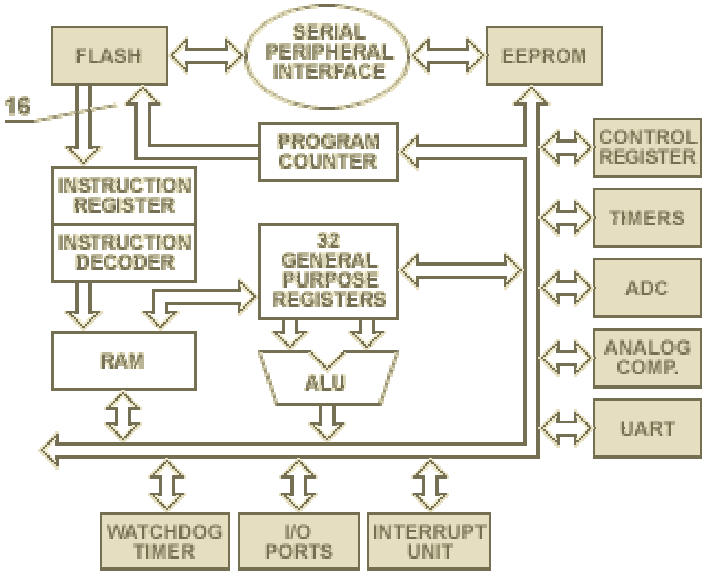
А. Водовозов

МИКРОКОНТРОЛЛЕРЫ ДЛЯ СИСТЕМ АВТОМАТИКИ

Учебное пособие



Вологда
2001

ББК 32.85
УДК 621.375(03)
В62

Рецензенты:
Кафедра РАПС Санкт-Петербургского государственного электротехнического университета (ЛЭТИ), зав. кафедрой профессор Рассудов Л.Н.,
Соловьев В.А., профессор, заведующий кафедрой автоматизированного электропривода Комсомольского на Амуре государственного технического университета .

В62
Водовозов А.М. Микроконтроллеры для систем автоматики: Учебное пособие.- Вологда: ВоГТУ, 2002.- 131 с.

Учебное пособие предназначено для студентов специальности 180400, изучающих курс «Элементы систем автоматики». Все основные понятия микропроцессорной техники в пособии рассмотрены на примерах современных AVR-микроконтроллеров фирмы *Atmel*.

Пособие может быть использовано студентами других специальностей при изучении современных микропроцессорных систем.

ISBN

ББК 32.85
УДК 621.375(03)

ВВЕДЕНИЕ

Создание фирмой Intel в 1971 году первой программируемой электронной схемы на одном кристалле явилось началом эпохи микропроцессорной техники. Объединив в себе достижения вычислительной техники и микроэлектроники, микропроцессорные системы стали удобным общепринятым инструментом для построения самых различных систем автоматики.

За 30 лет своего бурного развития микропроцессорные системы прошли путь от специализированных комплектов интегральных схем к сложным однокристальным микроконтроллерам, имеющим в своем составе целый набор самых различных программируемых элементов. Их разработкой занимаются практически все крупнейшие мировые производители компьютеров, бытовой техники, промышленных систем и электронных компонентов – всем известные: INTEL, AMD, ATMEL, MICROCHIP, MITSUBISHI, MOTOROLA, ANALOG DEVICE, NATIONAL SEMICONDUCTOR, SIEMENS, TEXAS INSTRUMENT, ZILOG и др. Они выпускаются сейчас, как и все остальные микросхемы, миллионными партиями, а стоимость одного микроконтроллера, как правило, составляет всего несколько долларов.

В условиях существующей в области микроэлектроники жесткой конкуренции каждый производитель выбирает свою модель развития, предлагая множество новых технических и технологических решений. При этом не успевают устояться условные обозначения и терминология, появляется множество архитектур и языков программирования, что существенно усложняет изучение и освоение этой техники. В нашей стране ситуация усугубляется языковым барьером, вся основная терминология в этой области техники имеет английскую аббревиатуру.

Вместе с тем, принципы построения микропроцессорных систем, несмотря на множество направлений развития, не подвергаются существенной переработке. Заложенные в 70-х годах, при всем многообразии изделий, они без значительных изменений сохраняются и до настоящего времени. По этой причине изучение микропроцессорной техники возможно, и всегда строится, на конкретных примерах какой либо одной архитектуры. Именно по этому принципу строятся учебные курсы, предлагаемые в настоящее время ведущими университетами и учебными центрами. В качестве примеров обычно рассматриваются микроконтроллеры *Intel, Motorola, Zilog, Microchip* [1-5].

