,8 В. Сигнал BUSY снимается с D16 K561IA7 вывод 4, сигнал TX/RX с переключателя S8 (выводы на схеме 55). Выход SOUND подключается к собственному динамику (телефон из трубки) или УНЧ-приемника, регулируется подбором R6 (не забывайте, что УНЧ блокируется шумоподавителем). Кнопки управления — любые. Индикатор — любой аналог HT1611, используемый в АОНах. Так как индикатор имеет горизонтальное положение, то устанавливая его придется на корпус сверху.

### Режим VFO. На индикаторе F 145025

Нажимая H или L, на один шаг увеличиваем или уменьшаем частоту.

При их удержании частота начинает изменяться с шагом 100 кГц.

Нажимая S/D, включаем режим DUAL.

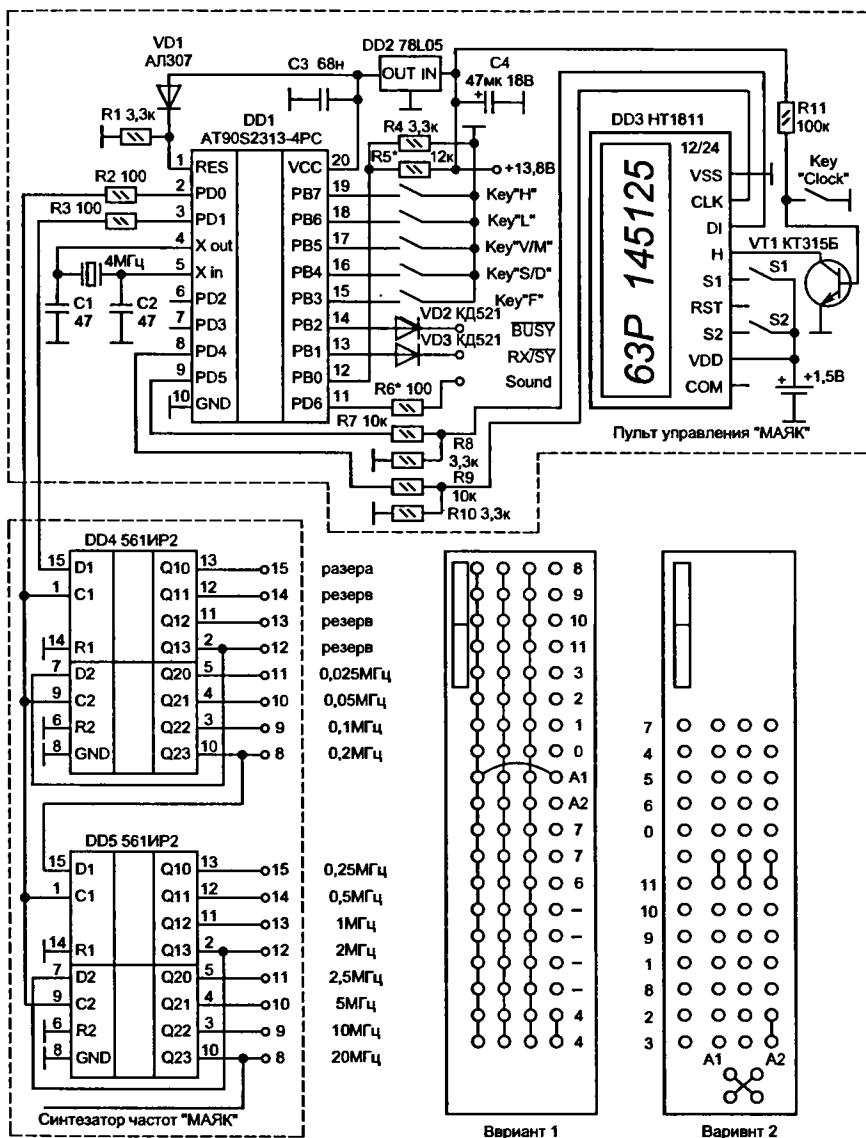


Рис. 6.2. Электрическая принципиальная схема управления синтезатором частот на AVR микроконтроллере фирмы ATME2

Нажимая F, переходим в режим **SERVIS**.

При выключении питания частота VFO сохраняется.

### **Режим MEMORY. На индикаторе 1P 145750**

Нажимая H или L, увеличиваем или уменьшаем канал памяти на один.

При удержании клавиш — непрерывное изменение номера канала.

Нажимая S/D, включаем режим **SCAN**.

Нажимая дважды F, записываем в память частоту, набранную в VFO.

### **Режим SERVIS**

Нажимая H или L, переходим между:

S - - - режим частота RX — TX совпадает;

S P — режим репитера;

S P - - режим антирепитера.

При включении питания устанавливается режим совпадения RX — TX. Для включения режима репитера необходимо выйти в **SERVIS**, выбрать режим, например репитер, и выйти из **SERVIS**, нажав F. То есть при нажатии на F выбранная на экране информация записывается в память.

### **Изменение частоты**

Перейдем в режим VFO. Нажимая H или L, на один шаг увеличиваем или уменьшаем частоту на заданный шаг. Затем можно включить режим репитера, антирепитера, перейти в режим **MEMORY**, выбрать канал и, нажав F, два раза записать содержимое VFO в память.

### **Режим SCAN**

Сканируются каналы памяти 53 — 63. При появлении несущей сканирование приостанавливается, а при пропадании — через 3 секунды возобновляется. Для выхода из режима надо нажать любую клавишу или PTT.

### **Режим DUAL**

Сканируются канал памяти и VFO. При появлении несущей сканирование приостанавливается, а при пропадании — через 3 секунды возобновляется. Для выхода из режима надо нажать любую клавишу или PTT.

## Режим LOCK

В режиме, передачи нажимая на Н, блокируем клавиатуру. Для снятия блокировки нажимаем на «L» в режиме передачи.

## Настройка

После проверки правильности установки всех деталей подается питание. Уровень срабатывания индикации пониженного напряжения — R5, на выводе 12 DD1 порог около 2,5 В.

## Возможная замена деталей

DD1 AT90S2313-4PC, AT90S2313-4PI, AT90S2313-10PC, AT90S2313-10PI. Перечисленные типы микроконтроллеров отличаются максимальной тактовой частотой (соответственно 4 и 10 МГц). Так как в схеме применен кварцевый резонатор на частоту 4 МГц, для схемы годится любой из них. VD1 — любой красный светодиод, можно уменьшить R1 до 470...910 Ом и вывести светодиод на переднюю панель как индикатор питания. VD2, VD3 — любые диоды. Кварц в пределах 2000...6000 кГц, но желательно ближе к 4 МГц, резисторы с допуском 20 %.

Программа для AT90S2313 находится на компакт-диске, прилагаемом к книге.

## Пособие по отладке

1. Монтируем плату микроконтроллера (без 561IP2). Достаем из панельки контроллер, если он был вставлен. Включаем «Маяк». Измеряем напряжение тестером. На ножке 1 должно быть 3,45...3,5 В. Светодиод должен быть красным. На ножке 20 должно быть 5 В. Выключаем питание. Вставляем в панельку контроллер. Включаем питание. Слышим телефонную трель в излучателе, подключенному к Sound. На индикаторе появляются 0-00 — это таймер, а затем первый канал памяти (1P 145125, например). Проверяем все кнопки (на них должно быть 5 В, а при нажатии — 0 В). Проверяем работу программы, набирая кнопками частоты, и т. п.

2. Если все нормально, монтируем плату с 561IP2, соединяем с контроллером и синтезатором. Питание на 561IP2 должно быть 9,0 В, а лучше его понизить на 0,7 В, подав питание через диод в прямом включении. В некоторых синтезаторах напряжение на плате су-

щественно выше, так что придется поставить несколько диодов или светодиод. Далее можно пройти по всем частотам и убедиться в правильности распайки синтезатора.

3. Если что-то не так, проверьте на 4-м выводе контроллера высокоомным щупом с делителем осциллографа наличие 4 МГц прямоугольников.

4. Если вы прошиваете микроконтроллер в плате — отключите светодиод на время программирования с компьютера.

Автор: Дергаев Э. Ю., UA4NX (E-mail: ua4nx@ezmail.ru).

### **6.3. Синтезатор частоты для УКВ ЧМ-радиостанции**

Данное устройство позволяет управлять частотой радилюбительской радиостанции в диапазоне 144...146 МГц. В режиме репитера и антирепитера индицируется частота передачи. Программа хранит в энергонезависимой памяти 31 частоту каналов и одну VFO, включая репитерный разнос +600 кГц, антирепитерный разнос -600 кГц, с шагом перестройки 5, 10 и 25 кГц. Запись частот в каждую ячейку памяти гарантируется 100 000 раз. В режиме SCAN происходит сканирование с 21-го по 31-й канал памяти, в режиме DUAL — сканирование между любым каналом памяти и частотой VFO. При понижении напряжения источника питания на индикаторе отображаются прочерки. Для индикации «захвата» синтезатора служит светодиод VD4. При выключении питания или нажатии клавиши CLOCK индикатор переходит в режим часов. Подтверждением нажатия клавиш служит короткий звуковой сигнал высокого тона. Возможна блокировка клавиатуры.

Принципиальная схема на рис. 6.3. Микроконтроллер DD2 AT90S1200 управляет работой синтезатора DD1 1508ПЛ1, переключает приемный (VT2) и передающий (VT5) генератор, управляемый напряжением (ГУН), анализирует состояние клавиатуры, нажатие тангенты РТТ, состояние шумоподавителя BUSY, напряжение питания радиостанции и передает информацию на жидкокристаллический 10-разрядный горизонтальный индикатор DD4 HT1611. Питание стабилизировано DD3 78L05, напряжение — 5 В. Для питания

ЖКИ используется отдельный источник питания +1,5 В (для установки и коррекции времени используются кнопки S1 и S2). В режиме передачи ГУН TX работает на частоте 144,5...145,8 МГц, модуляция подается на вход MOD, а в режиме приема ГУН RX — на частоте 133,3...135,3 МГц, для ПЧ — 10,7 МГц. В случае другой ПЧ необходимы изменения в программе (обращайтесь к автору).

Через буферные каскады VT4 и VT6 сигнал с ГУНов поступает далее. Напряжение на них подается через 1 кОм для приемного (VT4) и 330 Ом для передающего (VT6) с соответствующих коммутируемых шин питания приемника и передатчика, сигнал снимается через емкость 47...68 пФ.

Остановимся на работе программы. Существует два основных режима работы — VFO и MEMORY. Они переключаются соответствующей клавишей V/M.

### **Режим VFO. На индикаторе F 145025**

Нажимая H или L, на один шаг увеличиваем или уменьшаем частоту.

При их удержании частота начинает изменяться с шагом 100 кГц.

Нажимая S/D, включаем режим DUAL.

Нажимая F, переходим в режим SERVIS.

При выключении питания частота VFO сохраняется.

### **Режим MEMORY. На индикаторе 1P 145750**

Нажимая H или L, увеличиваем или уменьшаем канал памяти на один.

При удержании клавиш — непрерывное изменение номера канала.

Нажимая S/D, включаем режим SCAN.

Нажимая дважды F, записываем в память частоту, набранную в VFO.

### **Режим SERVIS**

Нажимая H или L, переходим между:

S - - - режим частота RX — TX совпадает;

S P — режим репитера;

S P — режим антирепитера;

S 5 — шаг 5 кГц;

S 10 — шаг 10 кГц;

S 25 — шаг 25 кГц.

При включении питания устанавливается шаг 25 кГц и режим совпадения RX — TX. Для изменения шага и включения режима репитера необходимо выйти в **SERVIS**, выбрать шаг, например 5 кГц, выйти из **SERVIS**, нажав F, затем снова зайти в **SERVIS**, выбрать режим, например репитер, и выйти из **SERVIS**, нажав F. То есть при нажатии на F выбранная на экране информация записывается в память.

### Изменение частоты

Перейдем в режим **VFO**. Нажимая H или L, на один шаг увеличиваем или уменьшаем частоту на заданный шаг. Затем можно включить режим репитера, антирепитера, перейти в режим **MEMORY**, выбрать канал и, нажав F два раза, записать содержимое **VFO** в память.

### Режим SCAN

Сканируются каналы памяти 21—31. При появлении несущей сканирование приостанавливается, а при пропадании — через 3 секунды возобновляется. Для выхода из режима надо нажать любую клавишу или **PTT**.

### Режим DUAL

Сканируются канал памяти и **VFO**. При появлении несущей сканирование приостанавливается, а при пропадании — через 3 секунды возобновляется. Для выхода из режима надо нажать любую клавишу или **PTT**.

### Режим LOCK

В режиме передачи, нажимая на H, блокируем клавиатуру. Для снятия блокировки нажимаем на L в режиме передачи.

### Настройка

После проверки правильности установки всех деталей подается питание. Подключается частотомер к выводу 7 DD1. Конденсатором C13 выставляется частота 4 000 000 Гц, с точностью до герца. Подключаются выходы буферов ГУНов к радиостанции и к частотомеру.

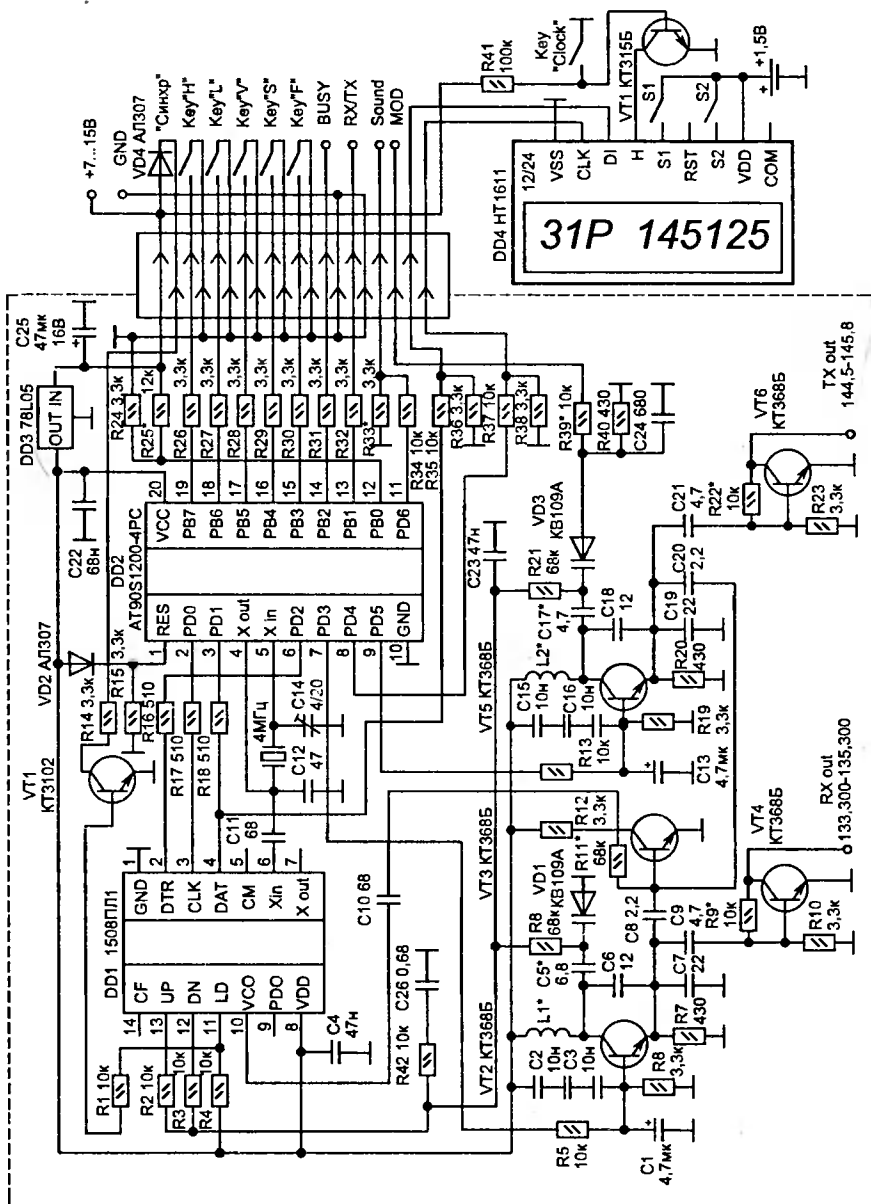


Рис. 6.3. Электрическая принципиальная схема синтезатора частот УКВ радиостанции

Контурами L1 и L2 ГУНЫ вгоняются в диапазон, емкостями C5, C17 достигается перекрытие 2,0...2,8 МГц. На коллекторе VT4 желательно иметь 2,5...3 В — R12. Уровень модуляции — R39. Уровень срабатывания индикации пониженного напряжения — R25, на выводе 12 DD2 порог около 2,5 В. Выход SOUND подключается к УНЧ-приемника, регулируется R33, при данных номиналах R33, R34 — размах 1,5 В. Уровни сигналов:

BUSY — «1» — несущая в канале, «0» — нет несущей;

RX/TX — «1» — режим приема, «0» — режим передачи.

Уровни, подаваемые на BUSY и RX/TX, не должны превышать 5 В!!! Катушки L1 и L2 бескаркасные, содержат 4...6 витков провода ПЭВ или ПЭЛ диаметром 0,4...0,8 мм и намотаны на оправке диаметром 4,5 мм. После настройки они заливаются прозрачным термоклеем. Подстройка после заливки производится паяльником, разогревая и сдвигая крайний виток. Также заливаются C5—C9, C17—C21, C11, C12, C14 и кварц.

Замена DD1 — 1508ПЛ11, 1508ПЛ11А; DD2 — AT90S1200-4PI, AT90S1200-12PC, AT90S1200-12PI, а также AT90S2313-4PC, -4PI, -10PC, -10PI, у последней число каналов памяти 63 + 1 VFO, KB109A — на KB132.

Плата должна экранироваться, для чего паяется коробочка из жести высотой 20...30 мм.

Данное устройство встраивается в радиостанцию, поэтому не имеет собственных защит от переполюсовки и перенапряжений, будьте внимательны!!!

Программы для AT90S1200 и для AT90S2313 находятся на компакт диске, прилагаемом к книге.

Автор: Дергаев Э. Ю., UA4NX (E-mail: ua4nx@ezmail.ru).

## 6.4. Телеграфный манипулятор

Для передачи азбуки Морзе в небольших объемах (10—20 групп), можно применить данный манипулятор. Он может манипулировать CW-передатчик и передавать звуковой тон в SSB- и FM-передатчиках. Рассмотрим принципиальную схему на рис. 6.4.





сигнал 5000 кГц от поверенного промышленного частотомера, подбирают коррекцию и подстраивают С2. Значение коррекции записывается в EEPROM автоматически. Нажимая S1 или S2, изменяют предел измерения в сторону увеличения или уменьшения соответственно (одну кнопку можно не ставить).

Перспективы: переход на универсальный индикатор МЭЛТ MT10T7-7, появится возможность устанавливать его вертикально, что сделает возможным доработку до частотомера — шкалы с записью ПЧ, увеличения верхней частоты до 40 МГц (добавлением одной микросхемы), возможностью подключения внешних делителей с учетом их коэффициентов деления.

Общее потребление — 10...15 мА.

Микросхему AT90S1200-4PC можно заменить на AT90S1200-4PI, AT90S1200-12PC, AT90S1200-12PI.

Программа для AT90S1200 имеется на компакт диске, прилагаемом к книге (файл).

Автор: Дергаев Э. Ю., UA4NX (E-mail: ua4nx@ezmail.ru).

## 6.6. Частотомер (шкала для UW3DI)

Данное устройство предназначено для измерения частоты ГПД (5500...6000 кГц) и отображения реальной частоты трансивера с учетом 1-го и 3-го гетеродинов. Частота измеряется 10 раз в секунду. Точность измерения — 20 Гц, причем программным способом полностью устранено «перескакивание» младшего и других знаков. Для переключения диапазонов (их может быть 12 плюс режим частотомера) соответствующие выводы микроконтроллера замыкаются на корпус галетным переключателем (рис. 6.6).

Монтаж производится на макетной плате, под микроконтроллер устанавливается панелька. Плата помещается в полностью экранированный корпус, например из жести. Все соединяющие провода — экранированные (в том числе и идущие на переключатель диапазонов). Транзистор VT2 (вместе с R11, R12 и C8) выносится к ГПД. Так как индикатор предназначен для горизонтального положения, на переднюю панель его разместить проблематично.

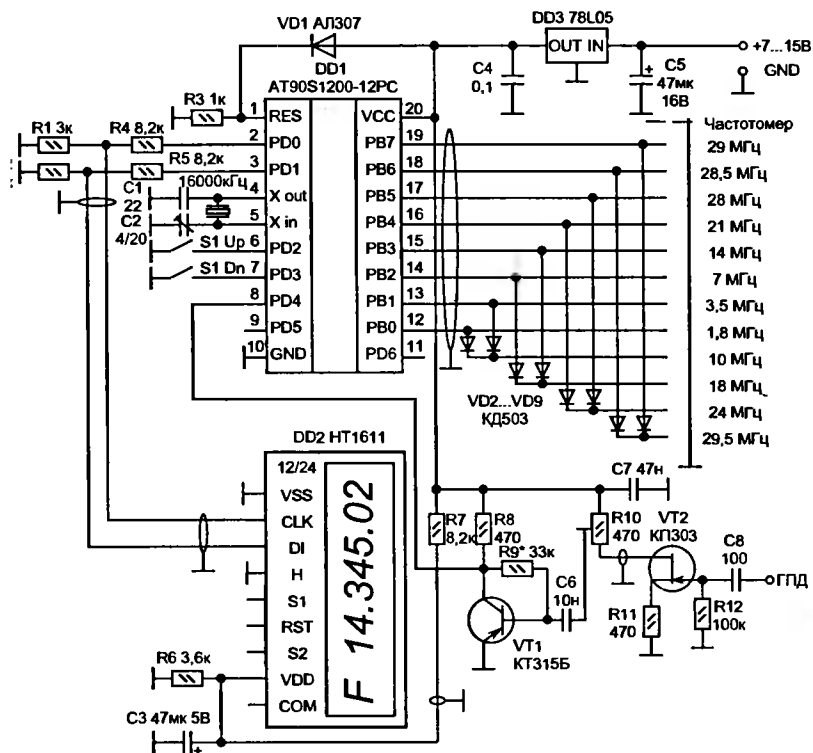


Рис. 6.6. Принципиальная электрическая схема частотомера

### Настройка

Проверяется напряжение на 20-м выводе контроллера, должно быть +5 В, на первом +3,5 В ( $\pm 0,2$  В), на VDD индикатора +1,5...1,6 В. Напряжение на коллекторе VT1 должно быть 2,5 В, подбирается резистором R9. Вставляем контроллер, если нет ошибок в монтаже и исправен кварц — шкала должна работать. Если нет — проверить колебания частотой 16,000...16,012 МГц на выводе 4. Подстроечный конденсатор C2 должен быть в среднем положении. Далее подаем на вход формирователя шкалы сигнал частотой 5000 кГц, с любого промышленного частотомера, предварительно прогрев его и шкалу. Переводим шкалу в режим частотомера. Нажимая клавиши Up и Down, добиваемся показаний как можно ближе к истине, затем

точно корректируем подстроечным конденсатором С2. (Если показания неустойчивые — регулируем усиление резистором R10). Отключаем генератор. Подключаем вход формирователя к выходу ГПД. Уровень сигнала с ГПД регулируется резистором R10. На коллекторе VT1 добиваемся синусоиды размахом 3...4,5 В, во всем диапазоне перестройки. Далее подстраиваем частоты на всех диапазонах: подключаем промышленный частотомер к выходу передатчика, слабо связав его с антенной или через нагрузку и аттенюатор. Даем несущую и, сравнивая показания шкалы и частотомера, нажатием клавиш Up и Down добиваемся совпадения показаний. И так на всех диапазонах. (Можно использовать «не круглые» кварцы в первом гетеродине). Детали: VD1 любой красный светодиод, падение на нем должно быть 1,5 В. Диоды VD2...VD9 — любые, монтируются прямо на переключателе навесным монтажом, если данных диапазонов в трансивере нет, то и ставить их не надо. С1 — желательно марки КМ, КД, КТ1; С2 — КТ4-21. Кварц на 16 000 кГц, а не 16 МГц, последний, скорее всего, гармониковый, может запустится на 1-й гармонике. Кварц жестко закрепить или приклеить силиконовым термолеем. Микросхему AT90S1200-12PC можно заменить на AT90S1200 - 12PI. Кнопки S1 и S2 — любые, расположены прямо в корпусе шкалы, так как требуются только для корректировки.

Микросхему AT90S1200-4PC можно заменить на AT90S1200-4PI, AT90S1200-12PC, AT90S1200-12PI.

Диапазон, МГц	Кварц, кГц	Диапазон, МГц	Кварц, кГц
1,8	8 000	21	15 000
3,5	10 000	24	18 850
7	13 500	28	22 000
10	16 500	28,5	22 500
14	8 000	29	23 000
18	12 000	29,5	23 500

Так как практически первый гетеродин не работает ниже 7 МГц, то в диапазоне 10 МГц используется не верхняя боковая (кварц 4 МГц), а нижняя, для телеграфного диапазона это не существенно.

Общее потребление — 10...15 мА.

Программа для AT90S1200 находится на компакт-диске, прилагаемом к книге.

Автор: Дергаев Э. Ю., UA4NX (E-mail: ua4nx@ezmail.ru).

## 6.7. Книгочей

Разработка этого устройства была вызвана желанием читать книги в электронном виде не только дома, с помощью настольного компьютера, но и в прочих местах — там же, где можно прочесть и обычную бумажную книгу.

Устройство предназначено для прочтения текстовой информации, записанной по COM-порту во flash-память AT49F040, на алфавитно-цифровом индикаторе с контроллером HD44780 фирмы Hitachi и русифицированным знакогенератором. Размер памяти 1 Мбайт, это позволяет записать 8 книг с возможностью оставить закладку на каждую книгу. Также предусмотрена возможность пролистывания книги — на случай, если закладка не была положена, а питание устройства было отключено. Выбор функций и чтение происходит по нажатию трех клавиш: выбор, вверх и вниз. Размер дисплея 40 × 2 строчки, что, как показала практика, вполне достаточно для восприятия смысла написанного.

Схема устройства приведена на рис. 6.7. «Сердцем» его является микроконтроллер AT90S8515, управляющий шинами адреса-данных, принимающий команды от компьютера и выводящий на индикатор. Для защелкивания младших 8 битов адреса служит регистр KP1533IP33 (аналог 74ALS573). Для преобразования уровней RS-232C служит также хорошо известная микросхема ADM202 фирмы Analog Devices.

Не связанное с компьютером устройство может только считывать записанную в него книгу. Для стирания памяти либо записи в нее надо выбрать пункт меню «Связь с ЭВМ», непосредственно затем послать либо число 0x5e, либо 0xe5. В первом случае устройство перейдет в командный режим, во втором — в отладочный, когда принимаемый от компьютера символ будет просто отображаться на экране.

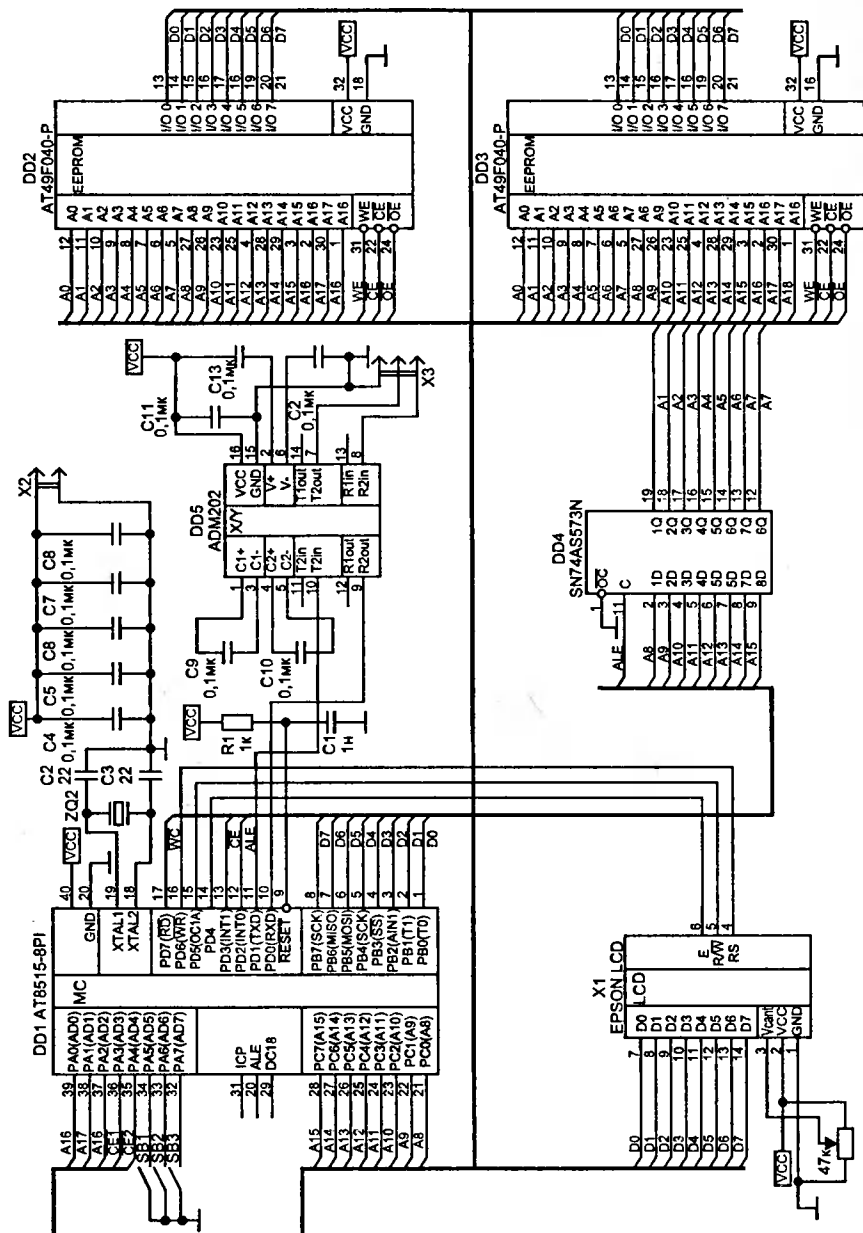


Рис. 6.7. Электрическая принципиальная схема

Команды, подаваемые на устройство, таковы:

10 — стереть первую МС памяти;

20 — стереть вторую МС памяти;

30 — записать в память данные;

40 — считать из памяти начиная с текущего адреса;

50 — выставить текущий адрес.

Последние две команды являются также отладочными и не используются в обычном нормальном функционировании устройства. Первые две не требуют никаких дальнейших действий, кроме ожидания сообщения на экране устройства, что память стерта. Команда записи в память потребует сначала ввести заголовок данной книги (т. е. автора и название, это нужно для дальнейшей идентификации текста при использовании устройства), завершив его посылкой символа 0x07, а затем передать массив текста. После окончания цикла записи символа в память устройство подтверждает прием, посылая этот же символ в компьютер, что позволяет контролировать скорость передачи данных и не потерять данные при записи. Передача текста также финишируется посылкой кода 0x07. Этот код был выбран потому, что является управляющим и обозначает звонок, писк на динамике компьютера (bell), т. е. принципиально не может встретиться в текстовых файлах.

На компакт диске, прилагаемом к книге, имеется файл resource.zip, в котором находятся следующие файлы:

reader.asm — содержит собственно код для AT90S8515;

rus.inc — содержит эквиваленты русских букв (AVR Studio традиционно некорректно работает с неанглийскими символами);

reader.hex — откомпилированный код, прошивка того, что получилось;

reader.cpp — пример программы для компьютера, написанной на языке Си под DOS, позволяющей записывать текстовые файлы;

reader.exe — это соответственно откомпилированный reader.cpp;

reader.ini — файл настроек для программы. Настроек, собственно, две: это на какой порт подключено устройство (в примере поддерживаются только COM1 и COM2, впрочем, легко добавить в исходник программы и остальные порты) и кодировка записываемого текста (WIN или DOS). Само устройство отображает текст, который записан в него только в кодировке WIN, но поскольку очень много

текстов в Интернете, доставшихся в наследство еще от FIDO, имеют кодировку DOS, то программа позволяет перекодировать при записи, на лету, освобождая пользователя от этой предварительной процедуры. Также программа отфильтровывает «лишние» пробелы, получившиеся при форматировании текста, убирает переносы, оставляя нетронутыми дефисы и тире перед фразами речи. Для программы при запуске существуют два возможных параметра — либо `reader.exe e`, означающий стирание памяти, либо `reader.exe w <имя_файла>`, позволяющее записать файл. При записи программа попросит ввести имя автора и название книги, причем для удобства выведет первые строчки текста (что, однако, удобно только для DOS-файлов. В случае win-кодировки на экран выведется бессмыслица).

И напоследок немного о кнопках. Это действительно тяжелая проблема, поскольку когда автор конструкции поставил «красивые» разноцветные кнопочки, они не выдержали интенсивного нажатия (все-таки двустрочный дисплей, нажимать где-то раз в секунду приходится), сдав за две недели, поэтому пришлось поставить производство отечественной «оборонки» — кнопки на основе концевиков — КМ1-1. И в итоге ни дребезга, ни залипания. Уже полгода успешно работают в довольно интенсивном режиме.

Автор: Александр Корниленко (E-mail: [seagull@mail.ru](mailto:seagull@mail.ru)).

## **6.8. Преобразование DTMF-сигнала в импульсный**

Устройство предназначено для детектирования введенного DTMF-методом номера телефона и последующего его набора пульсовым методом. Пример: с радиотелефона входим в телефонную линию, послав в качестве телефона цифру 8. База набирает «8» и подключает трубку. После подсоединения к междугородней телефонной станции абонент набирает код города и телефонного абонента методом DTMF, нажимает «\*». Нажатие «#» приводит к сбросу набранного номера. Устройство набирает телефон пульсовым методом и засыпает.

Канал шунтирования разговорного узла при наборе номера в прошивке не задействован, так как не хватает мощности, запасенной конденсатором.

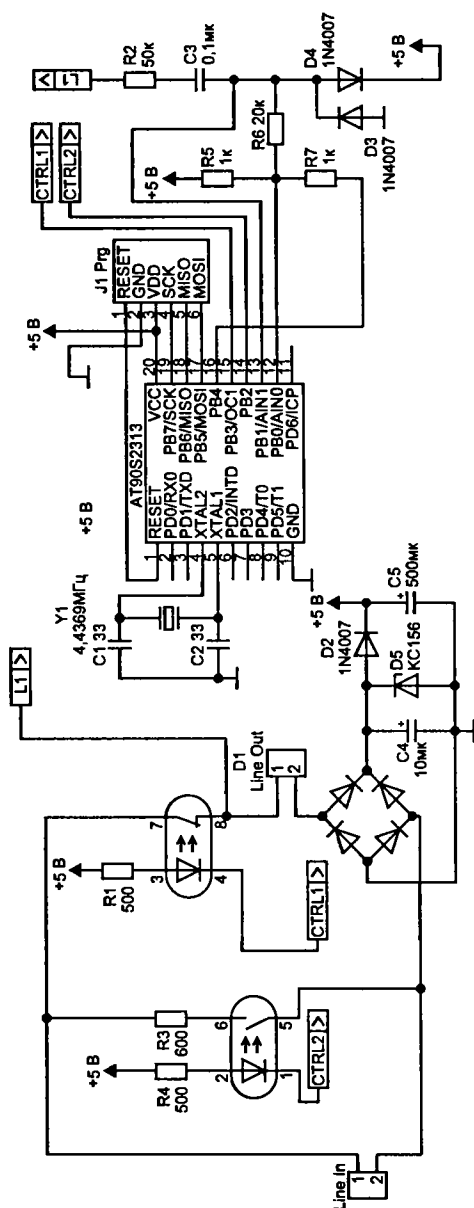


Рис. 6.8. Электрическая принципиальная схема

Устройство не допускает набора более 12 цифр.

Если набран код, содержащий ведущую «8», то после нее делается пауза 4 секунды, что позволяет выходить на межгород, просто набрав телефон и установив метод набора dtmf.

В случае, если активность не наблюдается в течение 10 секунд, устройство засыпает до следующего hung-up.

Конденсатор разряжается более 2 секунд, поэтому при малой паузе между звонками устройство может быть невосприимчиво.

Залипание телефонной линии, с точки зрения автора, невозможно, так как используется нормально замкнутое оптоуправляемое реле и при пропадании питания реле замкнуто.

Качество определения dtmf достаточно неплохое.

Автор схемы: Алексей Кургузов (E-mail: mandigit@chat.ru).

## 6.9. Многоточечный термометр

Проект родился в рамках изучения протокола сети MicroLan и возможности построения на ее основе автономных сетей, ориентированных на сбор данных, принятие решений и управление разнообразными объектами.



Рис. 6.9. Внешний вид конструкции

В качестве управляющего был использован микроконтроллер AT90S2313, индикация — ЖК-индикатор.

Применение преобразователя напряжения позволило создать автономное устройство с единственным источником питания 1,5 В. Большую часть времени устройство спит, просыпаясь на небольшие промежутки времени, чтобы найти новые устройства на шине, вывести информацию на индикатор, измерить температуру.

Устройство способно обслуживать до 16 датчиков типа DS18B20.

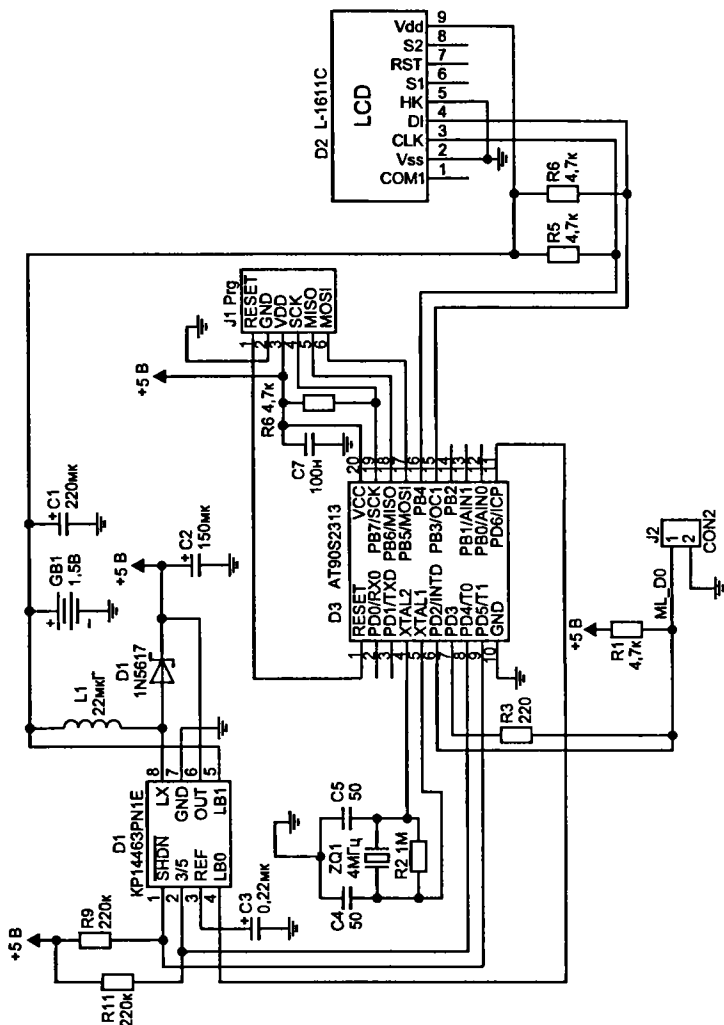


Рис. 6.10. Электрическая принципиальная схема цифрового термометра

Измерение температуры происходит примерно 1 раз в минуту, что накладывает ограничения на сферу применения прибора.

Устройство автоматически определяет количество подключенных датчиков, проводит измерения и отображает температуру, последовательно для каждого датчика, в порядке возрастания s/n.

Измерение потребляемой прибором мощности не производилось. Расчетно — 750 мкВт (ток около 500 мкА).

Устройство имеет защиту от КЗ.

Файлы, относящиеся к проекту, находятся на компакт диске, прилагаемом к книге.

Автор: Алексей Кургузов (E-mail: kurguzov@unact.ru).

## 6.10. Ключ для защиты от копирования

Описываемая здесь схема предназначена для защиты программ от нелегального копирования. Ее использование позволяет практически полностью исключить эту возможность, так как часть защищаемой программы можно хранить в микроконтроллере с установленным битом защиты, не позволяющим ее считать, а значит, и скопировать. Ключ подключается к параллельному порту и может работать одновременно с принтером. Сразу замечу, что эта схема имеет два ограничения. Во-первых, программа микроконтроллера ничего не делает, кроме получения данных от компьютера, их хранения и отправки обратно. Этого достаточно, чтобы программно определить наличие или отсутствие ключа, но, если в микроконтроллер записать часть программы или перед возвращением данных их шифровать, степень защиты значительно увеличится. Я думаю, границей между двумя этими вариантами является цена защищаемой программы в несколько сотен долларов. И во-вторых, эта схема не всегда позволяет работать принтеру, подключенному к тому же порту. Epson Stylus Color 600 прекрасно работал и даже автоматически определился, а Epson LX 1050+ нет. Для исправления этого надо немного переделать схему. Но и этот вариант можно использовать с некоторыми принтерами или, если принтер нужен и он не работает, подключить переключатель порта DataSwitch. Если принтер не подключен, схема устойчиво работает.

Схема конструкции показана на рис. 6.11.

### *Комментарии к схеме*

В целях уменьшения размера и стоимости устройства использован микроконтроллер AT90S1200 с внутренним RC-генератором. Это позволяет спокойно разместить всю собранную схему внутри

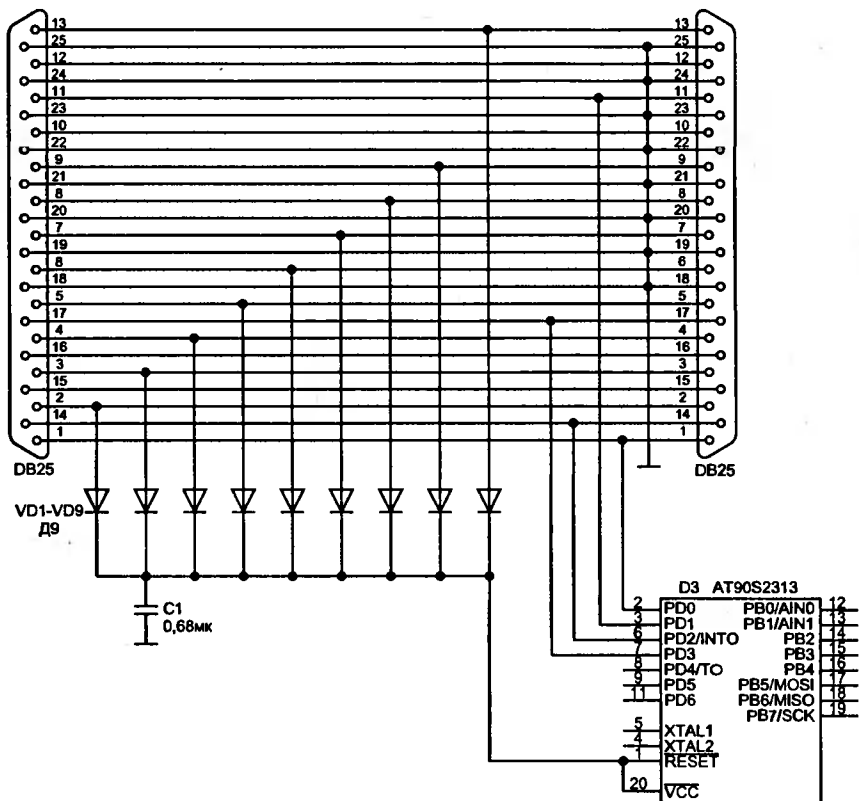


Рис. 6.11. Электрическая принципиальная схема

разъема или переходника, подключенного к порту, а стоимость не превышает примерно 100 рублей. Для обмена данными применяется синхронный последовательный интерфейс, назначение линий приведено в таблице:

Вывод микроконтроллера	Вывод порта	Назначение сигнала
PD3	ScltIn	Выбор принтера или ключа
Vcc, RESET	D0...D7	Питание ключа
PD0	Strobe	Данные от компьютера
PD1	Busy	Данные от ключа
PD2	AutoLF	Импульсы синхронизации от компьютера

Питание берется с того же порта, при работающем принтере микросхема почти всегда находится в режиме Power Down и потребляет меньше 1 мА, для ее питания достаточно единицы на одном из выводов шины данных. В активном режиме на выводы данных должны быть программно выставлены единицы, благодаря чему обеспечивается ток, достаточный для питания микроконтроллера. Желательно использовать германиевые диоды, так как падение напряжения на них меньше. Есть два исполнения микросхемы AT90S1200 с максимальными частотами 4 или 12 МГц и минимальными напряжениями питания соответственно 2,7 и 4 В. Лучше применять первое, так как из-за существующего разброса параметров параллельных портов на разных компьютерах может оказаться, что на питание микросхемы будет подано напряжение менее 4 В. Например, AT90S1200A-4PC.

### Программа микроконтроллера

```
.INCLUDE "t200def.inc" ; AT90S1200 @ 1 МГц
.CSEG
```

```
.DEF Byte=r16
.DEF LoopCounter=r18
.DEF Byte0=r19
.DEF Byte1=r20
.DEF Byte2=r21
.DEF Byte3=r22
.DEF Byte4=r23
.DEF Byte5=r24
.DEF Byte6=r25
.DEF Byte7=r26
```

```
.ORG 000
    rjmp RESET ; Reset Handler
.ORG 001
    rjmp EXT_INT0 ; IRQ0 Handler
```

```
RESET:
    ; Настраиваем направление работы портов
    ; Все линии портов после сброса настроены на работу в качестве входов
    ; а на неиспользуемых включены pull-up резисторы
    cli
    ldi r31,0
    out DDRB,r31
    out DDRD,r31
    ldi r31,$ff
    out PORTB,r31
    ldi r31,$72
    out PORTD,r31
```

; Ждем прихода импульса на вход INTO, находясь в режиме Power Down

```
ldi r31,$40
out GIMSK,r31
ldi r31,$30
out MCUCR,r31
```

```
UnLoop: sei
sleep
rjmp UnLoop
```

EXT\_INT0:

; Если PD3=0, данные относятся к принтеру

```
sbis PIND,3
reti
; Включение ключа
ldi r31,$02
out DDRD,r31
ldi Byte0,$31
ldi Byte1,$32
ldi Byte2,$33
ldi Byte3,$34
ldi Byte4,$35
ldi Byte5,$36
ldi Byte6,$37
ldi Byte7,$38
```

MainLoop:

```
rcall Byte8Exchange
; Здесь должен быть код, заменяющий часть защищаемой программы или
; шифрование данных
sbis PIND,3
rjmp RESET ; Работа с ключом завершена, переполнение стека допустимо
rjmp MainLoop
```

Byte8Exchange:

```
mov Byte,Byte0
rcall ByteExchange
mov Byte0,Byte
mov Byte,Byte1
rcall ByteExchange
mov Byte1,Byte
mov Byte,Byte2
rcall ByteExchange
mov Byte2,Byte
mov Byte,Byte3
rcall ByteExchange
mov Byte3,Byte
mov Byte,Byte4
rcall ByteExchange
mov Byte4,Byte
mov Byte,Byte5
rcall ByteExchange
mov Byte5,Byte
```

```

mov   Byte,Byte6
rcall ByteExchange
mov   Byte6,Byte
mov   Byte,Byte7
rcall ByteExchange
mov   Byte7,Byte
ret

```

```

ByteExchange:                ; Обмен одним байтом данных
    ldi   LoopCounter,8
    ; Обработка положительного фронта сигнала синхронизации
Loop8:    sbrs   Byte,7        ; Вывод бита
    cbi   PORTD,1
    sbrc  Byte,7
    sbi   PORTD,1
Wait1:    sbis   PIND,2        ; Ждем прихода положительного фронта
    rjmp  Wait1
    sec   ;c=1                ; Прием бита
    sbis  PIND,0
    clc   ;c=0
    rol   Byte
Wait0:    sbic   PIND,2        ; Ждем прихода отрицательного фронта
    rjmp  Wait0
    ; Цикл для 8 битов байта
    dec   LoopCounter
    brne  Loop8
    ret
.EXIT

```

## Программа PC

Исходник на C примера, проверяющего наличие ключа.

```

#include
#include
#include

char SendByte(char ByteOut)
{
    int i,j;
    long li;
    unsigned char ByteOutCpy, ByteIn = 0;
    ByteOutCpy = ByteOut;
    for (i=0; i<8; i++)
    {
        ByteIn = (ByteIn << 1) + ((inportb(0x379)&0x80)==0);
        outportb(0x37A,0x02|(((128&ByteOutCpy)==0)));
        for (li=0; li<20001; li++);
        outportb(0x37A,0x00|(((128&ByteOutCpy)==0)));
        for (li=0; li<20001; li++);
        outportb(0x37A,0x02|(((128&ByteOutCpy)==0)));
        ByteOutCpy = ByteOutCpy << 1;
    }
}

```

```
    for (li=0; li<20001; li++);
}
return ByteIn;
}

void KeyOn(void)
{
    int i;
    long li;
    for (i=0; i<64; i++)
    {
        outportb(0x37A, 0x00);
        for (li=0; li<20001; li++);
        outportb(0x37A, 0x03);
        for (li=0; li<20001; li++);
    }
    delay(100);
}

void KeyOff(void)
{
    int i;
    long li;
    for (i=0; i<128; i++)
    {
        outportb(0x37A, 0x0B);
        for (li=0; li<20001; li++);
        outportb(0x37A, 0x08);
        for (li=0; li<20001; li++);
    }
}

void main()
{
    KeyOn(); // Включение ключа
    printf("%02X ", SendByte(0)); // Отправка 8 байтов.
    printf("%02X ", SendByte(1));
    printf("%02X ", SendByte(2));
    printf("%02X ", SendByte(3));
    printf("%02X ", SendByte(4));
    printf("%02X ", SendByte(5));
    printf("%02X ", SendByte(6));
    printf("%02X ", SendByte(7));
    printf("\n");
    printf("%02X ", SendByte(7)); // Отправка следующих 8 байтов и одновременное
    printf("%02X ", SendByte(6)); // получение байтов, отправленных раньше
    printf("%02X ", SendByte(5));
    printf("%02X ", SendByte(4));
    printf("%02X ", SendByte(3));
    printf("%02X ", SendByte(2));
    printf("%02X ", SendByte(1));
    printf("%02X ", SendByte(0));
```

```
printf("\n");
printf("%02X ", SendByte(0xF0)); // Отправка следующих 8 байтов и одновременное
printf("%02X ", SendByte(0xF1)); // получение байтов, отправленных раньше
printf("%02X ", SendByte(0xF2));
printf("%02X ", SendByte(0xF3));
printf("%02X ", SendByte(0xF4));
printf("%02X ", SendByte(0xF5));
printf("%02X ", SendByte(0xF6));
printf("%02X ", SendByte(0xF7));
printf("\n");
printf("\n");
KeyOff(); // Отключение питания ключа
```

Автор: Сафонников В. В. (E-mail: svv@ufanet.ru).

## 6.11. Кодовый замок

### 1. Назначение и описание устройства

Схема предназначена для работы в качестве электронного кодового замка, музыкального звонка и устройства регистрации.

Секретный код состоит из 5 десятичных цифр и набирается на 11-кнопочной клавиатуре; дополнительная кнопка используется как сброс в случае, если была нажата неверная клавиша. Предусмотрена возможность ввода нового кода с этой же клавиатуры.

При нажатии на кнопку звонка микроконтроллер проигрывает мелодию или выдает звуковой сигнал, напоминающий звонок телефонного аппарата. Первый вариант выбирается в случае, если первая цифра секретного кода четная, а второй — если нечетная.

При проигрывании мелодии останавливается вывод данных во внешний компьютер, поэтому, если шел прием данных и был нажат звонок, потребуется дополнительное время на исправление возникшей ошибки. При проигрывании простого сигнала такого не происходит. Кроме того, для правильного воспроизведения мелодии желательно использовать внешний кварц вместо внутреннего RC-генератора микросхемы.

Микроконтроллер имеет 64 байта энергонезависимой памяти (EEPROM), которая используется для записи происходящих событий. Есть 3 типа событий: включение питания, правильный набор ко-

да и неправильный набор кода. Одновременно с событием записывается время, прошедшее после предыдущей записи. Точность записи времени около 8 минут. Если время между записями превысило 50 суток, дальнейший счет времени останавливается. Естественно, что для точного счета времени надо использовать кварцевый, а не встроенный RC-генератор.

Содержимое энергонезависимой памяти постоянно побитно, с частотой 3815 Гц выводится на один из выводов микроконтроллера и может быть считано для контроля внешним компьютером. Объем памяти позволяет хранить 30 записей по 2 байта. Число циклов записи в EEPROM ограничено, поэтому для увеличения срока службы используется специальный формат записи, разобраться в нем можно по исходному коду программы. После включения питания секретный код равен числу 23232. Для его изменения надо: 1) набрать старый код, при этом сработает соленоид, открывающий замок; 2) в момент отключения соленоида должна быть нажата кнопка 2 — это переведет устройство в режим ввода нового пароля; 3) ввести новый пароль, при ошибке можно пользоваться клавишей сброса.

Все нажатия на клавиши и некоторые другие действия озвучиваются короткими звуковыми сигналами.