Все основные понятия микропроцессорной техники в пособии рассмотрены на примерах современных AVR-микроконтроллеров фирмы *Atmel*. Этот выбор обусловлен целым рядом факторов, таких как распространенность микроконтроллеров в России, доступность технической информации, наличие свободно распространяемых программных и сравнительно недорогих аппаратных средств поддержки проектирования. Использованные при написании пособия оригинальные материалы можно найти на сайте www.atmel.ru.

1. АРХИТЕКТУРА МИКРОКОНТРОЛЛЕРА

Контроллером в технике регулирования считается управляющее устройство, осуществляющее регулирующие или контролирующие функции в системе. Контроллер, реализованный на одном кристалле, называется микроконтроллером. Современный микроконтроллер является большой цифровой интегральной схемой, объединяющей миллионы, выполненных по микронным технологиям, транзисторов.

Типовая структура микроконтроллера изображена на рис. 1.1. Микроконтроллер состоит из трех, связанных системными шинами, элементов: процессорного ядра, памяти и набора программируемых функциональных блоков различного назначения.

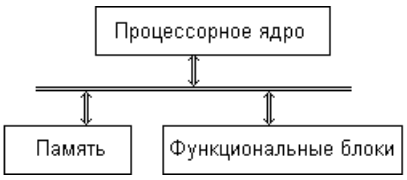


Рис. 1.1. Структура микроконтроллера

Процессорное ядро *(MCU - Microprocessor Core Unit)* является основой микроконтроллера. Оно выполняет все вычислительные операции и, одновременно, управляет работой всех остальных элементов схемы. По системным шинам процессорное ядро обменивается данными с памятью и всеми функциональными блоками. Разрядность процессорного ядра определяет разрядность микроконтроллера. Наиболее распространены в настоящее время 8-битные (8-разрядные) микроконтроллеры. Вместе с тем, широкое применение в простых задачах находят и 4-битные изделия, а в сложных высокопроизводительных системах 16- и 32-битные.

В памяти *(Memory*) хранится программа работы микроконтроллера, исходные данные и все промежуточные результаты вычислений. Память состоит из множества многоразрядных ячеек, каждая из которых имеет свой *адрес*. По этому адресу процессорное ядро находит конкретную ячейку памяти в процессе обмена. Память микроконтроллера обычно разделена на две части: *память данных* (*Data Memory*) и *память программ* (*Program Memory*).

Функциональные блоки различных типов обеспечивают взаимодействие микроконтроллера с внешним миром. Эти блоки могут выполнять самые различные функции: ввод и вывод информации, подсчет внешних событий и интервалов времени, передача внешних запросов на процессорное ядро, аналого-цифровые и цифроаналоговые преобразования сигналов, сравнение различных величин, контроль за напряжением питания и др. Для процессорного ядра любой функциональный блок представляется в виде одного или нескольких регистров. Каждый регистр имеет свой оригинальный адрес, по которому процессорное ядро находит его в процессе работы.

Программа работы микроконтроллера хранится в памяти в виде последовательности команд (инструкций). В процессе работы процессорное ядро последовательно извлекает из памяти инструкции, расшифровывает и выполняет их. В зависимости от инструкции в ядре выполняются различные арифметические и логические операции, пересылки данных. При необходимости, в процессе выполнения инструкции, процессорное ядро обращается за данными к ячейкам памяти и функциональным блокам, либо пересылает в них результаты вычислений. Множество инструкций, которые понимает процессорное ядро, образует *систему команд* микроконтроллера.

Практически все ведущие производители разрабатывают целые семейства микроконтроллеров с так называемой *модульной структурой*. При этом процессорное ядро для всего семейства неизменно, а память и состав функциональных блоков у каждого микроконтроллера различны. Процессорное ядро всегда имеет свою оригинальную схему

и, обязательно, оригинальное имя. Например, микроконтроллеры фирмы *Motorola* построены на базе ядра HC05 и HC08, фирма *Intel* создала ядро MCS-51 и MCS-96, контроллеры фирмы *Microchip* строятся на базе ядра PIC12, PIC16, PIC17, PIC18, фирма *Atmel* усиленно развивает семейство микроконтроллеров с ядром AVR.

Процессорное ядро на основе известных схемотехнических решений, технологий проектирования и производства цифровых схем реализует определенную *архитектуру* системы. Для микропроцессорной системы понятие «архитектура» включает в себя множество её структурных особенностей, основными из которых считаются: организация памяти и система команд. В настоящее время известны четыре общих архитектурных принципа в той или другой мере, реализуемые в любом процессорном ядре.