## 2. Схема

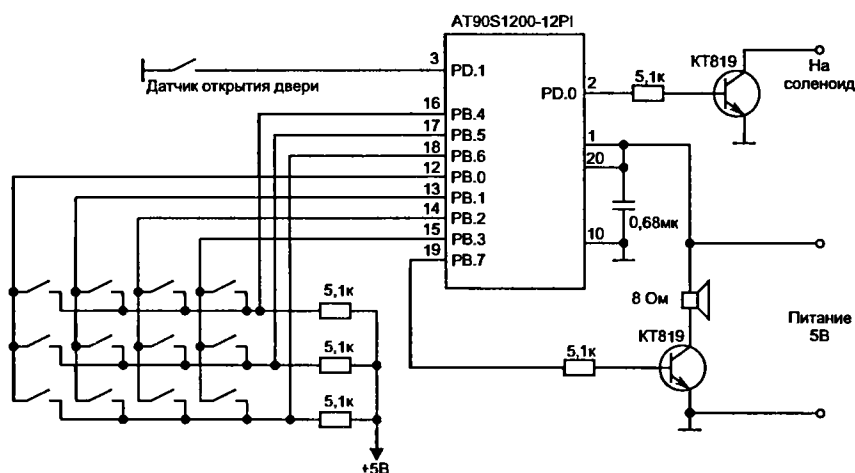


Рис. 6.12. Электрическая принципиальная схема кодового замка

### 3. Комментарии к схеме

Питание на микроконтроллер подается постоянно, поэтому желательно использовать сетевой блок питания, а не батарейки.

В качестве исполнительного устройства проще всего использовать подходящий электромагнит. Но можно и электродвигатель, тогда надо добавить небольшую схему на основе выключателей, которая должна обеспечивать открытие замка при единице на соответствующем выводе микроконтроллера и закрытие при нуле.

### 4. План

Управляющую программу для чтения информации из EEPROM автор не написал, но возможность чтения заложена в программе микроконтроллера, так что, если найдутся желающие, автор будет только рад.

### 5. Программа

Программа относительно сложная, и автор не рекомендовал бы использовать ее для изучения ассемблера AVR, но тем не менее разобраться можно. Текст программы:

```
.INCLUDE    "1200def.inc"        ; AT90S1200 @ 1 МГц
.CSEG

.EQU  evReset=1
.EQU  evPassed=3
.EQU  evDenied=5
.EQU  KeyRing=1          ; Код клавиши звонка
.EQU  KeyEnter=9         ; Код клавиши сброса
.EQU  NewPassKey=2       ; Код клавиши, нажимаемой для ввода нового пароля
.EQU  KeyLen=5           ; Длина секретного кода не должна быть больше 5 цифр
.EQU  FreqKeyPressed=50  ; Константа частоты для выдачи писка при нажатии
                        ; кнопок

.EQU  Note_1=227         ; Константы частот и длительности нот
.EQU  Note_1a0 = Note_1
.EQU  Note_2=202
.EQU  Note_ci0 = Note_2
.EQU  Note_3=191
.EQU  Note_do1 = Note_3
.EQU  Note_4=170
.EQU  Note_re1 = Note_4
.EQU  Note_5=152
.EQU  Note_mi1 = Note_5
.EQU  Note_6=143
.EQU  Note_fa1 = Note_6
.EQU  Note_7=128
```

```
.EQU Note_sol1 = Note_7
.EQU Note_8=114
.EQU Note_la1 = Note_8
.EQU Note_9=101
.EQU Note_ci1 = Note_9
.EQU Note_10=96
.EQU Note_do2 = Note_10
.EQU Note_11=85
.EQU Note_re2 = Note_11
.EQU Note_12=76
.EQU Note_mi2 = Note_12
.EQU Note_13=72
.EQU Note_fa2 = Note_13
.EQU Note_14=64
.EQU Note_sol2 = Note_14
.EQU DURATION = 350

.EQU Time_1=1000/Note_1
.EQU Time_2=1000/Note_2
.EQU Time_3=1000/Note_3
.EQU Time_4=1000/Note_4
.EQU Time_5=1000/Note_5
.EQU Time_6=1000/Note_6
.EQU Time_7=1000/Note_7

.DEF KeyPass=r0
.DEF tmp=r0
.DEF KeyPass1=r1
.DEF KeyPass2=r2
.DEF KeyPass3=r3
.DEF KeyPass4=r4
.DEF KeyPass5=r5
.DEF SSREG=r6
.DEF DelayVar=r7
.DEF IntUse0=r8
.DEF LastKey=r9
.DEF ZerReg=r10
.DEF Time0=r16 ; Время, увеличивается 3815 раз в секунду
.DEF Time1=r17 ; Время, увеличивается раз в 67 секунд
.DEF Time2=r18 ; Время, увеличивается раз в 47721 час,
                ; насыщается за 50 суток
.DEF EEadr=r20 ; Адрес для записи в EEPROM, равен 255,
                ; если в текущий момент запись не производится
.DEF EventType=r21
.DEF BeepVar=r21

.DEF TimerEEByte=r22
.DEF TimerEEBit=r23
.DEF Key=r24
.DEF BeepDuration=r25
.DEF FreqConst=r26
.DEF IntUse1=r27
```

```

; .DEF PassLoopCounter=r28
; .DEF Flags=r29
; bit 0 - Password accepted

.ORG 000
    clr    ZerReg
    rjmp   RESET
.ORG 002
; =====
; Здесь оказываемся при возникновении прерывания от таймера 3 раза в секунду
; =====
; Таймер используется как RTC и для вывода информации EEPROM
; Увеличение счетчика времени
    in      SSREG, SREG
    subi    Time0, 1
    adc     Time1, ZerReg
    adc     Time2, ZerReg
    brcc    Time0k
    com     Time1 ; Если прошло переполнение, счетчик времени остановлен
    com     Time2

Time0k:
; Чтение данных из EEPROM и выдача на внешний порт.
; Процедура обработки прерывания будет вызвана не раньше, чем через 300-3*4 мс
; после последнего обращения к EEPROM, поэтому проверка готовности EEPROM не
; производится, она должна быть уже готова.
    andi    TimerEEByte, $3F
    out     EEAR, TimerEEByte
    sbi     EECR, EERE           ; set EEPROM Read strobe
    sbi     EECR, EERE           ; set EEPROM Read strobe 2nd time
    in      IntUse0, EEDR        ; get data
    andi    TimerEEBit, 7
    brne    BitLop
    inc     TimerEEByte
BitLop:    inc     TimerEEBit ; = 1..8
    mov     IntUse1, TimerEEBit
    rol     IntUse0
BitShift:    ; Выбираем посылаемый бит
    ror     IntUse0             ; Используется как текущий считанный EEPROM байт
    dec     IntUse1             ; Используется как счетчик
    brne    BitShift
    in      IntUse1, PORTD
    bst     IntUse0, 0
    bld     IntUse1, 0
    out     PORTD, IntUse1
    out     SREG, SSREG
    reti
; =====
; Здесь оказываемся при включении питания схемы
; =====

```

RESET:

; Настройка микросхемы и программы.

```
ldi r31,$8F ; Настраиваем направление работы портов
out DDRB,r31
ldi r31,$03
out DDRD,r31
ldi r31,$05 ; Настраиваем таймер
out TCCR0,r31
ldi r31,$02
out TIMSK,r31 ; Разрешаем прерывания при переполнении таймера
ldi r31,2
mov KeyPass1,r31 ; Загружаем пароль по умолчанию
ldi r31,3
mov KeyPass2,r31
ldi r31,2
mov KeyPass3,r31
ldi r31,3
mov KeyPass4,r31
ldi r31,2
mov KeyPass5,r31
ldi EventType,evReset
clr Time1 ; Сбрасываем счетчик времени
clr Time2
rcall WrireEEPROM
sei ; Разрешаем прерывания
```

MainLoop:

```
; Подготавливаемся к вводу пароля.
clr r30,0
clr r31,0
ori Flags,1 ; При предъявлении пароля пока ошибок не было
clr KeyPass
clr LastKey
```

PassLoop: ; Цикл ввода пароля

```
rcall GetKey ; Записывает в переменную Key код нажатой клавиши
brtc PassLoop ; и устанавливает бит T, если клавиша нажата
cp Key,LastKey
breq PassLoop ; Если предыдущая клавиша еще не отпущена
cpi Key,KeyEnter
breq MainLoop ; Если нажат сброс, начинаем ввод строки с начала
ldi FreqConst,FreqKeyPressed
ldi BeepDuration,5
rcall Beep ; Выдача короткого звукового сигнала
mov LastKey,Key

inc r30
ld KeyPass,Z
cpse Key,KeyPass ; Сравниваем введенную цифру и цифру пароля из ОЗУ,
andi Flags,~1 ; если ошибка, сбрасываем бит успешности пароля.

cpi r30,KeyLen ; Длина секретного кода.
brne PassLoop
```

```

; Вывод бита на внешний вывод порта и запись события в EEPROM
ldi EventType,evDenied
bst Flags,0
brtc PasswordDenied
sbi PORTD,1 ; Если пароль совпал, включаем соленоид
ldi EventType,evPassed
PasswordDenied:
rcall WrireEEPROM

; Открытие двери при совпадении кода или цикл ожидания при ошибке
ldi Key,5
OpenLoop:
rcall Delay100
ldi BeepDuration,Time_7/2
ldi FreqConst,Note_7
rcall Beep
dec Key
sbic PIND,2 ; Если сработал концевик (открылась дверь), то выходим
brne OpenLoop ; или выходим из цикла по тайм-ауту
cbi PORTD,1 ; Отключаем соленоид

; Нажатие на клавишу NewPassKey в этом месте приводит к загрузке
rcall GetKey ; нового секретного кода
brtc MainLoop
cpi Key,NewPassKey
brne MainLoop
sbrs Flags,0 ; Если старый был предъявлен верно
rjmp MainLoop

; Ввод нового секретного кода.
ldi FreqConst,Note_3 ; Проигрывание "приглашения" для
ldi BeepDuration,20 ; изменения пароля
rcall Beep
ldi FreqConst,Note_5
ldi BeepDuration,20
rcall Beep
ldi FreqConst,Note_7
ldi BeepDuration,20
rcall Beep
rcall Delay100
NewPassword:
clr LastKey
clr r30
clr r31
NewPassLoop:
rcall GetKey
brtc NewPassLoop ; Ожидаем нажатия клавиши
cp Key,LastKey
breq NewPassLoop ; Если клавиша еще не отпущена, продолжаем цикл
cpi Key,KeyEnter
breq NewPassword ; Если сброс, начинаем ввод строки с начала
mov LastKey,Key

```

```

ldi BeepDuration, 2*10
ldi FreqConst, 2*FreqKeyPressed
rcall Beep ; Удлиненный звуковой сигнал

inc r30
st Z, Key ; Сохраняем введенную цифру

cpi r30, KeyLen ; Длина секретного кода
brne NewPassLoop

rcall Delay100
ldi FreqConst, Note_7 ; Проигрывание мелодии, говорящей об
ldi BeepDuration, 20 ; успешном изменении пароля
rcall Beep
ldi FreqConst, Note_5
ldi BeepDuration, 20
rcall Beep
ldi FreqConst, Note_3
ldi BeepDuration, 20
rcall Beep
rjmp MainLoop

```

```

; =====
; Далее идут процедуры, вызываемые из основной программы
; =====

```

Beep: ; Выдача звукового сигнала

```

ldi BeepVar, 20
Beep1: mov DelayVar, FreqConst
Delay2: rjmp NopJump1 ; 2 +
NopJump1: dec DelayVar ; 1 + = 5 мкс
brne Delay2 ; 2 +
sbi PORTB, 7
mov DelayVar, FreqConst
Delay3: rjmp NopJump2
NopJump2: dec DelayVar
brne Delay3
cbi PORTB, 7
dec BeepVar
brne Beep1
dec BeepDuration
brne Beep
sei
ret

```

```

; =====
; Задержка на время порядка секунды

```

```

Delay100:
ldi BeepDuration, 3
D100_3: clr FreqConst
D100_2: clr DelayVar
D100_1: dec DelayVar
brne D100_1

```

```

dec   FreqConst
brne  D100_2
dec   BeepDuration
brne  D100_3
ret

```

```

;=====

```

```

; Проверка приема данных от клавиатуры

```

```

GetKey:
    clt                ; Сброс признака, что была нажата клавиша
    ldi   r31,$0E       ; Младшая тетрада - запрос в порт, старшая -
    rcall GetKeyAnswer  ; смещение ответа
    brne  KeyPressed
    ldi   r31,$3D
    rcall GetKeyAnswer
    brne  KeyPressed
    ldi   r31,$6B
    rcall GetKeyAnswer
    brne  KeyPressed
    ldi   r31,$97
    rcall GetKeyAnswer
    brne  KeyPressed
    ret

```

```

KeyPressed:

```

```

    set
    swap r31
    andi r31,$0F
    swap Key
    andi Key,$07

```

```

CCFind:    inc   r31        ; Поиск, какой конкретно бит установлен

```

```

    lsr   Key
    brcc  CCFind
    breq  NoMul
    clt

```

```

; Если нажато несколько клавиш, то не нажата ни одна

```

```

NoMul:    mov   Key,r31

```

```

; Проверка, нажата ли кнопка Ring, если нажата, то будет выдан звуковой сигнал

```

```

; без сообщения об этом вызывающей функции

```

```

    cpi   Key,KeyRing
    breq  Ring
    rjmp  NoRing

```

```

Ring:     sbrc  KeyPass1,0    ; Выбор типа мелодии в зависимости от четности

```

```

    rjmp  SimpleRing        ; старшей цифры секретного кода
    rcall Delay100

```

```

; Проигрываем сложную мелодию (Отговорила роща золотая)

```

```

; 1 мс 1/8

```

```

    ldi   FreqConst,Note_m11
    ldi   BeepDuration,2*DURATION/Note_m11
    rcall Beep_Del

```

```

; 1 мс 1/8

```

```

    ldi   FreqConst,Note_m11

```

```

ldi BeepDuration,2*DURATION/Note_mi1
rcall Beep_Del
;1 ми 1/8
ldi FreqConst,Note_mi1
ldi BeepDuration,2*DURATION/Note_mi1
rcall Beep_Del
;1 фа 3/8
ldi FreqConst,Note_fa1
ldi BeepDuration,6*DURATION/Note_fa1
rcall Beep_Del
;1 ми 1/8
ldi FreqConst,Note_mi1
ldi BeepDuration,2*DURATION/Note_mi1
rcall Beep_Del
;1 фа 3/8
ldi FreqConst,Note_fa1
ldi BeepDuration,6*DURATION/Note_fa1
rcall Beep_Del
;1 ми 1/16 (1/8)
ldi FreqConst,Note_mi1
ldi BeepDuration,2*DURATION/Note_mi1
rcall Beep_Del
;1 ре 1/8
ldi FreqConst,Note_re1
ldi BeepDuration,2*DURATION/Note_re1
rcall Beep_Del
;1 до 1/8
ldi FreqConst,Note_do1
ldi BeepDuration,2*DURATION/Note_do1
rcall Beep_Del
;1 си 1/4
ldi FreqConst,Note_ci0
ldi BeepDuration,4*DURATION/Note_ci0
rcall Beep_Del
;1 соль 1/4
ldi FreqConst,Note_sol1
ldi BeepDuration,4*DURATION/Note_sol1
rcall Beep_Del
;1 фа 3/8
ldi FreqConst,Note_fa1
ldi BeepDuration,6*DURATION/Note_fa1
rcall Beep_Del
;1 фа 1/8
ldi FreqConst,Note_fa1
ldi BeepDuration,2*DURATION/Note_fa1
rcall Beep_Del
;1 ми 3/8
ldi FreqConst,Note_mi1
ldi BeepDuration,6*DURATION/Note_mi1
rcall Beep_Del

```

```

; 1 ми 1/6
    ldi FreqConst, Note_mi1
    ldi BeepDuration, 3*DURATION/Note_mi1
    rcall Beep_Del

; 1 ми 1/8
    ldi FreqConst, Note_mi1
    ldi BeepDuration, 2*DURATION/Note_mi1
    rcall Beep_Del

; 1 фа 1/8
    ldi FreqConst, Note_fa1
    ldi BeepDuration, 2*DURATION/Note_fa1
    rcall Beep_Del

; 1 ми 1/8
    ldi FreqConst, Note_mi1
    ldi BeepDuration, 2*DURATION/Note_mi1
    rcall Beep_Del

; 0 си 1/8
    ldi FreqConst, Note_ci0
    ldi BeepDuration, 2*DURATION/Note_ci0
    rcall Beep_Del

; 1 ре 1/8
    ldi FreqConst, Note_re1
    ldi BeepDuration, 2*DURATION/Note_re1
    rcall Beep_Del

; 1 фа 1/8
    ldi FreqConst, Note_fa1
    ldi BeepDuration, 2*DURATION/Note_fa1
    rcall Beep_Del

; 1 ми 5/8
    ldi FreqConst, Note_mi1
    ldi BeepDuration, 10*DURATION/Note_mi1
    rcall Beep_Del
    rjmp EndRing

SimpleRing:    ; Проигрываем простую мелодию
    ldi FreqConst, Note_7
    ldi BeepDuration, 1
    rcall Beep
    ldi FreqConst, Note_5
    ldi BeepDuration, 1
    rcall Beep
    ldi FreqConst, Note_3
    ldi BeepDuration, 1
    rcall Beep
    ldi FreqConst, Note_5
    ldi BeepDuration, 1
    rcall Beep

EndRing:
    clt    ; Сбрасываем флаг нажатия клавиши

NoRing:
    ret

```

```

;=====
Beep_Del:    ; Короткая пауза перед нотой и сама нота
            ldi BeepVar,50
Del2:  clr   DelayVar
Del1:  dec   DelayVar
            brne Del1
            dec BeepVar
            brne Del2
            cli
            rjmp Beep

;=====
; Сканирование клавиатуры: установка на выходных линиях заданного кода и прием
; ответа вызывается из функции GetKey
GetKeyAnswer:
            mov Key,r31
            andi Key,$0F
            out PORTB,Key
            clr DelayVar
Delay1:  dec DelayVar
            brne Delay1
            in Key,PINB
            com Key
            andi Key,$70          ; Если что-то было найдено, флаг Z не нулевой
            ret

;=====
; Запись события EventType в текущую ячейку EEPROM памяти и обнуление времени
WriteEEPROM:
            ; Поиск последней записи
            clr tmp
            ldi EEadr,2
FindNextAdr:
            out TCNT0,ZerReg      ; Пока не сработал таймер, читающий из EEPROM
                                   ; (через 300 мс), все операции должны быть завершены
            subi EEadr,-2
            cpi EEadr,62
            brne EERead
            ldi EEadr,4          ; Если дошли до 64-го адреса, начинаем с начала
EERead:
            sbic EECR,EWE        ; Ждем обнуления бита EWE
            rjmp EERead
            out EEAR,EEadr       ; Подготавливаем адрес
            sbi EECR,EERE        ; Устанавливаем бит для синхронизации
            sbi EECR,EERE        ; Дважды
            in r31,EEDR          ; Читаем данные
            dec tmp              ; Если было 256 неудачных попыток найти бит -
            breq NoZav           ; признак последней записи, значит, его нет
                                   ; вообще, значит, пишем по любому адресу
            sbrs r31,0
            rjmp FindNextAdr

```

```

NoZav:      andi r31,$FE
EEWrite0:   ; Запись для сброса флага, указывающего на последнюю запись
            sbic EECR,EEWE
            rjmp EEWwrite0
            out  EEAR,EEAdr
            out  EEDR,r31
            sbi  EECR,EEWE
            inc  EEAdr
EEWrite1:   ; Запись старшего байта времени
            sbic EECR,EEWE
            rjmp EEWwrite1
            out  EEAR,EEAdr
            out  EEDR,Time2
            sbi  EECR,EEWE
            inc  EEAdr
EEWrite2:   ; Запись времени, типа события и флага последней записи
            sbic EECR,EEWE
            rjmp EEWwrite2
            out  EEAR,EEAdr
            mov  r31,Time1
            andi r31,$F8
            or   r31,EventType
            out  EEDR,r31
            sbi  EECR,EEWE
            clr  Time1      ; Счетчик считает время между записями в EEPROM,
            clr  Time2      ; поэтому сбрасываем его
            ret
.EXIT

```

; Таблица использования линий микроконтроллера  
; Порт B используется для подключения клавиатуры и динамика  
; PORTB.7 - вывод на динамик  
; PORTB.0...3 - выходы на сканирование клавиатуры  
; PORTB.4...6 - входы от 12-клавишной клавиатуры, подключены к +5 В через резисторы  
; Порт D используется для управления замком, выдачи информации и проверки двери  
; PORTD.0 используется для вывода информации из EEPROM, на нем с частотой  
; 3815 Гц (@1МГц) побитно выводится все ее содержимое  
; PORTD.1 Подключен на электромагнит или двигатель, единица разрешает открытие  
; PORTD.2 Подключен на концевик, срабатывающему (=0) при открытии

Автор: Сафонников В. В. (E-mail: svv@ufanet.ru).

## 6.12. Музыкальный звонок

### 1. Краткое описание

Эта схема музыкального звонка собрана на микроконтроллере, может быть, это похоже на «забывание гвоздей логарифмической линейкой», тем не менее у нее есть некоторые преимущества. Во-пер-

вых, используется всего одна микросхема, благодаря чему, кроме традиционных преимуществ вроде увеличения надежности, уменьшения потребления энергии, появляется принципиальная возможность разместить всю схему в очень небольшом объеме, например оформив в виде музыкальной поздравительной открытки.

## 2. Схема

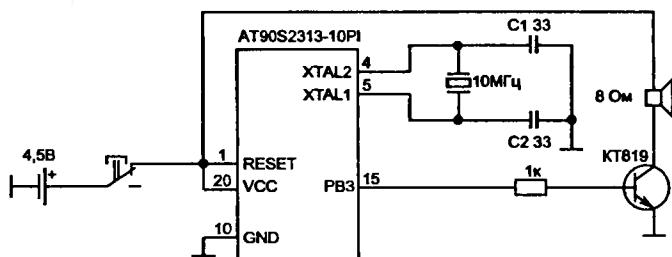


Рис. 6.13. Электрическая принципиальная схема музыкального звонка

Благодаря использованию микроконтроллера схема предельно упрощается, необходимо только подключить питание (от 2,7 до 6 В), кварцевый резонатор и, если необходимо, усилитель низкой частоты, собранный на транзисторе KT815. Если предполагается, что ток через нагрузку не будет превышать 20 мА (например, при использовании пьезоизлучателя), можно обойтись без усилителя.

Следует заметить, что 20 мА — это максимальный ток логического нуля, т. е. второй вывод звукового излучателя следует подключать к плюсу питания.

Кнопка звонка должна иметь нормально замкнутые контакты, при нажатии на нее схема обесточивается, а при отпускании на нее подается питание и происходит автоматический сброс микроконтроллера.

Сразу же после сброса начинает работать программа, проигрывающая заданную мелодию. После завершения мелодии микроконтроллер переходит в режим пониженного энергопотребления Power Down и находится в нем до следующего нажатия на кнопку.

Частоту кварца можно выбрать практически любую в диапазоне от 32 768 кГц до 10 МГц. Схема была проверена на частоте 10 МГц. Если вы хотите использовать кварц на меньшую частоту (при этом

уменьшится и потребляемый ток схемы), необходимо пропорционально уменьшить коэффициенты таблицы SoundTab и длительности звучания всех нот.

### 3. Программа

Эта программа написана на ассемблере для AT90S2313, таблица нот рассчитана для работы процессора на максимальной частоте 10 МГц, вместо мелодии используется проигрывание гаммы от более низких нот к более высоким и обратно. Текст программы:

```
.INCLUDE    "2313def.inc"

.DEF    Step=r20
.DEF    FreqIndex=r21
.DEF    FreqDelay=r22
.DEF    SSREG=r23

.CSEG

.ORG    0
        rjmp    Reset

.ORG    OVFOaddr
        in      SSREG, SREG
        dec     FreqDelay
        out     SREG, SSREG
        reti

Reset:
        ldi     r30, $DF
out     SPL, r30
        ; Настройка направления работы линий порта B
        ldi     r30, $08
        out     DDRB, r30
        ; Режим работы таймера 0 с максимальным предварительным делением
        ; Здесь же разрешаем прерывания
        ldi     r30, $05
        out     TCCR0, r30
        ldi     r30, $02
        out     TIMSK, r30
        sei
        ; Режим работы таймера 1 на переключение внешнего вывода,
        ; выбор коэффициента предварительного деления тактовой частоты 1:1
        ; и автоматический сброс таймера при совпадении
        ldi     r30, $40
        out     TCCR1A, r30
        ldi     r30, $09
        out     TCCR1B, r30
        ; Установка номера шага на начало
        clr     Step
```

```

ReadNote:      ; Чтение длительности и номера одной ноты
    ldi r31,High(2*ProgramTab) ; FreqIndex = Lo ProgramTab[Step]
    ldi r30,Low(2*ProgramTab)  ; FreqDelay = Hi ProgramTab[Step]
    mov r0,Step                ; Step++
    lsl r0
    add r30,r0
    inc Step
    lpm
    mov FreqIndex,r0
    inc r30
    lpm
    sbrc r0,7
    rjmp SleepReset           ; Если старший бит установлен, то
    mov FreqDelay,r0          ; это означает, что мелодия закончилась

SetFreq:       ; Настройка таймера 1 на вывод частоты текущей ноты
    ldi r31,High(2*SoundTab)  ; OCR1A = SoundTab[FreqIndex]
    ldi r30,Low(2*SoundTab)
    lsl FreqIndex
    add r30,FreqIndex
    lpm
    out OCR1AL,r0
    inc r30
    lpm
    out OCR1AH,r0

Wait: tst FreqDelay           ; Ожидаем заданное время, пока проигрывается
    brne Wait                 ; текущая нота
    rjmp ReadNote

SleepReset:
    ldi r30,$3F                ; Подготовка к переходу в режим Power Down
    out MCUCR,r30
    sleep                      ; Отключение микроконтроллера
    rjmp Reset                 ; Эта команда в данной версии программы не
                                ; должна исполняться никогда

.ORG $100
SoundTab:
; Таблица констант соответствующим нотам
; Желательно выравнивать таблицу по границе 256 байтов, чтобы упростить программу,
; отказавшись от операций с 16-битовыми словами
.DW 11364,10292,9322,8443,7647,6926,6273 ; 440 Гц - нота "ля"
.DW 5682,5146,4661,4222,3824,3463,3137 ; вторая октава
.DW 2841,2573,2330,2111,1912,1732,1568 ; третья октава
; При необходимости таблицу можно продолжить

ProgramTab:
; Таблица последовательности нот,
; формат таблицы: байт длительности/кода операции, байт номера частоты
; В этом примере записано проигрывание гаммы
.DW $1001, $1002, $1003, $1004, $1005, $1006, $1007

```

```
.DW $1008, $1009, $100A, $100B, $100C, $100D, $100E
.DW $200D, $200B, $2009, $2007, $2005, $2003, $2001
.DW $8000

.EXIT
```

Автор: Сафонников В. В. (E-mail: svv@ufanet.ru).

**6.13. Универсальный расширитель последовательного порта**

**Назначение**

Схема предназначена для управления с помощью компьютера различными устройствами: бытовыми электроприборами, шаговыми двигателями, электромагнитами и т. д. Можно ее также использовать как основу для программирования микросхем памяти, микроконтроллеров или других разработок. Для подключения к внешним устройствам имеются 24 универсальных выхода, совместимых с TTL, состояние и режим работы каждого из них можно задать с помощью управляющей программы. Она может быть написана на любом языке, поддерживающем технологию COM (ActiveX), например Delphi, MS Visual C++, MS Visual Basic, JavaScript, VBS и т. д.

**Характеристики устройства**

Способ подключения к компьютеру	COM-порт
Настройки порта	9600 8N1
Максимально допустимый ток через любой вывод, мА	40
Максимально допустимый суммарный ток через все выводы, мА	140
Минимальное время обновления состояния выходов, мс	7,3

Основу схемы составляет микроконтроллер AT90S8515, он подключен к компьютеру через последовательный порт, а для управления внешними устройствами используются выводы трех свободных портов — А, В и С. Для упрощения схемы данные передаются только в одну сторону — от компьютера к микроконтроллеру, никаких под-

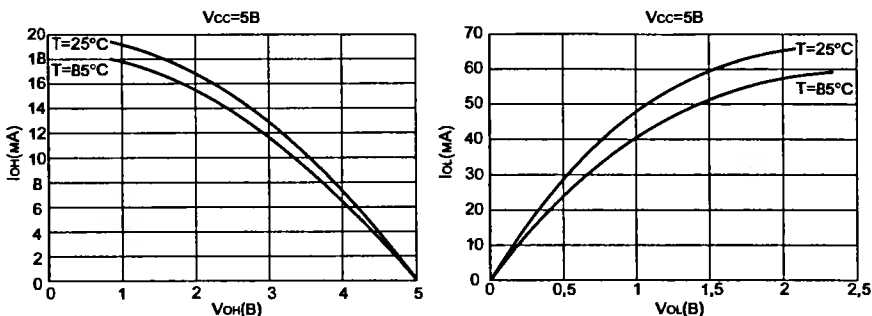


Рис. 6.14. Нагрузочные характеристики выводов микросхемы

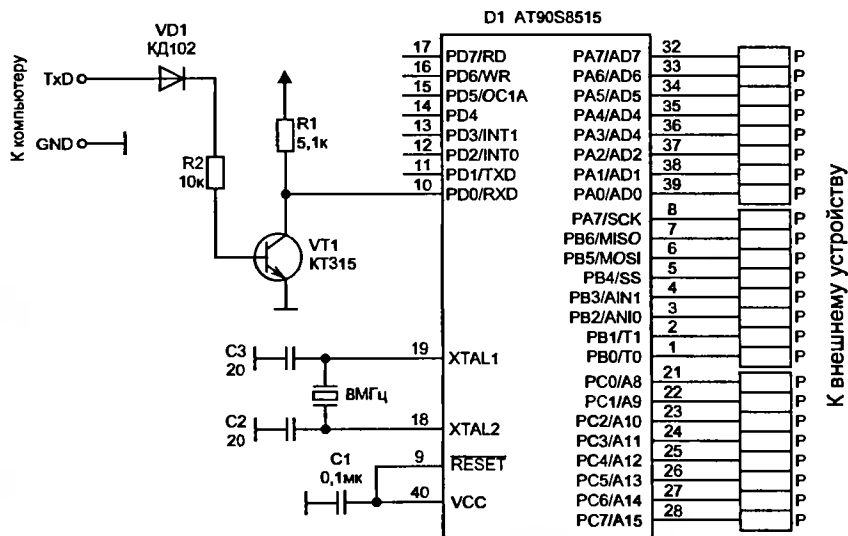


Рис. 6.15. Электрическая принципиальная схема

тверждений не возвращается, запросить состояния выводов компьютера тоже не может, хотя программа микроконтроллера поддерживает такой режим работы.

Уровни последовательного порта преобразуются в TTL с помощью схемы на транзисторе VT1. Если вы собираетесь использовать эту схему для управления устройствами, работающими с высоким напряжением, вместо транзистора следует использовать оптопару.

При работающей программе микроконтроллера на выводе 17 должны быть импульсы с частотой около мегагерца — это сделано для упрощения отладки.

После включения питания все порты переводятся в третье состояние.

Комплект программ для этой конструкции находится на компакт диске, прилагаемом к книге.

Автор: Сафонников В. В. (E-mail: svv@ufanet.ru).

## 6.14. Таймер

### 1. Краткое описание и назначение прибора

Таймер представляет собой электронное устройство для включения и выключения в заданное время в соответствии с расписанием подключенных к нему электроприборов. Расписание набирается с помощью специальной программы на компьютере, а затем загружается в таймер через последовательный порт. В качестве корпуса лучше всего подходит удлинитель с несколькими розетками.

### 2. Схема конструкции показана на рис. 6.16.

### 3. Программа

Для компилирования программы использовался AVR macro assembler version 1.21 и файл 2313def.inc с описанием периферии микроконтроллера, оба они доступны на сайте Atmel. Программа может быть перенесена для работы с другими микроконтроллерами семейства AVR. Для этого замените файл.inc, а константе END\_DATA\_AREA присвойте новое значение в соответствии с размером доступной памяти. Текст программы:

```
.INCLUDE "2313def.inc" ; @ 8 МГц
```

```
.DEF tmp=r0
```

```
.DEF SSREG=r1
```

```
.DEF tem=r25
```

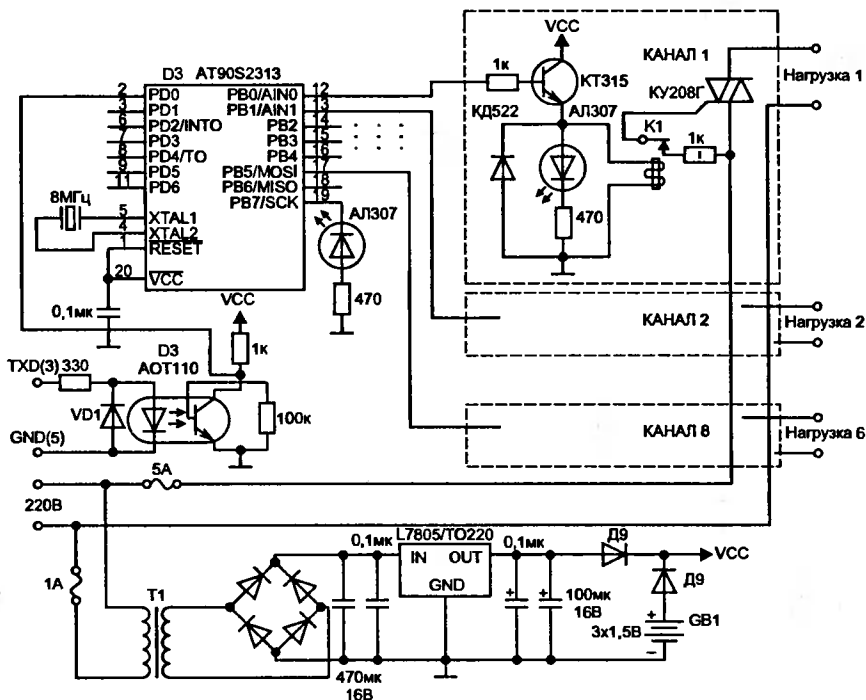
```
.DEF temp=r26
```

```
.DEF TheByte=r27
```

```
.DEF Time0=r16
```

```
.DEF Time1=r17
```

```
.DEF Time2=r18
```



**Рис. 6.16. Электрическая принципиальная схема**

```
.DEF TimeA=r19
.DEF TimeB=r20
.DEF Flags=r21
.EQU END_DATA_AREA=$DB ; 41 запись общей длиной 123 байта

.CSEG

.ORG 0
    rjmp Reset

.ORG OC1addr          ; $0004
    nop

.ORG OVf1addr         ; $0005
    in    SSREG, SREG

    inc   Time0
    brne  NoOvf
    inc   Time1
    brne  NoOvf
    inc   Time2

NoOvf:    ldi    r30, $60
```

```

ldi    r31,0
LoopCheck:      ; Loop of verify time for all records
set      ; Флаг T указывает на совпадение текущего и заданного времени
ld    tmp,Z+      ; Чтение младшего байта времени
cp    tmp,Time0
breq    E1
clt
E1:  ld    tmp,Z+      ; Чтение среднего байта времени
cp    tmp,Time1
breq    E2
clt
E2:  ld    tmp,Z+      ; Чтение последней тетрады времени (биты 0-3)
brtc    NoMath      ; и типа события (биты 4-7)
mov    tem,tmp
eor    tem,Time2
andi    tem,15
brne    NoMath

mov    tem,tmp      ; Время совпало, выполняем заданное действие
andi    tem,$F0
cpi    tem,$00
brne    Off0
cbi    PORTB,0      ; Отключение канала 0
Off0: cpi    tem,$10
brne    On0
sbi    PORTB,0      ; Включение канала 0
On0:  cpi    tem,$20
brne    Off1
cbi    PORTB,1      ; Отключение канала 1
Off1: cpi    tem,$30
brne    On1
sbi    PORTB,1
On1:  cpi    tem,$40
brne    Off2
cbi    PORTB,2
Off2: cpi    tem,$50
brne    On2
sbi    PORTB,2
On2:  cpi    tem,$60
brne    Off3
cbi    PORTB,3
Off3: cpi    tem,$70
brne    On3
sbi    PORTB,3
On3:  cpi    tem,$80
brne    Off4
cbi    PORTB,4
Off4: cpi    tem,$90
brne    On4
sbi    PORTB,4
On4:  cpi    tem,$A0

```

```

    brne Off5
    cbi  PORTB, 5      ; Отключение канала 5
Off5: cpi  tem, $80
    brne On5
    sbi  PORTB, 5      ; Включение канала 5
On5:  cpi  tem, $C0
    brne Off6
    andi Flags, $FE    ; Отключение канала 6 (используется как будильник)
Off6: cpi  tem, $D0
    brne On7
    ori  Flags, 1      , Включение канала 6
On7:  cpi  tem, $E0
    breq TimeReset    ; Обнуление счетчика времени
    cpi  tem, $F0
    brne NoMath
TimeReset:                ; Если биты 4-7 равны 1, то сбрасываем счетчик времени
    clr  Time0
    clr  Time1
    clr  Time2
NoMath:
    cpi  r30, END_DATA_AREA
    breq Loop_Check
    rjmp LoopCheck
Loop_Check:
    ; Включение секундного светодиода
    clr  TimeB
    sbi  PORTB, 7
    out  SREG, SSREG
    reti
Reset:
    ; Установка указателя стека на максимальный адрес SRAM
    ldi  r30, $DF
    out  SPL, r30
    ; Настройка направления работы линий порта B
    ldi  r30, $FF
    out  DDRB, r30
    ldi  r30, $7E
    out  DDRD, r30
    ; Режим работы таймера 1 со сбросом и прерыванием при совпадении
    ; Частота прерываний равна 8 000 000 / 256 / 31250 (7A12) = 1 Гц
    ldi  r30, 12
    out  TCCR1B, r30
    ldi  r30, $7A
    out  OCR1AH, r30
    ldi  r30, $12
    out  OCR1AL, r30
    ldi  r30, $C0
    out  TIMSK, r30
    ; Настройка WDT на 2 секунды

```

```

ldi    r30,$0F
out    WDTCR,r30
; Настройка UART на прием с параметрами 9600 8N1
ldi    r30,$18
out    UCR,r30
ldi    r30,$33
out    UBRR,r30
; Обнуление счетчика времени
clr    Time0
clr    Time1
clr    Time2

; Задание тестовых точек включения (после отладки схемы убрать)
ldi    r30,$05
sts    $60,r30
ldi    r30,$00
sts    $61,r30
ldi    r30,$10
sts    $62,r30

ldi    r30,$07
sts    $63,r30
ldi    r30,$00
sts    $64,r30
ldi    r30,$00
sts    $65,r30

ldi    r30,$0F
sts    $66,r30
ldi    r30,$00
sts    $67,r30
ldi    r30,$10
sts    $68,r30

clr    Flags
sei

UnlessLoop:
    rcall GetByte          ; Прием сообщения говорящего о начале данных
    cpi    TheByte,'T'
    brne  UnlessLoop
    rcall GetByte
    cpi    TheByte,'i'
    brne  UnlessLoop
    rcall GetByte
    cpi    TheByte,'m'
    brne  UnlessLoop
    rcall GetByte
    cpi    TheByte,'e'
    brne  UnlessLoop
    rcall GetByte
    cpi    TheByte,'r'

```

```

brne UnlessLoop

clr Time0           ; Обнуление счетчиков времени
clr Time1
clr Time2
ldi r28,$60
ldi r29,0

```

LoopRX:

```

rcall GetByte       ; Чтение байта данных
st Y+,TheByte       ; Сохранение считанного байта в памяти
cpi r28,END_DATA_AREA
brlo LoopRX
rjmp UnlessLoop
ser Time0           ; Обнуление счетчиков времени
ser Time1
ser Time2
ldi r28,$10
out TCNT1L,r28
ldi r28,$7A
out TCNT1H,r28

```

; Процедура чтения байта с UART

; Программа все свое свободное время находится внутри нее

GetByte:

```

wdr                ; Сброс сторожевого таймера
inc TimeA          ; Увеличение асинхронного счетчика времени
brne Early
inc TimeB
brne Early
cbi PORTB,7        ; Выключение секундного светодиода

```

Early:

```

sbrs Flags,0
rjmp NoBell
mov temp,TimeB     ; Звуковой сигнал включен
andi temp,$E7
breq NoBell
sbis PORTB,6       ; Инвертирование состояния вывода PB.6
rjmp NoCBI
cbi PORTB,6
rjmp NoBell

```

NoCBI: sbi PORTB,6

NoBell:

```

sbis USR,RXC
rjmp GetByte
in TheByte,UDR
ret

```

; Протокол управления таймером

; Формат загрузки расписания управления каналами от компьютера

; \* Настройки порта 9600 8N1

; \* Загрузка производится одним блоком, состоящим из 41-й записи

- ; \* Перед блоком с записями надо отправить строку Timer
- ; \* Каждая запись состоит из 3 байтов, сначала передается младший байт времени включения, затем средний, затем байт, состоящий из двух частей: младшие 4 бита – это самая старшая тетрада времени, бит 4 – это состояние, в которое будет переведен канал, и оставшиеся 3 бита – это номер канала. Если номер канала равен 7, то будет обнулен внутренний счетчик времени таймера и выполнение программы начнется сначала
- ; \* Время считается с момента загрузки новой программы в секундах

EXIT

#### 4. Технические характеристики

Максимальное количество нагрузок, поддерживаемое программой — 7, но оно может быть меньше, в зависимости от конкретной реализации схемы. Седьмой канал, в отличие от остальных, при срабатывании выдает не постоянное напряжение единицы, а импульсы звуковой частоты. Это позволяет использовать его, например, в качестве будильника. Допустимая мощность нагрузок тоже определяется схемой, например если используются симисторы КУ208, установленные на радиатор, то она составляет 1 кВт на каждый канал.

Память микроконтроллера, установленного внутри таймера, позволяет хранить расписание, состоящее из 41-го (для AT90S2313) события. Максимальное время от загрузки расписания до выполнения события не может быть больше 220 секунд (немного больше 7 дней). Точность задания времени события 1 секунда.

В схеме таймера предусмотрен автономный источник питания, который позволяет сохранить загруженное в него расписание и продолжать считать точное время даже при отсутствии напряжения в питающей сети. Ток, потребляемый от батареек, в этом режиме составляет около 3 мА.

#### 5. Управляющая программа и инструкция (рис. 6.17)

Программе не требуется для работы дополнительных библиотек, для хранения некоторых настроек используется системный реестр, расписание хранится в файле Timer.sav в текущем каталоге (текстовый файл).

Дополнительная информация о работе с программой и о самом устройстве находится в файле справки.

Комплект программ для проекта находится на компакт диске, прилагаемом к книге.

Автор: Сафонников В. В. (E-mail: svv@ufanet.ru).

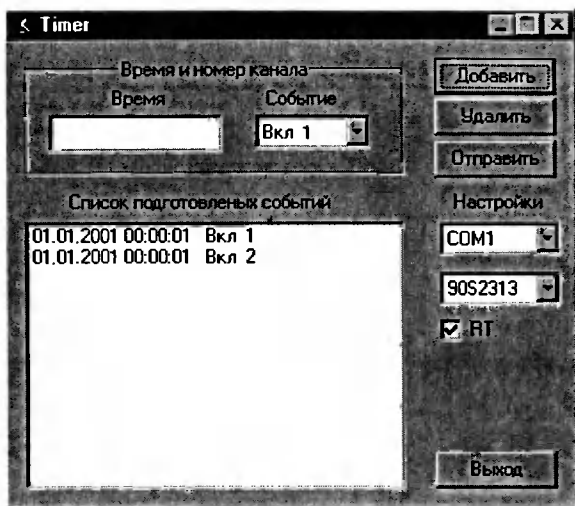


Рис. 6.17. Вид окна управляющей программы

## 6.15. Универсальный параллельный адаптер

### 1. Краткое описание

Универсальный параллельный адаптер предназначен для подключения к компьютеру различных устройств с цифровыми входами. Например, он может использоваться для записи информации в ПЗУ, прямого управления шаговыми двигателями, для налаживания различных электронных схем в качестве эмулятора и т. д. При наличии соответствующего программного обеспечения многие из перечисленных задач можно выполнить, используя только порт компьютера, но при этом возникает вполне реальный риск выхода порта из строя, так как его выходы не имеют защиты и рассчитаны на подключение только одного вывода, а для ремонта может потребоваться замена материнской платы. Кроме того, для подключения к параллельному порту чего-либо необходимо предварительно выключить компьютер. Адаптер устраняет эти проблемы и позволяет думать в первую очередь о разрабатываемой схеме, а не о том, как бы в процессе ее создания не спалить компьютер.

## 2. Схема

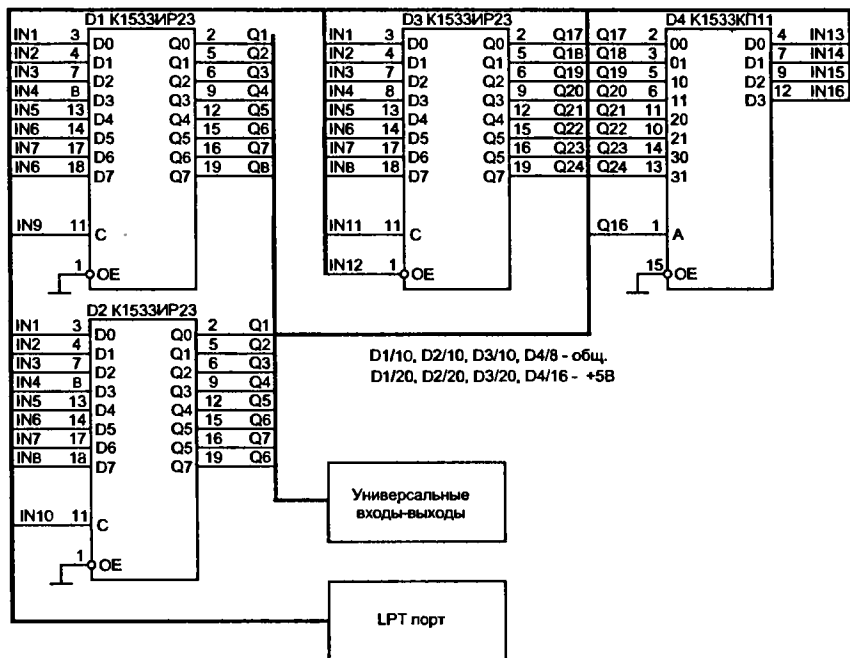


Рис. 6.18. Электрическая принципиальная схема адаптера

Идея использовать параллельный порт для выдачи и приема цифровых сигналов с ТТЛ-уровнями не нова. Предлагаемый здесь адаптер отличается простотой, при возможностях достаточных для большого количества применений. К тому же, если вам через какое-то время понадобится увеличить количество входов-выходов, можно просто собрать такую же схему и подключить ее по приведенной ниже таблице. Хотя если предполагается, что выводов одной схемы сразу будет недостаточно, лучше использовать более мощный вариант.

Схема состоит из трех регистров и одного мультиплексора. Все регистры включены по одинаковой схеме, за исключением третьего, его выходы могут быть переведены в высокоомное состояние, поэтому к нему также подведен управляющий сигнал разрешения включения выходов OE. Информационные входы всех регистров объединены и подключены к соответствующим выходам параллельного порта

компьютера, так как используется ТТЛШ серия, то допустимо нагружать один выход порта на несколько входов микросхем. Для стробирования используются управляющие линии порта, подключенные ко входам С регистров. Для увеличения количества входов используется мультиплексор D4.

### 3. Подключение к компьютеру и внешнему устройству

Схема подключается к параллельному порту, также необходимо подвести питание +5 В к микросхемам, лучше всего для этого использовать блок питания компьютера. В моем варианте собранная схема находится внутри компьютера, подключается к внутреннему разъему LPT-порта на системной плате, для питания использует 4-контактный разъем, а рабочие выходы выведены на 32-контактный разъем, вмонтированный в заглушку от отсека 5,25 на передней панели. На этот же разъем выведены напряжения питания +5, +12 В.

При необходимости увеличить количество выводов можно собрать второй такой же блок и подключить его в соответствии с приведенной ниже таблицей к первому, подключенному к компьютеру. При этом появятся дополнительно несколько входов и выходов, но время доступа увеличится.

#### Подключение к параллельному порту

Вывод порта	Тип	Название вывода	Цель адаптера
2	0	D0	IN1
3	0	D1	IN2
4	0	D2	IN3
5	0	D3	IN4
6	0	D4	IN5
7	0	D5	IN6
8	0	D6	IN7
9	0	D7	IN8
1	0	-STROBE	IN9
14	0	-AUTO FD	IN10

Подключение к параллельному порту (продолжение)

Вывод порта	Тип	Название вывода	Цепь адаптера
16	0	-INIT	IN11
17	0	-SLCT IN	IN12
10	1	-ACK	IN13
11	1	BUSY	IN14
12	1	PE	IN15
13	1	SLCT	IN16
18...25	—	GND	общ.

Подключение второго адаптера для увеличения количества выводов

Первый	Второй	Первый	Второй
01	11	09	19
02	12	010	110
03	13	011	111
04	14	012	112
05	15	017	113
06	16	018	114
07	17	019	115
08	18	020	116

Для подключения к внешнему устройству используются цепи 01...024, из них 01...016 являются обычными выходами, а 017...024 могут использоваться как входы или выходы. Цепь 016 параллельно используется для внутренних нужд.

4. Программирование

Программа должна писаться для каждого конкретного случая использования устройства отдельно, поэтому автор не приводит здесь никаких вариантов, а только рассматривает основные принципы программного управления схемой.

Практически любой язык программирования имеет функции, позволяющие записать число по заданному адресу в порт ввода-вывода. Управление схемой осуществляется через вызовы таких функций. Для записи 8-битового числа в буферный регистр схемы необходимо записать его в регистр данных параллельного порта, затем записать в регистр управления любое число, имеющее в соответствующем разряде единицу (соответствующем выводу С выбранного буферного регистра), и затем в него же ноль. Для чтения 4 битов данных достаточно просто прочитать регистр состояния порта, для чтения остальных битов предварительно измените состояние линии O16. Необходимо учитывать, что некоторые входные и выходные линии порта проинвертированы.

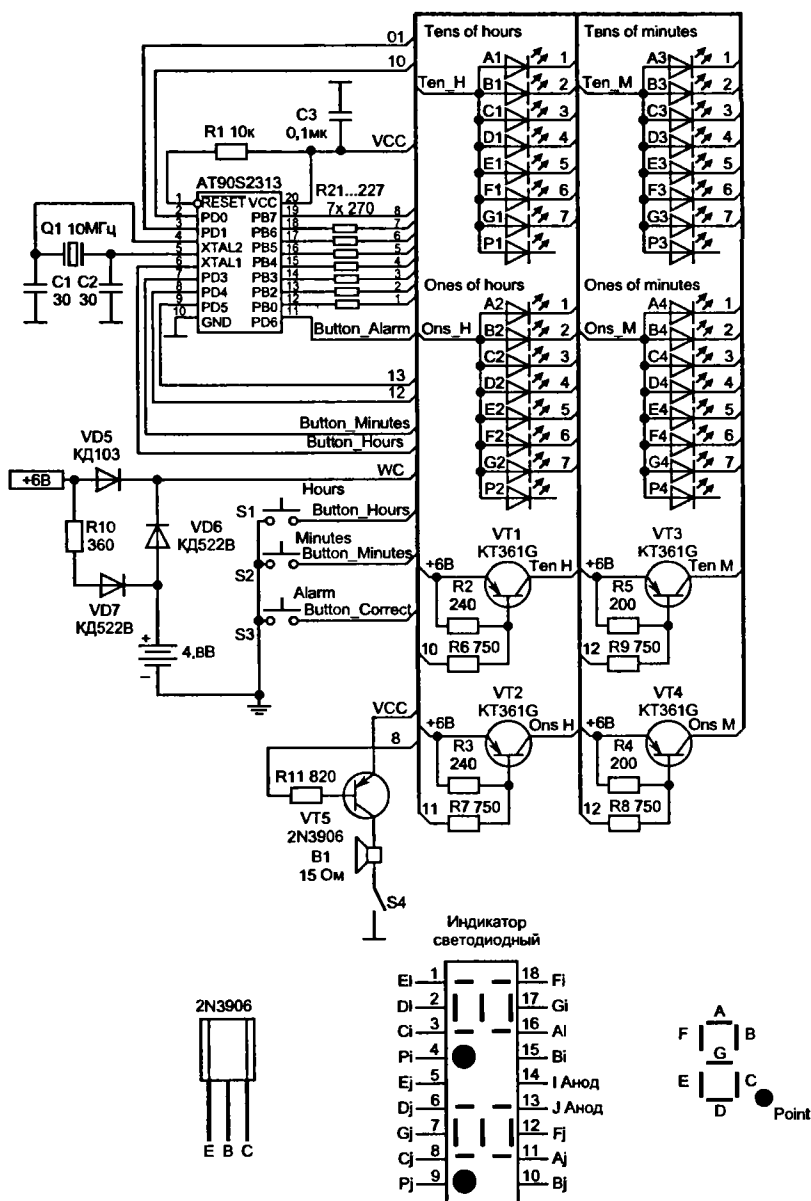
Адреса регистров для LPT1 приведены в таблице (верно для большинства компьютеров, но для корректного определения адресов следует использовать данные BIOS):

Регистр данных	378h
Регистр состояния	379h
Регистр управления	37Ah

Автор: Сафонников В. В. (E-mail: [svv@ufanet.ru](mailto:svv@ufanet.ru)).

## **6.16. Электронные часы с будильником на AT90S2313-10PI**

В данном проекте описаны электронные часы с будильником, выполненные на AVR-микроконтроллере типа AT90S2313-10PI. Часы имеют светодиодный индикатор отображения текущего времени и встроенный аккумулятор для поддержания хода при пропадании сетевого напряжения, что очень актуально в условиях непрекращающегося экономического кризиса. Устройство содержит минимум комплектующих и имеет несложную электрическую схему. Часы были испытаны автором на протяжении нескольких месяцев, что показало их надежность и работоспособность.



## Краткое описание устройства часов

Предлагаемый вашему вниманию проект был выполнен «по ходу дела» при освоении автором микроконтроллеров семейства AVR фирмы Atmel. Один из этих «жучков» и был использован при разработке часов. Выбор именно AT90S2313-10PI объясняется широкой доступностью и невысокой ценой этого кристалла, а также наличием в нем памяти программ объемом 2 Кб и программно реализованного стека.

Как видно, микроконтроллер является основной и единственной микросхемой, используемой в данной разработке. Для задания тактовой частоты контроллера используется кварцевый резонатор на 10 МГц, но управляющую программу очень легко переделать и для резонаторов с другими частотами. В качестве устройства отображения использованы два индикатора красного цвета свечения с общим анодом, каждый индикатор состоит из двух цифр с десятичными точками. Цифры имеют отдельные аноды. Можно применить любые индикаторы с общим анодом, лишь бы ток сегмента не превышал 20 мА и каждая цифра имела бы собственный анод. Рекомендуется выбирать индикаторы с большими цифрами, тогда часы будут хорошо видны в темноте.

Индикация текущего времени осуществляется динамически, в данный конкретный момент времени отображается лишь одна цифра, что позволяет значительно снизить аппаратные затраты. Происходит это так. Аноды каждой из четырех цифр обоих индикаторов являются раздельными, что позволяет в данный момент времени подключить к источнику питания только один анод и отобразить одну цифру. Для этого часы имеют четыре транзисторных ключа, выполненных на транзисторах типа КТ361Е (VT1...VT4), и резисторах (R2...R9). Ключи управляются микроконтроллером, причем соответствующий ключ открыт, если на выводе контроллера присутствует логический ноль. Одноименные сегменты всех четырех цифр соединены вместе и через токоограничивающие резисторы R21...R27 подключены к выводам порта В (выводы PB.0...PB.6). Десятичная точка не используется, она «принесена в жертву» добавленному в часы будильнику и всегда выключена. Управляющая программа один за другим подключает цифры к источнику питания, и одновременно на соответствующие выводы порта В выставляется код отображаемой

цифры. Поскольку сканирование индикатора происходит очень быстро, мерцание цифр становится незаметным.

Для перевода минут, часов и установки будильника используются кнопки S1...S3. Обновление показаний индикатора происходит каждую секунду. При нажатии более чем на одну кнопку управляющая программа игнорирует нажатие кнопок. Для установки будильника следует нажать кнопку ALARM и подождать двукратного звукового сигнала (не более секунды) для входа в режим будильника. Для перехода обратно в режим часов кнопку ALARM нужно удерживать до подачи однократного сигнала. В качестве выхода для сигнала будильника используется вывод PB.7 порта В, а в качестве усилителя — схема на транзисторе VT5. В роли сигнализатора применен звонок от импортных часов сопротивлением около 15 Ом. Для отключения будильника используется выключатель S4 (лучше всего — кнопка с фиксацией).

Питаются часы от стабилизированного источника напряжением 6 В. Причем индикация работает только при работе часов от сети. Ток потребления при наличии индикации — около 80 мА. При работе от аккумуляторов (четыре аккумулятора типа Д-0,26) индикация отключается, но часы продолжают идти и функционирует будильник. Диоды D5...D7 обеспечивают правильное использование источников питания при работе от сети и от аккумуляторов, сами же аккумуляторы при работе часов от сети заряжаются через резистор R10. Поскольку при отсутствии индикации часы потребляют ток около 6 мА, предложенный UPS способен поддерживать работу часов более суток, что чрезвычайно удобно. Лично я не люблю наводить часы всякий раз после броска сетевого напряжения. Кстати, это явилось одной из движущих сил данного проекта.

### **Краткое описание управляющей программы**

Управляющая программа часов написана на ассемблере.

Сразу после включения часов программа разрешает прерывания, настраивает порты контроллера соответствующим образом и устанавливает указатель стека на старшие адреса внутренней памяти данных (стек растет «сверху вниз», как в семействе 80X86). Далее программа переводит устройство в режим часов, настраивает будильник

на 6 часов 55 минут, после чего запускает цикл сканирования индикатора и цикл счета времени. Вся дальнейшая работа программы состоит в реализации пустого цикла, из которого ее выводят запросы прерываний от таймеров-счетчиков и в который она снова возвращается после завершения обработчика.

Основой программы являются два обработчика прерываний от таймеров. Обработчик прерывания от таймера-счетчика T/C0 используется для динамической индикации, а обработчик от таймера-счетчика T/C1 считает время. Естественно, можно было бы организовать сканирование индикатора и без привлечения прерывания от таймера и применить более дешевый кристалл, например AT90S1200. Но непривычность работы с аппаратным стеклом (автор много лет до этого занимался 80X86), недостаточный объем памяти и незначительное отличие между этими контроллерами в цене говорят явно не в пользу такого решения.

Обработчик счета времени вызывается каждую секунду. Он является более приоритетным, чем обработчик сканирования индикатора. В этом обработчике также выполняется определение состояния клавиш часов и при необходимости — перевод времени, переход в режим часов/будильника, а также проверяется равенство текущего времени установкам будильника. Обработчик сканирования индикатора вызывается около 1600 раз в секунду и по очереди отображает каждую из цифр текущего времени, т. е. каждая цифра появляется на индикаторе примерно 400 раз в секунду. Кстати, изменяя в разумных пределах частоту сканирования, легко регулировать яркость свечения индикатора.

Дальнейшие пояснения работы программы вы сможете получить, просмотрев ее текст с подробными комментариями.

## Резюме

К сожалению, автору не удалось численно определить точность хода часов. Можно сказать лишь одно — за месяц часы отстали примерно на минуту, что, согласитесь, вполне приемлемо для устройства такого класса.

Файлы, относящиеся к проекту, находятся на компакт-диске, прилагаемом к книге.

Автор: Игорь Коваль (E-mail: kovigor@yahoo.com).

## **6.17. Подключение внешнего статического оперативного запоминающего устройства**

Разработчики устройств на микроконтроллерах постоянно сталкиваются с задачей временного хранения данных, однако объема внутреннего SRAM (Static Random Access Memory — статическое запоминающее устройство с произвольным доступом, далее — память) зачастую недостаточно.

Для облегчения решения такой задачи микроконтроллер AT90S8515 (AT90LS8515) снабжен интерфейсом для подключения внешней памяти объемом до 64 Кбайт.

### **Интерфейс для подключения внешней памяти**

Интерфейс включает в себя:

- порт A: шина младшего байта адреса / шина данных;
- порт C: шина старшего байта адреса;
- контакт ALE (Address Latch Enabled): разрешение фиксации адреса;
- контакты RD и WR: стробы записи и считывания.

Для инициализации работы интерфейса надо установить (в единицу) бит SRE (Static RAM Enable — разрешение работы с внешней памятью) регистра MCUCR (MCU Control Register — регистр управления микроконтроллера). Теперь при использовании стандартных команд работы с памятью (команды LD, LDD, LDS, ST, STD и STS) микроконтроллер может обращаться к ячейкам памяти, адреса которых лежат в диапазоне \$0060...\$FFFF (знак «\$» используется в тексте для обозначения шестнадцатеричных чисел, так же, как это должно быть записано в программе на Ассемблере для AVR Studio; указанный диапазон адресов в десятичной форме будет иметь вид 96...65535).

Свободно размещать данные в памяти микроконтроллера AT90S8515 можно, начиная с адреса \$0060, как при работе с внутренней оперативной памятью, так и при использовании интерфейса внешней памяти, поскольку адреса оперативной памяти \$0000...\$001F

заняты регистрами общего назначения  $r0...r31$  (десятичное число 31 равно шестнадцатеричному числу  $\$001F$ ), адреса  $\$0020...\$005F$  отведены для регистров ввода-вывода.

### *Замечание.*

Если до инициализации интерфейса линии порта *A*, порта *C* и контакты *PD6 (WR)* и *PD7 (RD)* порта *D* были запрограммированы как линии ввода или вывода, установка бита *SRE* перепрограммирует эти линии для работы с внешней памятью.

Если бит *SRE* сбросить (в ноль), работа с внешней памятью прекращается, устанавливается обычный режим работы портов *A*, *C* и контактов *PD6* и *PD7* порта *D*, а область оперативной памяти, к которой может обращаться микроконтроллер *AT90S8515* ограничивается диапазоном  $\$0060...\$025F$  (всего 512 байт).

*Примечание.* При работе в диапазон адресов, например,  $\$2220...\$222F$  количество адресуемых ячеек не 15 ( $\$222F - \$2220 = \$F$ ), а 16 ( $\$F + 1$ ), так как обращение происходит и к ячейке с адресом  $\$2220$ .

Порт *A* служит как для формирования младшего байта адреса, так и для передачи данных, в то время, как порт *C* поддерживает на своих линиях старший байт адреса в течение всего цикла обращения к внешней памяти.

Для фиксации младшего байта адреса необходимо использовать дополнительный элемент хранения с записью положительным уровнем. Для этого подойдет 8-разрядный регистр *74HCT573N*. При высоком уровне сигнала на контакте *ALE* младший байт адреса, сформированный на восьми линиях порта *A*, записывается в регистр. При низком уровне на контакте *ALE* через порт *A* передаются данные. Сигналы на контактах *WR* и *RD* микроконтроллера активны только во время обращения к внешней памяти.

Обращение к внутренней памяти происходит за три цикла микроконтроллера.

Обращение к внешней памяти происходит за три цикла микроконтроллера, однако можно удлинить время обращения на один дополнительный цикл, установив бит *SRW (External SRAM Wait State* — состояние ожидания внешней памяти) в регистре *MCUCR*.

### Пример подключения внешней оперативной памяти к микроконтроллеру AT90S8515

Фрагмент схемы, в котором реализовано подключение внешней памяти 8Кх8 (8 Кбайта) к микроконтроллеру, приведен на рис. 6.20. Для подключения с минимальным количеством дополнительных элементов пригодны микросхемы памяти, имеющие двунаправленную 8-разрядную шину данных (D0...D7), шину адреса и инверсные входы управления записью и считыванием (WE, OE).

На фрагменте схемы к микроконтроллеру AT90S8515 (DD2) подключена микросхема внешней памяти HM62256 (DD5) с организацией 32Кх8, из которых используется только 8Кх8, что требует использования тринадцати линий адреса (A0...A12) из пятнадцати имеющихся. Свободные контакты A13 и A14 шины адреса микросхемы DD5 соединены с общим проводом. Замечу, что эти контакты можно

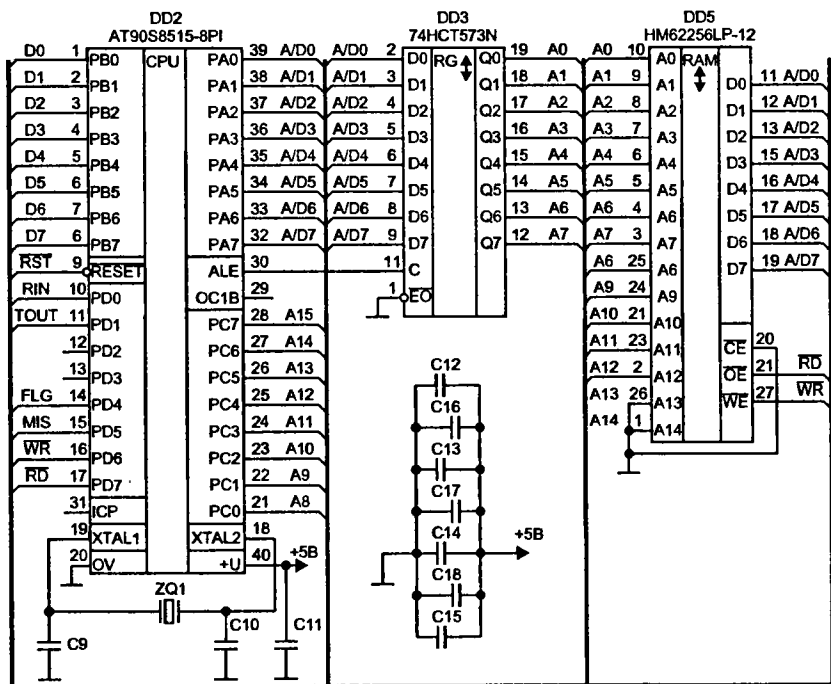


Рис. 6.20. Подключение внешней памяти

было соединить и с линией питания, что никак не сказалось бы ни на работе, ни на программе, физически использовалась бы другая область ячеек микросхемы памяти.

Не использующиеся в приведенной схеме для работы с внешней памятью линии адреса микроконтроллера A13, A14 и A15 могут использоваться, скажем, для обращения к дополнительным регистрам, обеспечивающим передачу данных к внешним устройствам.

### **Установка адреса**

По высокому уровню сигнала на контакте ALE микроконтроллера в регистр 74НСТ573 (DD3) производится запись младшего байта адреса, сформированного микроконтроллером на линиях A/D0...A/D7. Записанный в регистр младший байт адреса появляется на выходах регистра, соединенных с адресными входами микросхемы памяти (линии A0...A7).

На контактах порта C микроконтроллера формируется старший байт адреса (линии A8...A15).

Регистр 1533ИР33 является функциональным аналогом регистра 74НСТ573, однако они выполнены по разной технологии. Опыт использования регистра 1533ИР33 для фиксации младшего байта при работе микроконтроллера с внешней памятью показал, что при записи — считывании массива (0, 1, ..., 254, 255) появляются сбои, число которых сокращается при подключении конденсатора (десятки пикофарад) между контактом C регистра и общим проводом, использование же регистра 74НСТ573 полностью решало проблему и без конденсатора. Поэтому следует аккуратно относиться к замене регистра.

Для качественного тестирования памяти удобно использовать подключение контроллера к компьютеру, используя UART микроконтроллера (Universal asynchronous receiver-transmitter — универсальный асинхронный приемопередатчик).

### **Считывание данных из внешней памяти**

Если производится считывание данных из внешней памяти, то после того, как адрес установлен, линии порта A переводятся в режим ввода данных в микроконтроллер, на линии RD микроконтроллер формирует импульс отрицательной полярности. По сигналу RD шина данных D0...D7 микросхемы памяти DD5 переводится в режим

вывода данных, и по линиям A/D0...A/D7 из адресуемой ячейки в порт A микроконтроллера передаются хранимые данные.

### ***Запись данных во внешнюю память***

Для записи данных во внешнюю память, после того, как адрес установлен, линии порта A переводятся в режим вывода данных из микроконтроллера, на линии WR микроконтроллер формирует импульс отрицательной полярности. По сигналу WR шина данных D0...D7 микросхемы памяти DD5 переводится в режим ввода данных, и по линиям A/D0...A/D7 из порта A микроконтроллера в адресуемую ячейку внешней памяти передаются данные для хранения.

### ***Программный доступ к оперативной памяти***

Начнем с простой программы обращения к оперативной памяти. Для этого необходимо, чтобы на вашем компьютере была установлена среда разработки AVR Studio корпорации Atmel (<http://www.atmel.com>), в которой создаются и отлаживаются программы для микроконтроллеров AVR.

Кроме того, в любую программу обязательно включается файл, определяющий регистры и имена битов микроконтроллера, для которого пишется программа. Для микроконтроллера AT90S8515 — это файл 8515def.inc, для AT90S4433 — файл 4433def.inc. Все файлы с расширением .inc на момент написания главы хранились в самораспаковывающемся файле avr000.exe.

Найти ссылки на файл для установки AVR Studio 3.xx и на файл avr000.exe можно по адресу: <http://www.atmel.com/atmel/products/prod203.htm>.

Инструкции и картинки, размещенные далее, соответствуют версии AVR Studio 3.51.

Создайте директорию, в которой будут размещаться файлы вашего проекта, например, C:\Avr\Try. Запустите скачанный файл avr000.exe, а появившийся в результате распаковки файл 8515def.inc скопируйте в созданную директорию C:\Avr\Try.

Запустите AVR Studio, на экране появится окно AVR Studio (обрамленное сверху синей полосой окно с надписью AVR Studio).

Создадим новый проект: Project | New.

Запись вида «Project | New» обозначает, что надо установить указатель мышки на меню Project, расположенном в первой строке окна AVR Studio, в открывшемся окне установить указатель мышки на слове New и нажать левую клавишу мышки. Далее вместо слов «нажать левую клавишу мышки» будем употреблять слова «щелкнуть» или «выбрать»; если требуется нажатие правой клавиши мышки — будем использовать слова «щелкнуть правой клавишей мышки».

Начальная строка окна содержит его наименование (AVR Studio ...), только следующая строка доступна при работе, ее будем считать первой строкой окна.

В появившемся окне Select new project (рис. 6.21):

- введите имя проекта в строке ввода Project name (например, Memory);
- впишите вручную созданную в Windows директорию для проекта C:\Avr\Try в строку ввода Location или найдите эту директорию в дереве каталога, щелкнув по кнопке справа от строки ввода;
- выберите строку AVR Assembler для определения типа проекта в окне Project type;
- щелкните мышкой по кнопке OK.

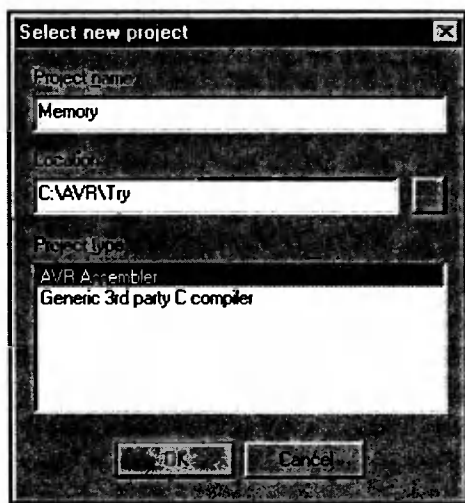


Рис. 6.21. Окно Select new project

Теперь на экране активно окно Project : Memory (рис. 6.22).

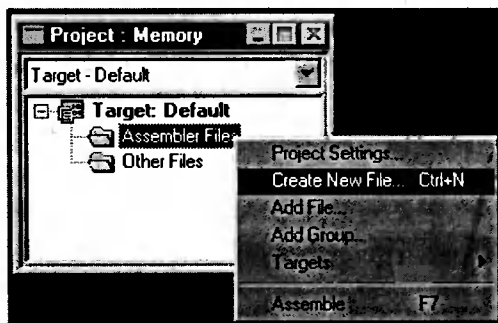


Рис. 6.22. Окно Project

Создадим новый файл, который будет содержать программу на Ассемблере:

- щелкните правой клавишей мышки в окне Project : Memory;
- в появившемся окне выберите строку Create New File;
- в строку Name открывшегося окна Create new file (рис. 6.23) введите имя файла программы, например, SRAM.asm, затем щелкните по кнопке ОК.

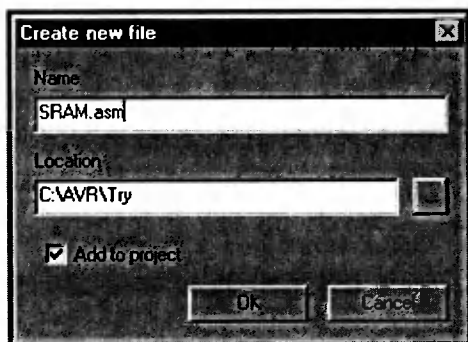


Рис. 6.23. Окно Create new file

В появившемся окне SRAM.asm наберите следующую программу, не игнорируя знаков препинания в начале строк.

Команды лучше писать строчными буквами, тогда они выделяются цветом. В тексте программы команды и метки выделены про-

писными буквами только для снижения вероятности ошибок при наборе программы. В AVR Studio 3.51 нет различия между прописными и строчными буквами. Между меткой, командой и ее операндами удобно вставлять знак табуляции вместо пробела, тогда программа лучше читается.

```
;          Программа SRAM.asm
;          Обращение к внешней памяти
;          =====
.INCLUDE    "8515def.inc"
; Вставка содержимого файла 8515def.inc в нашу программу
; Файл 8515def.inc не обязательно располагать в директории
; проекта тогда следует указать и путь к нему, например:
; include "d:\Def_dir\8515def.inc"

.DEF tmp    =r16 ; Регистру r16 присвоить имя tmp
                ; (значение переменной tmp будет храниться в
                ; регистре общего назначения r16)

.DEF cnt    =r16 ; Переменная cnt - счетчик цикла
; Замечание: одному регистру в программе могут присваиваться
; разные имена, хотя в данном случае в этом и не было
; необходимости, микроконтроллер имеет 32 регистра, но
; переменных в программе обычно гораздо больше

.EQU ArSize    =10
; Константа ArSize=10 - используем эту константу
; для определения размера массива, записываемого в память

.EQU aArBgn    =$0170
; Используем константу aArBgn как адрес начальной ячейки
; для хранения массива

RESET:        IN      tmp,MCUCR
; Ввод содержимого регистра MCUCR в регистр tmp. Физические
; адреса регистров MCUCR,ZL,ZH и константы SRE и SRW,
; используемые в программе, определены в файле 8515def.inc
; если по директиве include файл 8515def.inc не будет найден,
; команды с операндами MCUCR,ZL,ZH,SRE и SRW дадут ошибки

        ORI      tmp,(1<<SRE)+(1<<SRW)
; Операция V (логическое ИЛИ) содержимого регистра и константы.
; Для работы с внешней памятью надо установить
; биты SRE и SRW в регистре MCUCR (установка SRW - если
; нужно продлить состояние ожидания
; для "медленной" микросхемы памяти).
; Открыв файл 8515def.inc, мы обнаружим, что SRE=7, SRW=6,
; значит величина, которую надо занести
; в регистр MCUCR в двоичном коде: 11bb bbbb,
; b - это значения битов, которые мы не должны изменять
; запись (1<<SRE)+(1<<SRW) равна сумме
```

```
; единицы, сдвинутой влево SRE (или 7) раз (=1000 0000),
; и единицы, сдвинутой влево SRW (или 6) раз (0100 0000)
; Сумма 1000 0000+0100 0000=1100 0000
; операция V (bbbb bbbb V100 0000 =11bb bbbb)
; дает нужный результат
```

```
OUT MCUCR, tmp
```

```
; Вывод содержимого регистра tmp в регистр MCUCR
```

```
LDI ZL, low(aArBgn)
```

```
LDI ZH, high(aArBgn)
```

```
; LDI - операции загрузки в регистр однобайтной константы
; aArBgn - двухбайтная константа - должна быть загружена
; в пару однобайтных регистров ZH:ZL, которые составляют
; двухбайтный регистр Z. Регистр Z и аналогичные ему двухбайтные
; регистры X и Y используются в операциях с памятью (st, ld, ...).
; aArBgn=$0170, в ZL загрузится low($0170), т. е., младший байт $70
; в ZH загрузится high($0170), то есть, старший байт $01
; в Z (или в ZH:ZL) образуется двухбайтное слово $01:$70.
; то есть, адрес, выбранный для начальной ячейки массива
```

```
LDI cnt, ArSize ; Загрузить в cnt размер массива
```

```
NEXT: ST Z+, cnt
```

```
; ST - операция записи содержимого регистра cnt в ячейку памяти,
; адрес которой - в регистре Z, знак "+" после Z - значит
; с последующим увеличением адреса в регистре Z на единицу.
; В первом цикле данные занесутся по адресу $0170, а Z = $0171;
; во втором цикле - по адресу $0171, Z = $0172 и т. д.
; в память пишется состояние счетчика циклов
```

```
DEC cnt
```

```
; Уменьшить содержимое регистра cnt на единицу
```

```
NOP ; Команда добавляет один пустой цикл
```

```
NOP ; То же
```

```
BRNE NEXT
```

```
; Если бит (флаг) Z в регистре состояния процессора SREG
; (не путать бит Z регистра SREG с регистром адреса Z)
; не установлен (не равен) - перейти на команду с меткой NEXT:
; последняя команда, воздействующая на бит Z
; регистра SREG - DEC cnt. Команды NOP введены для
; демонстрации отсутствия их влияния на бит Z
; если вместо них вставить, например, команду INC tmp, влияющую
; на состояние бита Z, работа программы будет нарушена
```

```
; Далее - блок считывания данных из памяти:
```

```
LDI ZL, low(aArBgn)
```

```
LDI ZH, high(aArBgn)
```

```
LDI cnt, ArSize
```

```
RD_BLK:
```

```
LD tmp, Z+
```

```
; Здесь какие-то действия с tmp, например,  
; передача в компьютер через UART  
DEC    cnt  
BRNE   RD_BLK  
  
RJMP   RESET ; Перейти на команду с меткой RESET:
```

Вы заметили, что имена регистров и константы, определенные во включаемом в программу файле 8515def.inc довольно длинные? Не думайте, что, сокращая используемые имена до одной-двух букв, можно сэкономить время: через пару недель вы не разберетесь в собственной программе! А вот имена MCUCR, SRE, являясь сокращениями английских наименований MCU control register и Static RAM Enable, быстро запоминаются. Те же рекомендации можно предложить и для определяемых вами имен: и в чужой программе можно понять, что ArSize это Array Size (размер массива), а aArBgn — начальный адрес массива.

## Отладка программы

Ассемблируем программу: Project | Assemble. Обращайте внимание на наличие горячих клавиш: на строке Assemble указана горячая клавиша F7. Так как эта операция повторяется часто, удобнее пользоваться именно клавишей F7.

При первом ассемблировании автоматически открывается окно Simulator options. В строке Device этого окна надо выбрать микроконтроллер, на который ориентирована программа, в нашем случае это AT90S8515 with external SRAM. Затем нажмите кнопку OK.

Замечание: для микроконтроллеров с программно-аппаратной поддержкой подключения внешней памяти симулятор программируется как для поддержки внешней памяти (выбрать AT90S8515 with external SRAM), так и для работы только с внутренней памятью (выбрать AT90S8515). Последовательность команд в программе, задающая режим работы с внешней памятью, достаточна лишь для самого микроконтроллера; на работу в режиме отладки в AVR Studio влияет именно выбор устройства в окне Simulator options.

В окне Simulator options следует установить частоту кварцевого резонатора, использующегося вместе с микроконтроллером. Частоту можно либо выбрать из предлагаемого списка, либо ввести вручную,

если надо задать точное значение, например, 6,425 МГц. Это позволит симулятору корректно выводить время работы программы не только в циклах микроконтроллера, но и в единицах времени.

Вызвать окно *Simulator options* вновь можно только после успешного ассемблирования в режиме отладки (переход в режим отладки — по нажатию клавиши F11).

В результате ассемблирования появится окно *Project output*.

Если программа набрана без ошибок, а файл *8515def.inc* находится в рабочей директории проекта — в последней строке окна *Project output* будет сообщение об отсутствии ошибок (*Assembly complete with no errors*).

Интересной информацией в этом окне является:

- сообщение о включении файла *8515def.inc* в программу;
- предостережение: *C:\Avr\Try\SRAM.asm(15): warning: Register already defined by the.DEF directive* — предупреждение о том, что в строке (15) программы директивой *.DEF* назначается имя регистру, которому уже назначено другое имя; хотя программа невелика, считать строки нет необходимости, достаточно щелкнуть мышкой на любой строке, чтобы увидеть ее номер, а также номер колонки, в правом нижнем углу всего окна *AVR Studio*: *Ln15, Col5*;
- общий объем памяти, занимаемый программой (*Total*). Обратите внимание на то, что объем занимаемой памяти указывается в словах (*words*), каждое слово в памяти программы микроконтроллера двухбайтное, поэтому размер программы в байтах будет в два раза больше.

Займемся пошаговой отладкой программы: *Debug | Trace into (F11)*.

*Замечание.* При отладке (смотрите меню *Debug*) удобно пользоваться горячей клавишей F11, когда требуется пошаговая отладка каждой команды, если Вы уже отладили часть программы в предыдущих сеансах работы, установите курсор на команду, с которой хотели бы продолжить отладку и нажмите *Ctrl+F10*.

На левой границе окна против команды, которая будет выполняться, устанавливается стрелка — указатель выполнения команд.

Первая команда:

```
in    tmp, MCUCR
```

Для просмотра содержимого MCUCR выполнить: View | New IO view, в открывшемся окне IO выполнить: CPU | Control register (щелкнуть по знаку «+» слева в строке CPU, затем щелкнуть по знаку «+» слева в открывшейся строке Control register).

В строке Control register индицируется состояние этого регистра 0x00 и адрес этого регистра в оперативной памяти микроконтроллера 0x35 (это одна из форм записи шестнадцатеричного числа, соответствующая записи \$35). Ниже можно увидеть состояние интересующих нас битов SRE и SRW, а при необходимости установить состояние какого-либо бита прямо здесь, например, бита SM (sleep mode), состояние которого не влияет на работу этой программы. Для этого установите галочку (щелкните) в квадратике, помеченном цифрой 4, находящемся правее слова SM. Заметьте, состояние регистра (строка Control register) изменилось на 0x10.

Для проверки состояния регистра tmp выполним:

- Watch | Add watch;
- в строке ввода появившегося окна Watch введите tmp.

Дважды щелкнув по величине, хранимой в регистре (0x00), можно ввести с клавиатуры новое значение. Щелкнув правой клавишей мышки в окне Watch на имени переменной tmp, можно изменить формат вывода состояния регистра на десятичный, добавить новую переменную для просмотра или удалить выделенную переменную.

Теперь, когда отладка началась, можно изменить опции симулятора: Options | Simulator options. После их изменения программу надо вновь ассемблировать (F7) и перейти к пошаговой отладке (F11).

Выполним команду, нажав кнопку F11 на клавиатуре. Состояние переменной tmp (окно Watches) стало равным состоянию регистра MCUCR (0x10). Если окно Watches развернуть, то можно увидеть тип переменной, а также в каком регистре она хранится.

Следующий шаг (нажать F11 — далее просто нажать F11) — в переменной tmp величина 0xD0 или 1101 0000 в двоичном представлении, два старших бита установлены, остальные биты остались без изменения.

Нажать F11 — теперь в MCUCR установлены биты SRE и SRW, работа с внешней памятью разрешена.

Добавим две переменные ZH и ZL в окно переменных (Выполнить Watches | Add (или в открытом ранее окне Watches щелкнуть правой клавишей мышки, в появившемся окне выбрать Add watch), ввести имя ZH, повторить процедуру, введя имя ZL).

Откроем окно просмотра состояния процессора: View | Processor.

Дважды нажать F11, наблюдая за изменением переменных ZH и ZL в окне Watches. В окне Processor наблюдайте состояние Z-register.

Замечание. В окне Watches удобно группировать переменные в соответствии с размещением их в программе. Для одной части программы переменные можно разместить в группе, выбрав страницу окна Watch1 (смотрите нижнюю строку окна Watches), для другой части программы разместите переменные на странице Watch2 и т. д.

Если вы вводите группу переменных, после набора имени первой переменной дважды нажмите клавишу ENTER — теперь можно вводить имя следующей переменной, после набора имени последней переменной нажмите ENTER один раз.

Добавить переменную cnt в окно Watches, нажать F11.

Ввод имени константы в окно Watches в AVR Studio 3.51 может привести к неустранимой ошибке (например, константа aArBgn из нашей программы). То же может произойти, если Вы будете удалять переменные, пользуясь клавишей Del. Это внутренние ошибки AVR Studio 3.51.

Следующая часть программы — это цикл записи элементов массива в память. Откроем окно просмотра содержимого памяти (выполнить View | New memory view). В появившемся окне Memory будем просматривать оперативную память (выбрать режим Data в первой строке окна слева). В первой строке окна справа установлен адрес ячейки оперативной памяти 0x0060, расположенной в первой строке области просмотра памяти. Наша программа должна записывать массив, начиная с адреса 0x0170, для того, чтобы в окне Memory было удобно наблюдать занесение данных в память, переместим просматриваемую область памяти так, чтобы область просмотра памяти начиналась адресом 0x0170. Для этого в адресе 0x0060 в первой строке окна справа заменим 0060 на 0170, либо воспользуемся вертикальным скроллером окна Memory.

Теперь на экране находятся четыре окна, которые нам надо контролировать. Чтобы видеть все окна, надо выбрать их размеры и место размещения. Место размещения можно изменять, для чего, установив указатель мышки на наименование окна (например, *Watches*), нажать левую клавишу мышки и, не отпуская нажатую клавишу, перемещать мышку, передвигая окно по экрану к нужной позиции. Переместив окно, клавишу мышки отпустить.

Выбор размера окна, например, для окна *Watches*, начнем с уменьшения ширины колонок. Установите указатель мышки в первой строке окна на вертикальной разграничительной линии между наименованиями колонок *Watch* и *Value*. При попадании на разделительную линию форма курсора изменится, приобретя вид жирной вертикальной линии с горизонтальными стрелками, направленными вправо и влево, как на рис. 6.24. Нажмите левую клавишу мышки и, не отпуская нажатую клавишу, перемещайте вместе с мышкой вертикальную разделительную линию влево, пока колонка не сузится до минимально необходимой ширины. На рисунке третья колонка, не представляющая интереса в данной программе, максимально сужена. Установите указатель мышки на любую стенку или любой угол окна (при этом должны появиться стрелки, направленные в противоположные стороны), манипулируя мышкой так же, как при перемещении разделительной линии, измените размер окна. Если окно развернуто полностью, изменить его размеры можно после нажатия кнопки «Восстановить» (в строке наименования окна средняя кнопка справа, помечена квадратами).

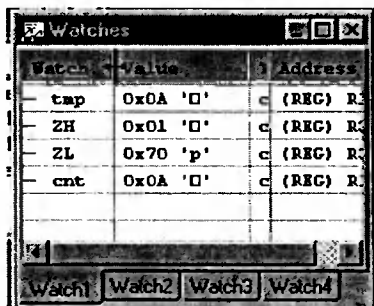


Рис. 6.24. Вид окна

На рис. 6.25 представлены окна, использующиеся при отладке программы SRAM.asm. Размеры и размещение окон позволяют одновременно наблюдать происходящие в них изменения.

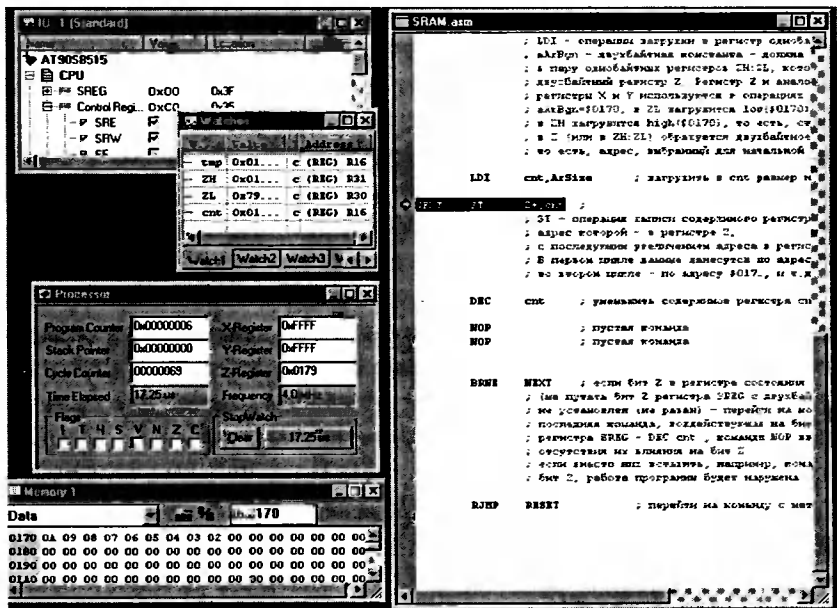


Рис. 6.25. Вид окна

Все до сих пор выполнявшиеся команды занимали по одному циклу микроконтроллера, мы выполнили 6 команд, что соответствует состоянию счетчика циклов (Cycle counter 00000006) в окне Processor.

Нажмите F11. В ячейку памяти 0x0170 записалось число \$0A (окно Memory). Состояние счетчика циклов увеличилось лишь на два (окно Processor). Для команды ST такое увеличение соответствует обращению лишь к внутренней памяти микроконтроллера. В нашем случае обращения к внешней памяти количество циклов должно было увеличиться на четыре, так как один дополнительный цикл требуется для обращения к внешней памяти и еще один цикл, так как мы задали режим работы с дополнительным циклом ожидания установкой бита SRW в регистре MCUCR.

При обращении к внешней памяти количество циклов, подсчитанное симулятором, оказывается ошибочным.

Нажмите F11 четыре раза, наблюдая за состоянием счетчика циклов. При выполнении условия команды BRNE (если бит Z в окне Processor | Flags сброшен) состояние счетчика циклов увеличивается на два. Обратите на это внимание, так как в описании для этой команды указано число циклов 1/2 (один или два). Один цикл потребуется при невыполнении условия команды (смотрите ниже).

Теперь мы перешли ко второму циклу записи массива в память. Ничего нового в каждом отдельном цикле мы не увидим, поэтому воспользуемся сочетанием клавиш Ctrl+F10 (нажать клавишу Ctrl, не отпуская ее, нажать F10).

По каждому нажатию Ctrl+F10 контролируйте переменную cnt в окне Watches. Число нажатий — пока переменная cnt не станет равной единице. При каждом нажатии можно также следить за записью данных в память (окно Memory), адресом, по которому производится запись (окно Processor | Z-register и переменные ZH, ZL в окне Watches).

Если в окне Processor | Stop watch нажать кнопку Clear, то после следующего нажатия Ctrl+F10 правее кнопки Clear появится информация о длительности одного цикла.

Теперь, когда мы снова остановились на команде ST, а cnt=1, воспользуемся клавишей F11.

Дважды нажать F11. Выполнение команды DEC привело к нулевому результату в регистре cnt, поэтому бит Z установлен (окно Processor | Flags).

Дважды нажать F11. Состояние бита Z не изменилось, так как команды NOP не влияют на него.

Нажать F11. Так как бит Z установлен, выполнение команды BRNE не приводит к переходу на метку NEXT, выполняется переход к следующей команде. Состояние счетчика циклов при невыполнении условия команды BRNE (когда бит Z в окне Processor | Flags установлен) увеличивается на единицу.

*Замечание.* Команду BRNE лучше ставить непосредственно за командой, определяющей, как работать команде BRNE (в нашем случае правильное было бы разместить перед ней команду DEC cnt).

Отладка блока считывания данных из памяти не отличается от отладки блока записи в память. Вы можете самостоятельно отладить эту часть программы.

Замечу лишь, что блок будет отлаживаться нормально, так как в память записывалось состояние счетчика циклов `cnt`. Попробуйте перед началом отладки блока считывания из памяти занесенное в ячейку `0x0170` значение `0A` заменить вручную на значение `01` и отладить программу. Выполнится лишь один цикл считывания, так как переменные `cnt` и `tmp` хранятся в одном регистре `r16`.

До начала цикла в регистр `r16` командой `LDIcnt,ArSize` записывается начальное состояние счетчика циклов (`10` или `$0A`). Затем в тот же регистр `r16`, как в переменную `tmp`, из памяти считывается значение `01` (команда `LDtmp,Z+`), далее содержимое регистра `r16`, как переменной `cnt`, декрементируется (`DECcnt`), становясь равным нулю и приводя к завершению цикла.

При присвоении одному регистру нескольких имен переменных, исключайте использование этих имен в одних и тех же блоках программы.

Завершая работу, закрываем окно AVR Studio, в ответ на запрос «Сохранить изменения в проекте `Memory.apr`?» нажимаем кнопку Да.

Теперь в директории `C:\Avr\Try\` находится файл проекта `Memory.apr`, файл программы `SRAM.asm`, включенный в проект, файл `8515def.inc`, а также файл `SRAM.hex`, который может быть загружен с помощью программатора и соответствующего программного обеспечения во флеш-память микроконтроллера.

При следующем запуске AVR Studio наш проект можно вызвать из меню `File`. Здесь можно выбрать один из файлов, с которыми мы уже работали. Для нашего проекта следует выбрать строку `C:\Avr\Try\Memory.apr`. Другой вариант — выполнить `Project | Open` и в появившемся каталоге выбрать необходимый файл проекта (`Memory.apr`). В открывшемся окне `Project` дважды щелкните на имени файла программы `SRAM.asm`. Теперь все готово к новому сеансу работы с нашим проектом.

Подключение буферов вывода с использованием адресного пространства внешней памяти микроконтроллера `AT90S8515`.

В предыдущем примере рассматривалось подключение внешней памяти к AT90S8515. Из четырех портов микроконтроллера порты A и C, а также два контакта порта D заняты обслуживанием внешней памяти.

Как быть, если в дополнение к подключению внешней памяти необходимо организовать передачу полутора — двух десятков управляющих сигналов от микроконтроллера к внешним устройствам? Возможный вариант — использование необходимого количества 8-разрядных регистров (буферов) с обращением к ним, как к ячейкам памяти, расположенным в общем адресном пространстве внешней памяти. На рис. 6.26 — фрагмент схемы, реализующий такое подключение. Данный фрагмент и фрагмент, изображенный на рис. 6.20, являются частями одной схемы, а шина BUS1 является общей для обоих рисунков.

Три адресные линии A13...A15 остались свободными при подключении внешней памяти объемом 8Kx8, используем их для записи данных в три 8-разрядных регистра DD6...DD8.

Использование дополнительной микросхемы дешифратора адреса A13...A15 позволило бы подключить до восьми (23) регистров.

Входные контакты D0...D7 регистров DD6...DD8 подключены к порту A микроконтроллера (линии A/D0...A/D7). К выходным контактам Q0...Q7 регистров можно подключить до 24-х линий внешних устройств (линии B0...B23). Выходы регистров всегда активны и не переводятся в третье состояние, поэтому контакты EO регистров соединены с общим проводом.

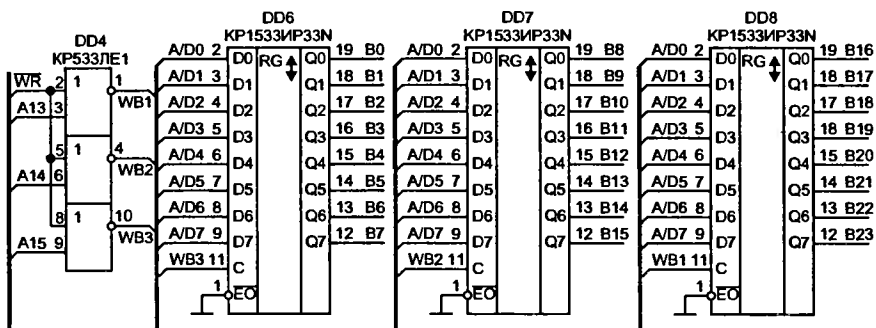


Рис. 6.26. Электрическая принципиальная схема

Запись данных в регистры происходит при появлении высокого уровня на их контактах С. Микроконтроллер же формирует низкий уровень записи во внешнюю память (сигнал WR). В схеме для формирования сигнала записи высокого уровня применены элементы ИЛИ-НЕ (микросхема DD4).

В этой схеме сигнал высокого уровня на линии WB3 для записи данных в регистр DD6 образуется при низких уровнях сигналов на линиях A15 и WR. То есть, чтобы запись произошла в один регистр DD6, необходимо, чтобы на линиях WR и A15 был низкий уровень, на линиях A13 и A14 — высокие уровни.

Одновременное присутствие низких уровней на всех четырех линиях (WR, A13...A15) вызовет параллельную запись данных в три регистра.

Приведенная схема подключения регистров требует тщательного подхода к выбору адресов при написании программы.

Запись в регистр DD6 должна происходить при A15=0, A14=1 и A13=1, то есть, в двоичном коде адрес

регистра DD6: 011x xxxx xxxx xxxx,

регистра DD7: 101x xxxx xxxx xxxx,

регистра DD8: 110x xxxx xxxx xxxx.

Для того, чтобы данные не записывались в регистры, адрес должен иметь вид:

111x xxxx xxxx xxxx,

где x — состояние бита (0 или 1) не имеет значения.

Хотя состояния битов x не имеют значения, они должны быть определены. Это значит, что при записи в какой-либо регистр на контактах A0...A12 микросхемы памяти (рисунок [bvn\\_Pic1.gif](#)) будет сформирован адрес, а в ячейку с этим адресом будет произведена запись того же значения, которое посылается в регистр. Надо позаботиться, чтобы ячейка с таким адресом не использовалась для хранения данных.

Удобно определить все биты x равными единице. Тогда адреса регистров примут вид:

адрес DD6: 0111 1111 1111 1111 или \$7FFF,

адрес DD7: 1011 1111 1111 1111 или \$BFFF,

адрес DD6: 1101 1111 1111 1111 или \$DFFF,

а в ячейке памяти с адресом xxx1 1111 1111 1111 нельзя хранить данные, линии адреса A13...A15, соответствующие битам xxx в этом адресе, к микросхеме памяти не подключены.

Поэтому, какими бы они ни были, запись производится в одну и ту же ячейку. Однако при записи в память xxx следует определить как 111 для того, чтобы не произошло параллельной записи в регистры. Значит, данные нельзя хранить в ячейке 1111 1111 1111 1111 или \$FFFF.

Для хранения данных доступны ячейки памяти с адресами

1110 0000 0000 0000 ... 1111 1111 1111 1110 или \$E000...\$FFFE.

В разделе «Интерфейс микроконтроллера AT90S8515 для подключения внешней памяти» была обозначена нижняя граница доступной для безопасного хранения данных области памяти: адрес \$0060.

С учетом этого ограничения для безопасного хранения данных доступны ячейки памяти с адресами \$E060...\$FFFE.

Есть более привлекательный вариант распределения памяти: при записи в регистр параллельно производить запись в соответствующую этому регистру ячейку памяти. При этом всегда можно проверить, что именно записывалось в регистр, считав данные из ячейки памяти.

Выберем для регистров следующие адреса:

адрес DD6: 0111 1111 1111 1101 или \$7FFD,

адрес DD7: 1011 1111 1111 1110 или \$BFFE,

адрес DD6: 1101 1111 1111 1111 или \$DFFF.

Теперь для безопасного хранения данных доступны ячейки памяти с адресами \$E060...\$FFFC, а из ячеек с адресами \$FFFD, \$FFFE и \$FFFF можно считать данные, которые были записаны в регистры DD8, DD7 и DD6 соответственно.

Считывая данные из ячеек, соответствующих регистрам, мы получаем информацию о том, что записывалось в эти регистры. Следовательно, считывание данных правомерно только после первой записи в регистры. Поэтому рекомендую в начале программы записать исходные данные в регистры.

Состояние выходов регистров может не соответствовать считаным из соответствующих ячеек памяти данным

- из-за неисправности регистров;
- из-за возможного соединения выхода регистра с общим проводом во внешнем устройстве.

Создадим новую директорию C:\Avr\Buff\ . В AVR Studio создадим новый проект, назовем его Buffers, воспользуемся уже созданной директорией C:\Avr\Buff\ , в проекте создадим новый файл BUF\_SRAM.asm, в котором напомним следующую программу.

```
;          Программа BUF_SRAM.asm
;          обращение к внешней памяти, запись данных в буферы
;=====

.INCLUDE    "C:\Avr\Try\8515def.inc"
; Вставка содержимого файла 8515def.inc, находящегося
; в директории C:\Avr\Try в нашу программу

.DEF  tmp   =r16 ; Регистру r16 присвоить имя tmp
;          ; (значение переменной tmp будет храниться в
;          ; регистре общего назначения r16)

.DEF  cnt   =r17 ; Переменная cnt - счетчик цикла

RESET:      IN      tmp,MCUCR
;          ; Ввод содержимого регистра MCUCR в регистр tmp

      ORI      tmp,(1<<SRE)
;          ; Установка бита SRE (режим работы с
;          ; внешней памятью)
;          ; Бит SRW не устанавливаем
;          ; (работа без дополнительного
;          ; цикла ожидания)

      OUT      MCUCR,tmp
;          ; Вывод содержимого регистра tmp в регистр MCUCR

;=====ЗАГРУЗКА ДАННЫХ В БУФЕРЫ

.EQU  wrBuf1    = $7FFD
;          ; Адрес для записи данных в буфер 1
;          ; (по схеме - элемент DD6)

.EQU  wrBuf2    = $8FFE
;          ; Адрес для записи данных в буфер 2 (DD7)

.EQU  wrBuf3    = $DFFF
;          ; Адрес для записи данных в буфер 3 (DD8)

.EQU  rdBuf1    = $FFFD
;          ; Адрес для считывания данных из ячейки, соотв. буферу 1

.EQU  rdBuf2    = $FFFE
;          ; Адрес для считывания данных из ячейки, соотв. буферу 2

.EQU  rdBuf3    = $FFFF
```

; Адрес для считывания данных из ячейки, соотв. буферу 3

LDI tmp,\$AA ; Загрузка константы в tmp

STS wrBuf1, tmp

; Сохранение содержимого регистра tmp в буфере 1

LDI tmp,\$55 ; Загрузка константы в tmp

STS wrBuf2, tmp

LDI tmp,\$71 ; Загрузка константы в tmp

STS wrBuf3, tmp

;=====ЧТЕНИЕ / ИЗМЕНЕНИЕ ДАННЫХ В БУФЕРАХ

LDS tmp, rdBuf1

; Передача данных из ячейки с адресом rdBuf1 в регистр tmp

ORI tmp, (1<<7)+(1<<6)+(1<<5)+(1<<4) ; Установка битов 4...7

STS wrBuf1, tmp

; Сохранение содержимого регистра tmp в буфере 1

LDS tmp, rdBuf2

ORI tmp, 0b11110000

; Установка тех же битов, что и для буфера 1,

; только константа представлена в двоичной форме,

; для обозначения двоичной константы 1111 0000

; перед ней ставят символы "ноль" и "b"

STS wrBuf2, tmp

; Сохранение содержимого регистра tmp в буфере 2

LDS tmp, rdBuf3

ANDI tmp, 0b11110000 ; сброс тех же битов

STS wrBuf3, tmp

; Сохранение содержимого регистра tmp в буфере 2

;=====ЗАГРУЗКА В ПАМЯТЬ ПЕРВОГО МАССИВА

.EQU ArSize =10 ; Размеры массивов

.EQU aArBgn1 = \$E060

; Используем константу aArBgn как адрес начальной ячейки

; для хранения массива 1

LDI ZL,low(aArBgn1)

LDI ZH,high(aArBgn1)

; Загрузка в регистр Z адреса начала массива 1

LDI cnt,ArSize ; Загрузка в cnt размера массива

LDI tmp,\$FF ; Загрузка константы в tmp

ARR1: ST Z+, tmp ;

; Запись содержимого регистра tmp в ячейку памяти,

; адрес которой - в регистре Z,

; с последующим увеличением на 1 адреса в регистре Z

; В первом цикле содержимое tmp запишется

; по адресу aArBgn1 (\$E060), во втором цикле -

; по адресу aArBgn1+1 (\$E061) и т. д.

INC tmp ; Увеличить содержимое tmp на единицу

```
DEC    cnt    ; Уменьшить содержимое счетчика циклов на единицу
```

```
BRNE   ARR1
```

; Если бит (флаг) Z в регистре состояния процессора SREG

; не установлен – перейти на команду с меткой ARR1:

=====ЗАГРУЗКА В ПАМЯТЬ ВТОРОГО МАССИВА

```
.EQU   aArBgn2    = $FFFC
```

; Используем константу aArBgn2 как адрес последней ячейки

; для хранения массива 2

```
LDI    ZL,low(aArBgn2+1)
```

```
LDI    ZH,high(aArBgn2+1)
```

; Загрузка в двухбайтный регистр Z адреса конечного

; элемента массива 2, увеличенного на единицу

```
LDI    cot,ArSize ; Загрузка в cnt размера массива
```

```
LDI    tmp,$03    ; Загрузка константы в tmp
```

```
ARR2: ST    -Z, tmp    ;
```

; Сначала уменьшается на единицу адрес,

; хранящийся в регистре Z (поэтому в Z загружался

; адрес, увеличенный на единицу), затем по новому

; адресу запишется содержимое регистра tmp

; В первом цикле содержимое tmp запишется

; по адресу (aArBgn2+1)-1, то есть, по адресу aArBgn2

; во втором цикле – по адресу aArBgn2-1, и т. д.

```
DEC    tmp    ; Уменьшить содержимое tmp на единицу
```

```
DEC    cnt    ; Уменьшить содержимое счетчика циклов на единицу
```

```
BRNE   ARR2
```

; Если бит (флаг) Z в регистре состояния процессора SREG

; не установлен – перейти на команду с меткой ARR2:

```
STOP: RJMP STOP    ; Заикливание программы
```

; (перейти на команду с меткой STOP:)

Ассемблируем программу (клавиша F7). При обнаружении ошибок проверьте правильность набора программы в строках, содержащих ошибки — номера строк с ошибками и комментарии к этим ошибкам выводятся в окне Project output, появляющемся после ассемблирования.

Директивой `.INCLUDE` в программу вставлен файл `8515def.inc`, использовавшийся в предыдущем примере и находящийся с проектом в разных директориях. Поэтому ассемблирование пройдет нормально, если файл `c:\avr\try\8515def.inc` еще существует.

Начнем отладку программы, нажав клавишу F11. В появившемся окне Simulator options выберем микроконтроллер: Device | AT90S8515 with External SRAM.

Для контроля отладки вызовем окна переменных, процессора и памяти: View | Watches, View | Processor и View | New Memory view.

В окно Watches введем переменные tmp и cnt.

Первые три команды были отлажены в проекте Memory.apr, поэтому переместим курсор на первую команду ЗАГРУЗКИ ДАННЫХ В БУФЕР: LDItmp,\$AA и нажмем Ctrl+F10.

Нажмем F11 — в переменную tmp (смотрите в окне Watch) загрузилась константа \$AA.

В окне просмотра памяти Memory перейдем на адрес \$7FFD (адрес wrBuf1), для этого в окне ввода, расположенном в первой строке окна справа, заменим 0x0060 на 0x7FFD. Жмем F11 — в ячейке с адресом 7FFD появились данные AA.

Таким же способом переходим к ячейке \$BFFE (wrBuf2), дважды жмем F11, проверяем состояние ячейки, оно равно 55.

Повторяем действия для ячейки \$DFFF — ее состояние станет равным 71.

Теперь указатель выполнения команд находится на первой команде ЧТЕНИЯ/ИЗМЕНЕНИЯ ДАННЫХ В БУФЕРАХ:

```
LDS tmp, rdBuf1
```

К сожалению, отладчик не позволит правильно отобразить процессы, происходящие в нашей схеме из-за того, что симулятор предполагает подключение к микроконтроллеру лишь внешней памяти размером 65536 Кбайт, поэтому параллельная запись данных в буфер и в соответствующую ячейку памяти никак не отражается симулятором.

Возможный выход — до выполнения команды LDS вручную загрузить в ячейки с адресами rdBuf1, rdBuf2 и rdBuf3 данные, которые должны были в них загрузиться при записи в буферы 1...3 (\$AA, \$55 и \$71 соответственно).

Итак, в окне Memory перейдем к ячейке с адресом \$FFFD (rdBuf1). В этом случае удобнее воспользоваться скроллером, переместив его в самый низ, так как интересующие нас ячейки с адресами \$FFFD, \$FFFE, \$FFFF (rdBuf1, rdBuf2 и rdBuf3 соответственно) явля-

ются тремя последними ячейками, которые могут отображаться в окне Memory.

Щелкнем на третьей от конца ячейке — ее адрес в виде 0xFFFD отобразится в окошке адреса (справа в первой строке окна Memory). Вместо содержимого этой ячейки (скорее всего 00) введем AA, курсор автоматически перейдет на следующую ячейку с адресом 0xFFFE, введем в нее значение 55, в ячейку 0xFFFF — значение 71.

Второй вариант (здесь не реализован, проверьте его отладку самостоятельно): после команд записи типа

```
STS    wrBuf1, tmp
```

добавлять команды

```
STS    rdBuf1, tmp
```

то же для буферов 2 и 3. Что при этом произойдет?

*В реальной схеме:*

- при выполнении первой из команд запись произойдет как в буфер, так и в микросхему памяти (подробности смотрите в описании схемы);
- вторая команда еще раз запишет те же данные в ту же ячейку микросхемы памяти.

*При отладке:* одинаковые данные запишутся в два адреса: один — имитирующий ячейку микросхемы памяти, второй — имитирующий буфер.

При этом программа несколько удлинится. Однако после отладки дополнительные команды можно удалить.

Продолжим отладку. Следя за ячейками памяти wrBuf1, wrBuf1 и wrBuf3, а также за переменной tmp, выполним 9 команд (9 нажатий F11).

Теперь указатель выполнения команд находится на первой команде ЗАГРУЗКИ В ПАМЯТЬ ПЕРВОГО МАССИВА.

Интерес в записи первого массива в память представляет первый цикл.

В окне Memory перейдем к ячейке с адресом 0xE060 (aArBgn1). Переведем курсор на команду INCtmp и нажмем Ctrl+F10. В ячейке с

адресом 0xE060 появился байт данных FF, соответствующий значению переменной tmp.

Нажмем F11 — выполнение команды INC привело к тому, что переменная tmp стала равна нулю, в результате чего установился бит Z (окно Processor). Поэтому если бы команда INC стояла перед командой BRNE, выполнялся бы лишь один цикл записи элементов массива в память.

Замечание:  $\$FF+1 = \$100$ , результат — двухбайтная величина, но поскольку регистр может хранить только один байт, в нем остается младшая часть, то есть, ноль. Аналогичным образом  $\$00-1 = \$FF$ , при этом вычитание происходит как бы из двухбайтного числа \$100.

Жмем F11 — выполнение команды DEC уменьшило до 9 значение переменной cnt, то есть, результат не нулевой, поэтому флаг Z сбросился (окно Processor).

Отладка аналогичного цикла уже комментировалась, поэтому можно установить курсор на команде с меткой ARR2: и нажать Ctrl+F10. Заметьте, в регистр Z (окно Processor) записался адрес 0xFFFFD, соответствующий константе aArBgn2+1.

В окне Memoгу перейдем к ячейке 0xFFFFD, жмем F11: запись содержимого регистра tmp произошла в ячейку с адресом 0xFFFFC (aArBgn2), как и было необходимо.

Отладка цикла не требует комментариев.

Установим курсор на последней команде программы и нажмем Ctrl+F10. Проконтролируйте содержимое ячеек памяти, в которые записался массив 2.

Что будет, если убрать последнюю команду? Вообще-то для проверки удобнее заменить ее командой NOP. Ассемблируем программу, нажмем F11, установим курсор на последней команде NOP. В окне Processor | Cycle counter заметим число циклов процессора (153). Нажмем F11: выполнение программы опять началось с первой команды, но посмотрите на число циклов процессора: их стало 4206! После того, как все команды нашей программы были извлечены из памяти программ и выполнены, продолжался поиск команд в свободной части памяти программ, при переходе к новой свободной ячейке счетчик циклов процессора увеличивал свое состояние, а вместе с ним и счетчик команд (Processor | Program counter). Максимальное значение, которое может храниться в этом счетчике для микроконт-

роллера AT90S8515, равно 4095, следующее значение — снова ноль, поэтому и произошел переход в начало программы, а к содержимому счетчика циклов процессора добавились циклы загрузки отсутствующих команд.

Обычно микроконтроллеры работают в бесконечном цикле. Если программа завершила свое выполнение и ожидает прерывания, установите в конце команду зацикливания на себя:

Stop:rjmpStop

Автор: Баранов Вадим Николаевич (E-mail: bvn123@bk.ru).

### ***Подключение внешней памяти 512 Кбайт к микроконтроллеру AT90S8535***

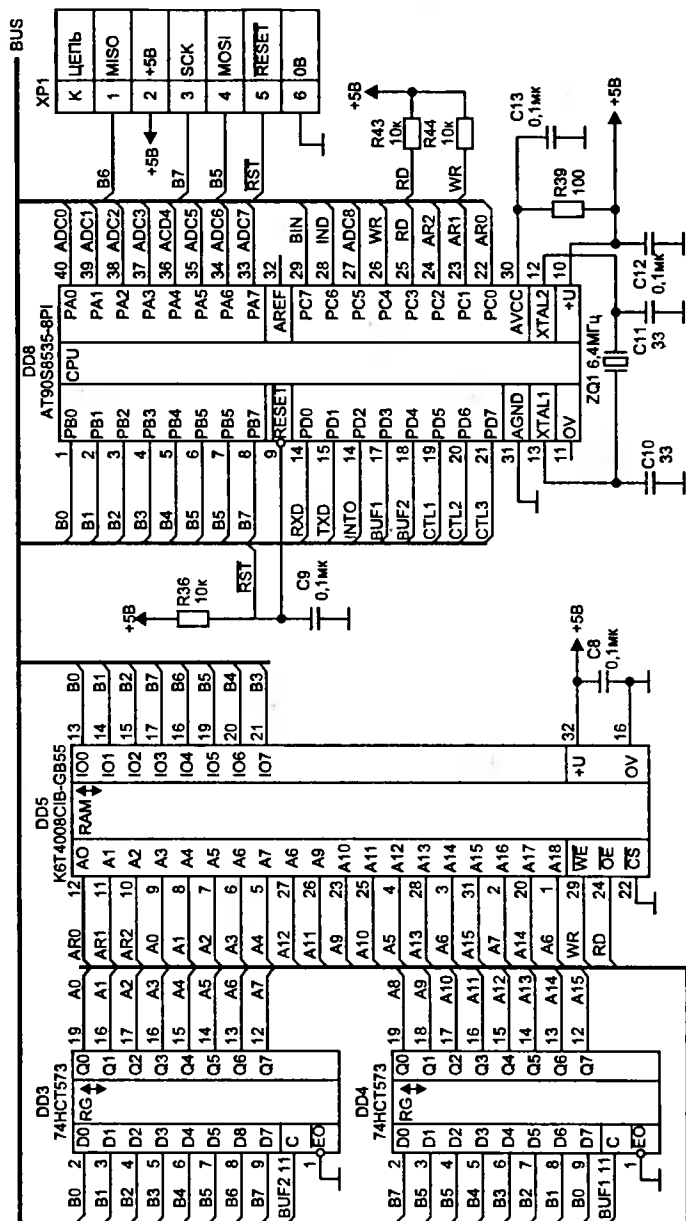
На момент написания книги лишь два микроконтроллера серии AT90 были снабжены интерфейсом подключения внешней памяти: AT90S4414 и AT90S8515. К тому же первый из них уже снят с производства.

В этой же серии есть микроконтроллеры со встроенным аналого-цифровым преобразователем (АЦП). Часто встречающаяся задача, возлагаемая на такие микроконтроллеры, — преобразование аналоговых сигналов в код с записью кода в память в реальном времени. Однако внутренней памяти микроконтроллеров для этого обычно не достаточно, интерфейса для подключения внешней памяти они не имеют.

Здесь представлен возможный вариант подключения внешней памяти большого объема к микросхеме AT90S8535. Микроконтроллер снабжен встроенным 10-разрядным АЦП с восемью входами, коммутируемыми программно. В качестве входов АЦП в микроконтроллере используются контакты порта А, поэтому не будем задействовать этот порт для организации связи с внешней памятью.

На рис. 6.27 представлен фрагмент схемы, реализующий подключение микросхемы статической памяти K6T4008CIB-GB55 (DD5) к микроконтроллеру AT90S8535 (DD6).

Микросхема памяти K6T4008CIB-GB55 производства фирмы Samsung Electronics по своим функциям не отличается от аналогичной микросхемы HM62256, использованной в схеме подключения внешней памяти к микроконтроллеру AT90S8515 (смотрите преды-



**Рис. 6.27. Подключение микросхемы статической памяти K6T4008CIB-GB55**

дущий пример). Процедура записи и считывания данных для обеих микросхем одинакова.

Одинаково функционируют их двунаправленные 8-разрядные шины данных (контакты IO0...IO7), а также линии

- записи (контакт WR);
- управления шиной данных (контакт OE);
- выбора микросхемы (контакт CS).

Отличие состоит в объеме памяти: 19-разрядная шина адреса (контакты A0...A18) микросхемы K6T4008CIB-GB55 обеспечивает обращение к 219 ячейкам памяти, это значит, что в микросхеме можно хранить до 512 Кбайт данных (512K × 8).

#### Описание схемы

В таблице описаны линии, по которым микроконтроллер производит управление микросхемой памяти.

Линия	Контакт микроконтроллера	С чем соединяет/конт.	Назначение
WR	DD6 / PC4	DD5 / WE	Запись данных в DD5 при низком уровне
RD	DD6 / PC3	DD5 / OE	Считывание данных из DD5 при низком уровне
BUF1	DD6 / PD3	DD4 / C	Запись старшего байта адреса в регистр DD4 при высоком уровне
BUF2	DD6 / PD4	DD3 / C	Запись младшего байта адреса в регистр DD3 при высоком уровне
AR0...AR2	DD6 / PC0...PC2	DD5 / A0...A2	Адрес страницы

В исходном состоянии микроконтроллер устанавливает высокие уровни на линиях WR и RD, низкие уровни на линиях BUF1 и BUF2.

Программно память как бы разбита на 8 страниц по 64 Кбайт в каждой. Три бита адреса из 19-ти микроконтроллер формирует на линиях AR0...AR2, выбирая одну из страниц памяти.

Адрес внутри страницы формируется двумя 8-разрядными регистрами 74HCT573 (DD3 и DD4).

С этой целью на шину данных (линии B0...B7) через порт В микроконтроллера (контакты PB0...PB7) выводится младший байт адреса, после чего микроконтроллер устанавливает высокий уровень на

линии BUF2 для записи младшего байта адреса в регистр DD3. Затем на линии BUF2 устанавливается низкий уровень.

Далее на шину данных через порт В выводится старший байт адреса, запись которого в регистр DD4 происходит по установке микроконтроллером высокого уровня на линии BUF1. После записи на линии BUF1 устанавливается низкий уровень.

Теперь на линиях A0...A18, следовательно, и на контактах A0...A18 микросхемы памяти адрес ячейки памяти установлен, можно производить запись или считывание.

Некоторое смущение может вызвать то, что к контактам микросхемы памяти A0...A19 подсоединены линии адреса с другими именами (к контакту A17, например, линия A14). Это сделано для удобства трассировки печатной платы, на которой размещены детали схемы. Для микросхемы памяти безразлично, в какой последовательности выбираются ячейки, последовательность же выбора при записи и при считывании будет одинаковой.

Если вы занимаетесь трассировкой, то представляете, как были перепутаны дорожки до оптимизации соединений линий адреса с контактами адреса микросхемы памяти.

### ***Запись в ячейку***

После формирования адреса микроконтроллер выводит через порт В на шину данных B0...B7 байт информации. Затем микроконтроллер устанавливает низкий уровень на линии WR, что приводит к записи байта информации в ячейку памяти. После записи на линии WR устанавливается высокий уровень.

### ***Считывание из ячейки***

После формирования адреса микроконтроллер переводит порт В в режим ввода данных в микроконтроллер. Затем на линии RD микроконтроллер устанавливает низкий уровень, что приводит к выводу данных из адресуемой ячейки памяти на шину данных B0...B7. Микроконтроллер через порт В считывает байт данных, установленный на линиях B0...B7, после чего устанавливает высокий уровень на линии RD. Обратите внимание на резисторы R43 и R44.

При программировании линии портов микроконтроллера находятся в высокоимпедансном состоянии, а поскольку входы WE и OE

микросхемы памяти также имеют высокий импеданс, наводки на подключенных к ним линиях могут привести как к записи в произвольную ячейку памяти, так и к выводу данных из произвольной ячейки на линии B0...B7. Последняя ситуация опасна, так как запись программы в микроконтроллер производится с использованием линий B5...B7 (разъем для подключения программатора XP1 изображен справа на схеме). Поэтому возможны сбои при программировании (наблюдались в отсутствие резистора R43), выход из строя программатора (маловероятно из-за достаточной стойкости используемой в нем микросхемы) или микросхемы памяти (вполне возможно).

Установка резистора R43, подключенного к источнику питания +5 В, создает в описанной ситуации на линии RD высокий уровень и переводит выходы IO0...IO7 микросхемы памяти в высокоимпедансное состояние, исключая подобную ситуацию, резистор R44 устраняет возможность хаотичной записи при программировании. В обычном режиме работы эти резисторы не мешают микроконтроллеру управлять микросхемой памяти.

### **Программа записи данных в ОЗУ 512 Кбайт**

В рабочей программе, взятой за основу для нашего примера, АЦП микроконтроллера производил группу преобразований, результаты которых записывались во внутреннюю оперативную память микроконтроллера. Результаты группы преобразований обрабатывались и также записывались во внутреннюю оперативную память в виде массива размером в 45 байтов, начиная с ячейки с адресом aPasc. Этот массив переносился во внешнюю память для хранения. АЦП выполнял новую группу преобразований, которые тем же способом обрабатывались, а очередной массив размером в 45 байтов добавлялся во внешнюю память. Процесс продолжался до заполнения страницы внешней памяти (64 Кбайт), после чего вся информация со страницы внешней памяти через микроконтроллер передавалась в компьютер.

Представленная программа включает в себя подпрограммы, взятые из рабочей программы, отсюда же заимствовано распределение внутренней оперативной памяти микроконтроллера, а также имена констант и переменных.

Создадим новую директорию `c:\Avr\Ram512`, поместим в нее файл `8535def.inc`, полученный при распаковке файла `avr000.exe` (ищите его там же, где распаковался файл `8515def.inc`). В директории `c:\Avr\Ram512` создадим проект `Ram512` с новым файлом программы `Ram512.asm`, в который следует перенести следующую программу.

```
; Подключение 03У512К к АТ90С8535
;=====
.include "c:\atmel\8535def.inc"
.equ RamH  = $16a ;Адрес ячейки внутренней SRAM, хранящей
; старший байт адреса страницы внешней памяти
.equ RamL  = $16b ;Адрес ячейки внутренней SRAM, хранящей
; младший байт адреса внешней памяти

; Три старших бита адреса внешней памяти
; (номер страницы внешней памяти):
.equ Ar0   = PC0
.equ Ar1   = PC1
.equ Ar2   = PC2

.equ Wr     = PC4 ;Управление записью во внешнюю память
.equ Rd     = PC3 ;Управление считыванием из внешней памяти

.equ Buf1   = PD3 ;Управление записью в буфер 1
.equ Buf2   = PD4 ;Управление записью в буфер 2

.equ aADC   = $60 ; Адрес для переноса массива из внешней
; памяти во внутреннюю SRAM
.equ aPack  = $b0 ; Адрес 45-байтового массива во
; внутренней SRAM микроконтроллера

.def tm     = r16
.def cnt    = r19

;==

RESET:
ldi tm, (1<<Wr)+(1<<Rd); Устанавливаем высокие уровни
out PORTC, tm ; на линиях WR и RD
ldi tm, $ff ; контакты порта C -
out DDRC, tm ; в режиме выходов

clr tm ; На линиях порта D низкие уровни
out PORTD, tm ; (в том числе BUF1, BUF2=0)
ldi tm, $ff ; контакты порта D -
out DDRD, tm ; в режиме выходов

ser tm ; Установка регистра tm (tm=$ff)
out DDRB, tm ; вывод в порт DDRB содержимого tm
; контакты порта B - в режиме выходов

ldi tm, low(RAMEND)
;Стек - начиная с конца внутренней SRAM
```

```

out    SPL,tm
ldi    tm,high(RAMEND)
out    SPH,tm

```

```

; Загрузка во внутреннее ОЗУ микроконтроллера,
; начиная с ячейки с адресом aPack
; 45-ти байтов массива: $20, $21, $22,...

```

```

ldi    XL,low(aPack)
ldi    XH,high(aPack)
ldi    tm,$20
ldi    cnt,45

```

```
StRAM:
```

```

st     X+,tm
inc    tm
dec    cnt
brne   StRAM

```

```

; Перепишем этот массив во внешнюю память на страницу №2

```

```

; (старшие биты адреса внешней памяти AR2 AR1 AR0 = 101)

```

```

in     tm,PORTC
andi   tm,$ff-((1<<AR2)+(1<<AR1)+(1<<AR0))
                                ; очистка битов AR2, AR1, AR0
                                ; 1111 1000 = $ff-@K0D = ((1<<AR2)+(1<<AR1)+(1<<AR0))

```

```

ori     tm,(1<<AR1)
                                ; Установка бита AR1(адрес страницы 010
                                ; (страница №2))

```

```

out     PORTC,tm    ; Вывод в порт C содержимого tm
clr     tm           ; Очистка содержимого tm (tm=0)
sts     RamH,tm      ; Сохранение в ячейке RamH содержимого tm
sts     RamL,tm
rcall   St45bt       ; Вызов подпрограммы St45bt

```

```

; Скопируем массив из внешней памяти на странице №2

```

```

; (старшие биты адреса внешней памяти AR2 AR1 AR0 = 101)

```

```

; во внутреннее ОЗУ микроконтроллера

```

```

clr     tm           ; Очистка содержимого tm (tm=0)
sts     RamH,tm      ; Сохранение в ячейке RamH содержимого tm
sts     RamL,tm
rcall   DOutPrp

```

```
cycle:rjmp cycle
```

```

; Подпрограммы:
;=====

```

```

; Копирование массива из внутренней во внешнюю память

```

```
St45bt:
```

```

ldi    ZL,low(aPack)
ldi    ZH,high(aPack)
ldi    cnt,45

```

```
mSt45:
```

```

rcall   SetAddr
rcall   DataSt

```

```

dec    cnt
brne   mSt45
ret

```

```

;===

```

```

; Подпрограмма копирования 45 байтов из внешней памяти

```

```

; во внутреннее ОЗУ

```

```

D0utPrp:

```

```

ldi    ZL,low(aADC)
ldi     ZH,high(aADC)
ldi     cnt,45

```

```

D0ut1:

```

```

rcall   SetAddr
rcall   DataLd
dec     cnt
brne    D0ut1
ret

```

```

;==

```

```

; Подпрограмма установки адреса

```

```

SetAddr:

```

```

lds     XL,RamL      ; Скопировать содержимое ячейки RamL в XL
lds     XH,RamH
out     PORTB,XL     ; Вывести в порт B содержимое XL
nop     ; Задержка 156 нс
nop     ; Задержка 156 нс
sbi     PORTD,Buf1   ; Установить бит Buf1 порта D
nop
nop
cbi     PORTD,Buf1   ; Сбросить бит Buf1 порта D
nop
nop
out     PORTB,XH     ; Вывести в порт B содержимое XH
nop
nop
sbi     PORTD,Buf2   ; Установить бит Buf2 порта D
nop
nop
cbi     PORTD,Buf2   ; Сбросить бит Buf2 порта D
adiw    XL,1         ; Увеличить содержимое пары XH:XL на 1
sts     RamH,XH      ; Сохранить содержимое XH в ячейке RamH
sts     RamL,XL
ret     ; Возврат из подпрограммы

```

```

;==

```

```

; Подпрограмма копирования байта из внутреннего ОЗУ

```

```

; во внешнюю память

```

```

DataSt:

```

```

ld      tm,Z+
out     PORTB,tm
nop
nop

```

```

cbi   PORTC,Wr
pop
pop
sbi   PORTC,Wr
ret

```

```

;==
; Подпрограмма копирования байта из внешней памяти
; во внутреннее ОЗУ
DataLd:

```

```

clr   tm           ; Очистка tm
out   DDRB,tm      ; Контакты порта B в режиме входов
cbi   PORTC,Rd     ; Сбросить бит Rd порта C
pop
pop
in    tm,PINB      ; Считать данные на контактах порта B в tm
sbi   PORTC,Rd     ; Установить бит Rd порта C
st    Z+,tm        ; Сохранить содержимое tm в ячейке ОЗУ
ser   tm           ; Установить tm
out   DDRB,tm      ; Все контакты порта B в режиме выходов
ret

```

### Отладка программы

После ассемблирования (клавиша F7) при отсутствии ошибок приступим к отладке (клавиша F11), в появившемся окне Simulator options | Device выберем микроконтроллер AT90S8535.

Рассмотрим определение констант в программе. Ячейки внутренней памяти микроконтроллера с адресами RamH и RamL хранят старший и младший байты адреса внешней памяти. То есть, в этих ячейках будет отражаться информация об адресе внешней памяти, которая выводится в буферы адреса (регистры DD3 и DD4 по схеме электрической).

Константа AR0 равна константе PC0; PC0 в свою очередь определена в файле, вставляемом в программу директивой .include.

Остальные константы, не определенные в тексте программы, также определяются в файле 8535def.inc.

Нажмем F11. Выполним View | New IO view, в открывшемся окне выполним PortB | +, здесь же выполним PortC | + и PortD | + для контроля информации, выводимой на контакты портов B, C и D, а следовательно и на шину данных нашей схемы (порт B), и на линии управления внешней памятью (соответствующие контакты портов C и D).

Замечание. Имена констант в программе соответствуют именам линий управления внешней памятью на электрической схеме.

Переносим курсор на команду `ldiXL,low(aPack)` блока загрузки внутренней SRAM массивом в 45 байтов и жмем `Ctrl+F10`.

Произошла инициализация портов B, C и D, а также установлен начальный адрес стека.

Проверяем состояние линий портов в окне IO. Порт B инициализирован для вывода данных из микроконтроллера. Линии PC4 и PC3 порта C установлены (в электрической схеме на соответствующих линиях WR и RD будут высокие уровни). На линиях PD3 и PD4 порта D (соответствуют линиям BUF1, BUF2 в схеме) — низкие уровни. В этом же окне IO выполняем `CPU | +` и проверяем состояние регистров указателя стека SPH и SPL.

Для имитации массива, получаемого в результате одной группы преобразований АЦП, в область памяти, начинающуюся адресом `aPack`, записывается простой массив размером в 45 байтов.

Вызовем окно Memory, выполнив `View | New memory view`, перенесем курсор на команду, следующую за циклом записи массива (команда `intm,PINC`) и нажмем `Ctrl+F10`. Во внутренней памяти микроконтроллера создан массив, первый байт которого (`$22`) находится в ячейке с адресом `aPack (0x00B0)`.

Жмем F11 пять раз, проверяем состояние порта C в окне IO. Линии PC2 PC1 PC0 приобрели состояние 010, таким же будет состояние трех старших линий адреса (AR2 AR1 AR0), определяющих номер рабочей страницы внешней памяти.

Трижды жмем F11, очищаем старший и младший адреса внешней памяти в ячейках `RamH`, `RamL`.

Указатель выполнения команд остановился на команде вызова подпрограммы переноса данных во внешнюю память. В окне Memory перейдем в самый конец внутренней памяти, переместив скроллер окна вниз до конца, выполним `View | Processor`, в открывшемся окне процессора заметим состояние счетчика команд (`0x00001D` — это номер команды `gcall`, которая будет выполняться) и указателя стека (`=0x000025F`).

Нажмем F11. Указатель выполнения команд перескочил на команду с меткой `St45bt:` (окно программы), в счетчике команд — номер команды с этой меткой. Во внутренней памяти по адресу `25F` находится двухбайтный номер команды, на которую надо вернуться после выполнения подпрограммы (`001E` — именно эта команда сле-

дует за командой с номером 001D, и именно адрес 001E загрузится в счетчик команд после выполнения подпрограммы). Указатель стека сместился на две ячейки влево (25D), если внутри нашей подпрограммы будет подпрограмма следующего уровня вложения, в ячейки 25D:25C запишется двухбайтный адрес возврата из этой подпрограммы.

Устанавливаем курсор на команду с меткой `mSt45:`, жмем `Ctrl+F10`. Регистр `Z` содержит адрес начала массива во внутренней оперативной памяти микроконтроллера. Указатель выполнения команд находится в начале цикла побайтного копирования данных из внутренней оперативной памяти микроконтроллера во внешнюю память. Отличие цикла от уже рассматривавшихся в этой главе циклов — в наличии двух вызовов подпрограмм. Первый из них: `rcall SetAddr` записывает младший и старший байты адреса внешней памяти в два буфера (по электрической схеме — в регистры `DD3` и `DD4`). Второй вызов подпрограммы `rcall DataSt` извлекает из ячейки внутренней оперативной памяти микроконтроллера (внутреннего ОЗУ) байт данных и записывает его в ячейку внешней памяти.

Наблюдая содержимое счетчика команд и указателя стека в окне `Processor`, а также изменения в окне `Memory` (последняя строка, начинающаяся адресом `0x025F`), нажимаем `F11`, указатель выполнения команд перемещается на первую команду подпрограммы установки адреса внешней памяти.

Подпрограмма установки адреса `SetAddr`:

- из ячеек `RamH`, `RamL` внутреннего ОЗУ в регистры `XH`, `XL` загружаются младший и старший байты адреса внешней памяти (две команды `lds`);
- младший байт адреса выводится на шину данных `B0...B7` (смотрите схему) через порт `B` (команда `out PORTB,XL`);
- после задержки, необходимой для завершения переходных процессов (две команды `por`), на линии `BUF1` устанавливается высокий уровень (команда `sbi PORTD,BUF1`), по которому в регистр `DD4` записывается информация, установленная на шине данных;
- после задержки (две команды `por`) на линии `BUF1` устанавливается низкий уровень (команда `sbi PORTD,BUF1`), запись в регистр завершена;

- старший байт адреса выводится на шину данных B0...B7 через порт B (команда `out PORTB,XH`);
- после задержки на линии BUF2 устанавливается высокий уровень (команда `sbi PORTD,BUF2`), по которому в регистр DD3 записывается информация, установленная на шине данных;
- после задержки (две команды `por`) на линии BUF2 устанавливается низкий уровень (команда `sbi PORTD,BUF2`), завершая запись в регистр DD3;
- команда `adiw` увеличивает содержимое пары регистров XH:XL, теперь в них хранится адрес следующей ячейки внешней памяти, запись в нее будет произведена в следующем цикле установки адреса.

Проведите пошаговую отладку подпрограммы установки адреса, наблюдая за тем, какие данные выводятся в порты. Сопоставьте эти данные с состояниями, которые должны устанавливаться на соответствующих линиях электрической схемы. При выполнении команды выхода из подпрограммы (`ret`) наблюдайте за изменением состояний счетчика команд и указателя стека. Постарайтесь понять, где хранился адрес возврата из подпрограммы, на какой адрес возврата установлен указатель стека теперь.

В регистре ввода-вывода PORTC хранится то, что программа вывела в порт, в регистре ввода—вывода PINC — то, что присутствует на контактах порта (так в PORTC можно вывести байт 11111111, контакты микроконтроллера PC0 и PC2 соединить с общим проводом, тогда состояние PORTC останется без изменений, состояние PINC станет равным 11111010). Для команд типа `out PORTC,tm` в окне IO информация появляется сначала в строке Port C Data (PORTC), а после выполнения следующей команды — в строке Input pins (PINC).

Перед входом в цикл, начинающийся меткой `mSt45`., в регистр Z был занесен адрес начала массива во внутреннем ОЗУ микроконтроллера.

Подпрограмма копирования байта из внутреннего ОЗУ `DataSt`.

- первая команда подпрограммы (`ld tm,Z+`) копирует первый байт массива из ячейки, адрес которой хранится в регистре Z, в регистр `tm`, затем адрес, хранящийся в регистре Z, увеличивает

ется на единицу, теперь Z указывает на следующий элемент массива;

- элемент массива выводится на шину данных через порт B (команда `out PORTB,tm`);
- после задержки (команды `por`) на линии WR устанавливается низкий уровень (команда `cbi PORTC,Wr`), информация, установленная на шине данных, записывается в ячейку внешней памяти, адрес которой был определен при выполнении подпрограммы `SetAddr`;
- после задержки (команды `por`) на линии WR устанавливается высокий уровень (команда `sbi PORTC,Wr`), копирование байта данных во внешнюю память завершено;
- выполняется возврат из подпрограммы.

Выполните один раз пошаговую отладку подпрограммы `DataSt`. Отладка цикла, в котором находится эта подпрограмма, не представляет интереса.

Отследить состояние внешней памяти с занесенным в нее массивом мы не сможем, так как наш вариант подключения внешней памяти к микроконтроллеру не предусмотрен симулятором AVR Studio.

По этой же причине мы не сможем полностью провести отладку копирования массива из внешней памяти во внутреннее ОЗУ микроконтроллера, однако можно воспользоваться приемами, которые предлагались в предшествующем примере:

- отразить массив в области внутреннего ОЗУ специально для отладки;
- или после команды ввода в порт информации с шины данных вводить эти данные в порт вручную.

Воспользуемся вторым приемом при отладке подпрограммы копирования данных из внешней памяти во внутреннее ОЗУ микроконтроллера `DataLD`.

Переместите курсор на команду `rcall DoutPtr` и нажмите `Ctrl+F10`.

Первая пара команд подпрограммы загружает в регистр Z адрес новой области внутреннего ОЗУ микроконтроллера для массива, копируемого из внешней памяти.

Далее выполняется команда загрузки счетчика циклов, затем цикл, начинающийся меткой `Dout1:`.

Цикл не отличается от ранее рассматривавшихся циклов. Подпрограмма SetAddr рассматривалась ранее. Поэтому, доходя до команды вызова этой подпрограммы, нажимайте клавишу F10, чтобы не производить повторную отладку.

В результате выполнения подпрограммы SetAddr на линиях адреса внешней памяти сформирован адрес.

Подпрограмма копирования данных из внешней памяти во внутреннее ОЗУ DataLd.

В порт DDRB выводится нулевое значение, переводящее порт В на прием информации (все контакты порта В работают как входы).

На линии RD устанавливается низкий уровень (команда sbi PORTC,Rd), поэтому из ячейки внешней памяти, адрес которой сформирован при выполнении подпрограммы SetAddr, на шину данных выводится информационный байт.

После задержки (команды por) через порт В с шины данных в регистр микроконтроллера tm считывается байт информации (команда in tm, PINB).

На линии RD устанавливается высокий уровень (команда sbi PORTC,Rd), контакты микросхемы памяти IO0...IO7 переводятся в высокоимпедансное состояние.

Данные из регистра tm переносятся в ячейку внутреннего ОЗУ микроконтроллера, адрес которой хранится в регистре Z, после чего содержимое Z увеличивается на единицу, теперь регистр Z указывает на следующую ячейку ОЗУ, в которую будет произведена запись в следующем цикле (команда stZ+,tm).

Порт В переводится в режим вывода данных (все контакты порта — выходы).

Возврат из подпрограммы.

Переводить порт В из режима передачи данных (выход) в режим приема (вход) только на время выполнения подпрограммы DataLd удобно, так как в других подпрограммах порт В должен всегда находиться в режиме передачи.

При отладке подпрограммы DataLd перед командой in tm,PINB для имитации ввода данных установите какие-нибудь флажки в строке Input pins для окна IO | Port B, тогда соответствующие данные будут занесены в ячейку внутреннего ОЗУ микроконтроллера.

В заключение следует заметить, что приведенные программы адаптированы для лучшего понимания работы устройства, программы и отладчика. Однако считать их завершенными, а файлы hex, полученные при их ассемблировании, загружать в микроконтроллер не следует. В реальной рабочей программе сторожевой таймер (Watch dog timer) должен периодически программно сбрасываться командами `wdr`, размещенными по всей программе, иначе программа будет регулярно сбрасываться этим таймером, возвращаясь на метку `RESET`. В приведенных примерах программ не размещены векторы прерываний, являющиеся неотъемлемой частью практически любой программы. Перечисленные темы в этой главе не рассматриваются.

Автор: Баранов Вадим Николаевич (E-mail: [bvn123@bk.ru](mailto:bvn123@bk.ru)).

## **Глава 7.**

# **Идеи для проектов**

В этой главе описываются интересные идеи конструкций на микроконтроллерах.

Автор сразу хотел бы ответить на вопрос о некоторых из конструкций, описанных здесь: «А зачем это делать на микроконтроллерах?» или: «А кому это устройство нужно?» и т. д. Описанные здесь идеи конструкций предназначены в первую очередь для приобретения практического опыта проектирования и изготовления различных схем на микроконтроллерах. Преимущества — многие устройства достаточно простые, что позволяет их быстро изготовить. При этом, однако, в каждом есть своя изюминка — где-то придется разобраться, как микроконтроллер может определять положение переменного резистора, где-то приобрести полную ясность, что такое дребезг контактов, и т. д. И самое главное, каждая схема — простейшее, но законченное устройство. А сделать что-то работающее и имеющее определенное назначение обычно интереснее, чем просто решить задачу.

Итак, ниже приведены описания конструкций, своего рода домашняя работа для терпеливых читателей, поставивших перед собой цель научиться полноценно использовать микроконтроллеры семейства AVR.

**Самодельный калькулятор** с реализацией вычисления специфических функций для какой-либо специальности (например, расчеты с комплексными числами или расчет значений для записи в регистры микроконтроллеров семейства AVR для формирования заданного временного интервала с помощью таймера и т. д.).

**Счетчик витков** с автоматическим определением направления вращения.

**Электронная записная книжка** с ЖКИ-индикатором.

**Универсальный контроллер** для управления различными устройствами — 8 дискретных входных и 8 дискретных выходных линий.

**Усовершенствованный автомат световых эффектов** — увеличено число управляемых линий, усложнены реализуемые эффекты.

**Прибор для проверки цифровых микросхем** — увеличено число типов проверяемых микросхем.

**Электронные часы-будильник** — введена возможность заводить его на несколько времен (например, на 6, 8 и 10 часов).

**Электронные измерительные приборы** — омметр, амперметр, вольтметр.

**Электронный частотомер с расширенным диапазоном измеряемых частот.**

**Управляющий микрокомпьютер для детской игрушки**, позволяющий задать с помощью специальной клавиатуры с названиями действий и цифрами последовательность действий и их количество, после чего запустить их на выполнение. Пример управляющей последовательности: вперед 4, включить фары, звуковой сигнал 2, погасить фары.

**Цифровой ПИД-регулятор** — ввод параметров регулятора с помощью клавиатуры, они записываются во флэш-память данных и сохраняются при выключении регулятора. Индикация текущего состояния и сообщений при вводе параметров регулятора — с помощью символьного ЖКИ-индикатора 2 строчки по 16 символов.

**Прибор для проверки целостности и маркировки проводов многожильного кабеля.**

**Прибор для снятия характеристик биполярных транзисторов** — характериограф с генерацией видеосигнала снятой характеристики для получения картинки на любом телевизоре.

**Простой логический анализатор.** Отображение состояния 8 линий на графическом ЖКИ-индикаторе 128 × 64 точки. Время измерения зависит от интервала между измерениями.

**Прибор для снятия характеристик стабилитронов с отображением характеристики на графическом ЖКИ.**

**Генератор сигналов произвольной формы.**

**Прибор велосипедиста** — измеритель пройденного пути, спидометр.

**Измеритель частоты оборотов** — тахометр.

**Простейший цифровой осциллограф** с индикацией на графическом ЖКИ индикаторе 128 × 64.

**Генератор испытательных сигналов** для настройки и ремонта телевизоров.

**Кодовый замок.**

**Электронные «крестики-нолики».**

**Электронный тренажер велосипедиста.** Тренировка велосипедистов, особенно в зимнее время, проводится на специальных велосипедных станках, которые часто не имеют каких-либо средств контроля. Это затрудняет работу тренера и не позволяет оценить результаты тренировки. Предлагаемое устройство позволяет при тренировках на любом велосипеде или велосипедном станке определять эквивалентный пройденный путь, фиксировать время прохождения этого пути, а также определять скорость прохождения пути.

Датчиком прибора является фотоэлектрическое устройство, состоящее из инфракрасного светодиода и фотодиода. Его следует установить таким образом, чтобы при вращении колеса велосипеда луч прерывался.

**Измеритель скорости реакции человека.** Реакция человека — действие в ответ на определенный раздражитель (сигнал). В жизнедеятельности человека, быстрота реакции имеет немаловажную роль. Люди с замедленной реакцией не могут быстро, а иногда и безопасно выполнять некоторые работы. Например, водители с замедленной реакцией чаще всего совершают дорожные аварии.

Время реакции человека непостоянно. Оно зависит от состояния нервной системы и многих других факторов. В значительной мере на замедление реакции (увеличение времени на ответные действия) влияют переутомление, употребление алкоголя, отрицательные эмоции и т. д. Прибор позволяет определить время реакции человека на

световой и звуковой сигналы. В приборе имеются лампочки (или светодиоды) трех цветов и звуковой излучатель. Перед началом испытания следует нажать кнопку «Старт».

После ее нажатия через произвольное время в интервале 10—15 с включится звуковой или световой сигнал. Испытуемый должен нажать кнопку, соответствующую поданному сигналу. При этом производится отсчет времени, прошедшего с момента подачи сигнала и нажатия соответствующей кнопки. Измеренное время высвечивается на индикаторе. Для проведения нового испытания следует снова нажать кнопку «Старт».

**Простой одnogолосный электромузыкальный инструмент.** Представляет собой программно реализованный генератор звуковых прямоугольных импульсов. Микроконтроллер проверяет нажатие кнопок клавиатуры и в соответствии с нажатой клавишей формирует звуковой сигнал нужной частоты.

**Игра «Красный или зеленый».** Внешне игра состоит из кнопок «Красный», «Зеленый» и двух светодиодов — красного и зеленого. После того как включена схема, судья дает команду, и каждый из двух игроков старается быстрее нажать свою кнопку. При этом игра реагирует на нажатие только одной кнопки, при этом загорается соответствующий ей светодиод.

**Внимание, реакция и чувство ритма.** Прибор имеет светодиодный индикатор, переменный резистор для установки частоты смены цифр на индикаторе, кнопки «Зажечь/погасить индикатор» и «Фиксация». После включения прибора на индикаторе будут поочередно появляться числа от 0 до 9. Скорость их изменения зависит от положения переменного резистора. Одно нажатие на кнопку «Зажечь/погасить» индикатор включает, а следующее — выключает. Счет при этом продолжается. Работают с прибором вдвоем. Проверяющий включает прибор и вращением переменного резистора устанавливает частоту смены цифр на индикаторе, а затем предлагает испытуемому непрерывно фиксировать нажатием кнопки «Фиксация» какую-нибудь одну цифру. При этом счет останавливается до тех пор, пока он не отпустит эту кнопку. Чтобы усложнить задачу, проверяющий может на время отключать индикатор или менять

частоту смены импульсов переменным резистором. При любом способе контроля реакцию можно оценивать по соотношению количества удачных попыток к их общему числу. Можно делать это вручную, а можно усовершенствовать прибор, добавив в него возможность счета удачных и неудачных ответов и средства индикации результата.

**Модель светофоров на перекрестке.** Это может быть простейшая модель одного светофора, а может быть модель сложного перекрестка. Информацию о его работе несложно получить, просто понаблюдав за сменой сигналов светофора на перекрестке.

**Двухтональный электронный звонок.**

**Цифровой регулятор мощности для нагрузок при напряжении 220 В с гальванической развязкой силовой части и цепей управления.**

**Автоматическое зарядное устройство, проверяющее ток зарядки, напряжение на аккумуляторах, длительность зарядки и температуру аккумуляторов.**

**Простой метроном с регулированием скорости подачи звуковых сигналов двумя кнопками — «Быстрее» и «Медленнее».**

**Камертон музыканта — звуковой генератор, воспроизводящий ноту «ля» — звуковой сигнал с частотой ровно 440 Гц.**

Сложный проект, но достаточно интересный — **микрокомпьютер карманного формата.** Индикация может быть как на знаковых ЖКИ-индикаторах со встроенным контроллером, так и на графических ЖКИ-индикаторах, клавиатура — 64 клавиши, звуковой излучатель, СОМ-порт, позволяющий обмениваться информацией с настольным ПК. Можно добавить LPT-порт для подключения к принтеру. Для хранения информации можно предусмотреть специальный разъем для подключения электрически перепрограммируемой микросхемы постоянной памяти для хранения программ для микрокомпьютера. Современные микросхемы позволяют сделать такой микрокомпьютер достаточно компактным, быстрым и экономичным. Самое сложное в его разработке — написание программного обеспечения.

# Приложение 1

## Вопросы и ответы

### *Вопрос 1*

Есть микроконтроллер AT90S2313 и реле коммутирующее 220 В индуктивную нагрузку (50 Вт сетевой трансформатор от другого устройства). Реле управляется через микросхему ULN2803. В момент переключения реле AVR перезапускается с вероятностью 80%, причем происходит это чаще всего в момент отключения нагрузки. Реле на плате стоит рядом с микроконтроллером. Если я экранирую реле, то становится лучше, но не совсем. Конденсаторы по питанию, супервизор, диод на обмотке реле и т. д., как обычно, на месте. Разводка печатной платы тоже вроде нормальная. Что делать?

### *Ответ 1*

1) Раз микроконтроллер перезапускается — скорее всего помеха наводится на вывод Reset. Рецепт общеизвестен: если Reset глобальный, и бегаёт по всей плате, то вход микроконтроллера надо от него отвязать резистором ( $\sim 1$  кОм), и со входа на «земляную ногу» микроконтроллера навесить керамический конденсатор 10—100 нФ. Если Reset локальный — то стоит проверить разводку печатной платы, и опять — же керамический конденсатор.

2) Супервизор — не панацея от помех, при коротких помехах на шине питания он сам может генерировать ложный сигнал сброса. Поэтому супервизору на питание нужен индивидуальный развязывающий керамический конденсатор, вплотную к его ножкам.

3) Может ресетиться по WDT. Зависание микроконтроллера довольно часто случается, если помеха наводится на кварцевый генератор. Конденсаторы кварцевого резонатора следует подключить к земляной ноге микроконтроллера напрямую, и больше к этой дорожке ничего не подключать.

4) Контакты реле должны быть зашунтированы искрогасящими цепочками. Просто конденсатор я бы не советовал, т. к. не всякие контакты такое выдерживают. Навскидку можно посоветовать конденсатор 100 нФ (сертифицированный на работу  $V_{ac} = 250$  В) после-

довательно с проволочным или объемным углеродным резистором 10...100 Ом (чем меньше тем лучше, но контакты при малых сопротивлениях могут потихоньку портиться). Кроме этого, не мешает зашунтировать обмотку трансформатора MOV-варистором на  $V_{ac} = 275\text{ В}$ , причем MOV лучше отнести как можно ближе к нагрузке (трансформатору).

5) Обмотка реле должна быть зашунтирована диодом (одним).

6) Остальное — разводка. Возможно, есть где-то проводок, к которому и AVR подключен, и ток с реле или нагрузки бежит. Проверяется просто: AVR запитывается от батарейки, отключаются от схемы с AVR все внешние соединения кроме одного провода земли и оптронной развязки на реле. Важно: заземление только в одном месте одним проводом, лучше всего от GND вывода микроконтроллера на ввод питания остальной схемы. Будет сбоить, тогда похоже это электромагнитное излучение, но почти наверняка не будет. Тогда надо подключать назад и искать, где ошибка.

### *Вопрос 2*

Почему программа, написанная для AT90S1200, не работает на AT90S2313?

### *Ответ 2*

Если пробовать использовать код, написанный для AT90S1200, для микроконтроллера AT90S2313, он часто не работает, даже если в исходном тексте был изменено название файла определений 1200.def на 2313.def. Признаки могут быть следующие: неожиданный сброс микроконтроллера или он может вообще не запускаться. Часто программа при ее отладке в программе-симуляторе работает правильно, а в собранной схеме — не работает. Простые программы работают на обоих микроконтроллерах, а ваше приложение — нет.

### *Решение проблемы*

Микроконтроллер AT90S1200 не имеет встроенной оперативной памяти данных (SRAM), поэтому для работы подпрограмм используется 3-х уровневый аппаратный стек. Его не нужно инициализировать в начале программы. Микроконтроллер AT90S2313 (и многие другие) имеет оперативную память данных, и его стек расположен в этой памяти. Для правильной работы в начале программы этот стек должен быть инициализирован.

Так как стек растет вниз, наиболее удобно располагать стек в конце оперативной памяти данных.

Пример для AT90S2313:

```
ldi rTEMP, RAMEND  
out SPL, rTEMP
```

Микроконтроллеры с большим объемом оперативной памяти данных, такие как AT90S8515, имеют два регистра для указателя стека.

Пример для AT90S8515:

```
ldi rTEMP, low(RAMEND)  
out SPL, rTEMP  
ldi rTEMP, high(RAMEND)  
out SPH, rTEMP
```

Простейшие программы могут работать без инициализации стека в том случае, если они не используют ни одной команды, для которой нужен стек (вызова подпрограмм и т. д.).

*Вопрос 3*

Когда я использую встроенный UART, могу ли я использовать все таймеры микроконтроллера?

*Ответ 3*

Да, можете, т. к. UART имеет свой отдельный таймер для формирования необходимой для работы частоты.

*Вопрос 4*

Где я могу найти фирменные описания микроконтроллеров?

*Ответ 4*

На официальном сайте фирмы Atmel: [www.atmel.com](http://www.atmel.com)

*Вопрос 5*

Какие особенности имеют микроконтроллеры семейства AVR по сравнению с традиционными?

*Ответ 5*

Более эффективная система команд, позволяющая получить более компактный код, особенно при программировании на языке C. В 4—5 раз большая скорость работы. Перепрограммируемая в системе флеш память программ и данных. Маленькое потребление энергии.

**Вопрос 6**

Почему для каждого порта ввода/вывода имеется 3 регистра?

**Ответ 6**

Для того, чтобы позволить реализовать действительно безопасную систему, AVR позволяет организовать работу с портами по принципу «прочел-модифицировал-записал». Если Вы хотите прочитать физическое состояние выводов микроконтроллера, следует прочитать значение из регистра PINx. В случае, когда нужно обновить данные на выходе линий, следует прочесть содержимое регистра PORTx. При этом будет считано число, которое было выведено в порт. После модификации его можно снова записать в этот же регистр. Этот метод позволяет получать верные результаты независимо от физического состояния выводов микроконтроллера. Кроме того, эта особенность микроконтроллеров семейства AVR позволяет избавиться от необходимости хранить копию содержимого регистра ввода/вывода в памяти.

При использовании команд SBI и CBI, следует использовать PORTx.

## Приложение 2

### Полезные ссылки в Интернет

Фирма Atmel — производитель микроконтроллеров семейства AVR:

<http://www.atmel.com>

Русская страничка Atmel:

<http://www.atmel.ru>

Страничка компилятора AVR GCC, а также примеры программ:

<http://www.avrfreaks.com>

Страничка Peter Fleury:

<http://www.mysunrise.ch/users/pfleury/index.html>

Страница финальных проектов Cornell University:

<http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects>

«Телесистемы». Здесь можно найти очень интересные конференции:

<http://www.telesys.ru/>

# Содержание

<b>ВВЕДЕНИЕ.</b>	<b>3</b>
<b>ГЛАВА 1. МИКРОКОНТРОЛЛЕР</b>	<b>5</b>
1.1. Знакомство с микроконтроллером	5
1.2. Разработка конструкций на микроконтроллерах	7
1.3. Микроконтроллеры семейства AVR	9
Почему именно AVR?	9
<b>ГЛАВА 2. ОПИСАНИЕ МИКРОКОНТРОЛЛЕРА</b>	
<b>AT90S2313 ФИРМЫ ATMEL</b>	<b>15</b>
2.1. Описание выводов	15
Кварцевый генератор	17
2.2. Обзор архитектуры AT90S2313	17
Файл регистров общего назначения	17
Режимы адресации	20
Арифметико-логическое устройство	25
Память программ	25
EEPROM память данных	25
Оперативная память данных	25
Время выполнения команд	27
Регистр состояния — SREG	29
Указатель стека SP	30
2.3. Перезапуск микроконтроллера (сброс) и обработка прерываний	31
Источники сброса	32
Сброс по включении питания	33
Внешний сброс	33
Сброс по сторожевому таймеру	33
2.4. Обработка прерываний	33
Внешние прерывания	37
Время реакции на прерывание	38
2.5. Режимы пониженного энергопотребления	40
Режим холостого хода	40
Экономичный режим	41
2.6. Таймеры/счетчики	41
8-разрядный таймер/счетчик 0	41
16-разрядный таймер/счетчик 1	44
Таймер/счетчик в режиме ШИМ	49
Сторожевой таймер	51
2.7. Чтение и запись в энергонезависимую память	53
2.8. Универсальный асинхронный приемопередатчик	55
Передача данных	56
Прием данных	57
Управление UART	59
Генератор скорости передачи	62

2.9. Аналоговый компаратор . . . . .	63
2.10. Порты ввода/вывода. . . . .	65
Порт D . . . . .	68
Порт D как порт ввода/вывода общего назначения . . . . .	69
Альтернативные функции порта D . . . . .	70
2.11. Программирование памяти . . . . .	70
Программирование битов блокировки памяти . . . . .	70
Биты конфигурации (Fuse bits) . . . . .	71
Код устройства. . . . .	71
2.12. Параметры микроконтроллера AT90S2313 . . . . .	71
Максимально допустимые параметры . . . . .	72
Характеристики по постоянному току . . . . .	72
Параметры внешнего тактового сигнала . . . . .	73
Варианты исполнения микроконтроллера . . . . .	74
2.13. Набор команд AT90S2313 . . . . .	74
Арифметические и логические команды . . . . .	74
Команды ветвления. . . . .	75
Команды пересылки . . . . .	77
Команды работы с битами. . . . .	78

### **ГЛАВА 3. ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА AVR. . . . . 80**

3.1. Источник питания. . . . .	80
Батарейное питание. . . . .	81
Питание от сети . . . . .	81
Питание от линий портов . . . . .	81
3.2. Внешние элементы тактового генератора. . . . .	82
Использование кварцевого резонатора. . . . .	82
Использование встроенного RC-генератора. . . . .	83
3.3. Цепь сброса . . . . .	83

### **ГЛАВА 4. ПРОГРАММЫ И ИНСТРУМЕНТЫ. . . . . 84**

4.1. Ассемблер . . . . .	84
Начало работы . . . . .	85
Ассемблирование первой программы . . . . .	85
Поиск и исправление ошибок . . . . .	86
Формат программы на ассемблере . . . . .	88
Команды микроконтроллера . . . . .	89
Директивы транслятора ассемблера . . . . .	89
Выражения . . . . .	100
Описание программы WAVRASM . . . . .	104
Версия ассемблера для командной строки . . . . .	107
Формат файлов, генерируемых транслятором ассемблера . . . . .	107
4.2. Компилятор языка C CodeVision AVR . . . . .	110
Изготовление кабеля для внутрисхемного программирования «STK200/300» . . . . .	112
Простая демонстрационная схема на микроконтроллере AT90S8535 . . . . .	113

Использование встроенного программатора CodeVision . . . . .	114
Примеры программ для компилятора CodeVision AVR C . . . . .	116
4.3. Компилятор Imagecraft C . . . . .	135
4.4. Компилятор языка C AVR GCC . . . . .	137
4.5. Программатор . . . . .	146
<b>ГЛАВА 5. ОСНОВНЫЕ СХЕМНЫЕ РЕШЕНИЯ</b>	
<b>ИНТЕРФЕЙСОВ . . . . .</b>	<b>164</b>
5.1. Параллельные выходы . . . . .	164
Управление светодиодами или оптронами . . . . .	164
Управление реле . . . . .	166
Управление светодиодными цифровыми индикаторами . . . . .	167
5.2. Параллельные входы . . . . .	168
Кнопки и переключатели . . . . .	169
<b>ГЛАВА 6. ПРАКТИЧЕСКИЕ ПРИМЕРЫ ПРИМЕНЕНИЯ</b>	
<b>МИКРОКОНТРОЛЛЕРОВ AVR . . . . .</b>	<b>171</b>
6.1. 10-разрядный светодиодный индикатор на AT90S1200 . . . . .	171
6.2. Управление синтезатором частоты радиостанции «Маяк». . . . .	172
6.3. Синтезатор частоты для УКВ ЧМ-радиостанции . . . . .	177
6.4. Телеграфный манипулятор . . . . .	181
6.5. Частотомер до 7200 кГц . . . . .	183
6.6. Частотомер (шкала для UW3DI) . . . . .	184
6.7. Книгочей . . . . .	187
6.8. Преобразование DTMF-сигнала в импульсный . . . . .	190
6.9. Многоточечный термометр . . . . .	192
6.10. Ключ для защиты от копирования . . . . .	194
6.11. Кодовый замок . . . . .	200
6.12. Музыкальный звонок . . . . .	212
6.13. Универсальный расширитель последовательного порта . . . . .	216
6.14. Таймер . . . . .	218
6.15. Универсальный параллельный адаптер . . . . .	225
6.16. Электронные часы с будильником на AT90S2313-10PI . . . . .	229
6.17. Подключение внешнего статического оперативного запоминающего устройства . . . . .	234
Интерфейс для подключения внешней памяти . . . . .	234
Пример подключения внешней оперативной памяти к микроконтроллеру AT90S8515. . . . .	236
Отладка программы . . . . .	243
Программа записи данных в ОЗУ 512 Кбайт . . . . .	264
<b>ГЛАВА 7. ИДЕИ ДЛЯ ПРОЕКТОВ . . . . .</b>	<b>275</b>
<b>Приложение 1. Вопросы и ответы . . . . .</b>	<b>280</b>
<b>Приложение 2. Полезные ссылки в Интернет . . . . .</b>	<b>283</b>