По организации памяти различаются:

- *Неймановская архитектура* - характеризуется общим пространством памяти для хранения данных и программы. При этом разрядность памяти зафиксирована (как правило, равна одному байту). Такую архитектуру имеют, например, микроконтроллеры HC05 и HC08 фирмы *Motorola*, в которых общий массив 8-битных ячеек памяти включает в себя как память программ, так и память данных [5].
- *Гарвардская архитектура* – отличается разделением памяти программ и памяти данных. При этом разрядность памяти программ и памяти данных, а также шины доступа к ним, различны. В частности, все микроконтроллеры PIC12, PIC16 фирмы Microchip имеют 8-битную память данных, а разрядность памяти программ у них различна: PIC12 имеют 12 битную память программ, а PIC16 – 14 битную [3].

По системе команд различаются:

- *CISC-архитектура (Complicated Instruction Set Computer)* – архитектура с развитой системой команд. Система команд процессорного ядра имеет инструкции разного формата: однобайтовые, двухбайтовые, трехбайтовые. Различные инструкции при этом имеют и существенно разное время исполнения.
- *RISC-архитектура (Reduced Instruction Set Computer)* – архитектура с сокращенным набором команд. Одна инструкция, как правило, занимает только одну ячейку памяти, и все инструкции имеют равное время исполнения.

Микроконтроллеры с RISC-архитектурой имеют сравнительно более высокую производительность при той же тактовой частоте сигнала синхронизации и в настоящее время более распространены.

Разные производители в своих изделиях используют зачастую различные архитектурные принципы. Поэтому приведенное выше деление довольно условно

Например, AVR-микроконтроллеры фирмы *Atmel,* по мнению её создателей, имеют улучшенную RISC (*enhanced RISC*) архитектуру. В соответствии с принципами RISC – архитектуры практически все команды микроконтроллера (исключая те, у которых одним из операндов является 16-разрядный адрес) занимают только в одну ячейку памяти программ. Но сделать это разработчикам удалось за счет одновременного использования принципов Гарвардской архитектуры и расширения ячейки памяти программ до 16 разрядов. Поэтому в системе команд AVR-микроконтроллеров целых 130 различных команд, что значительно больше, чем у большинства современных RISC – архитектур. Для сравнения, контроллеры фирмы *Microchip* с ядром PIC12, PIC16, PIC17 имеют всего 33 команды [3].

2. ПАМЯТЬ

Память микроконтроллера предназначена для хранения инструкций программы и данных. В микроконтроллерах с Гарвардской архитектурой она разделена на отдельные блоки: память программ *(Program Memory*) и память данных *(Data Memory).*

2.1. Память программ

Программа микропроцессора представляет собой последовательность команд (инструкций). Каждая инструкция имеет свой оригинальный двоичный код. Коды инструкций и хранятся в памяти программ

Память программ состоит из множества ячеек определенной разрядности, каждая из которых имеет свой номер (адрес). Количество ячеек (объем памяти) может быть различно. Обычно ячейки памяти программ нумеруются в шестнадцатеричной системе счисления, начиная нуля: $0, $1, $2 …..Знаком $ в дальнейшем будем обозначать числа в шестнадцатеричной системе счисления.

Память программ, по существующей классификации, всегда является какой либо разновидностью постоянной памяти (*ROM – Read Only Memory*). Постоянная память энергонезависима, она способна хранить записанную в неё информацию при отсутствии питающего напряжения. Основным режимом такой памяти является считывание данных, но способы записи программы (способы программирования) памяти могут быть самые разные. В зависимости от способа программирования память типа ROM разделяется на несколько групп: *maskROM, OTPROM, EPROM, EEPROM u flash memory*.

МАСОЧНАЯ ПАМЯТЬ

Масочная память *(maskROM*) программируется с помощью фотошаблонов *(масок)* на стадии изготовления микроконтроллера. Т.е. контроллер с масочной памятью изготавливается с записанной в память программой. Содержимое такой памяти не может быть изменено в процессе настройки и эксплуатации изделия. Память этого типа считается самой дешевой, но её применение целесообразно только при изготовлении уже спроектированных и не требующих доработки устройств очень большими партиями, когда становится целесообразным оформление отдельного заказа на изготовление большого количества запрограммированных по одному шаблону кристаллов.

ОДНОКРАТНО ПРОГРАММИРУЕМАЯ ПАМЯТЬ

Однократно программируемая память (*OTPROM - One Time Programmable ROM*) по принципу построения и функционирования аналогична масочной, но она поставляется изготовителем микроконтроллера незапрограммированной, а контроллер имеет режим программирования. Каждая ячейка памяти в исходном состоянии, как правило, содержит код $FF. Операция программирования заключается в избирательном разрушении (пережигании) части плавких перемычек, включенных в элементы памяти. В этом случае отдельные биты в ячейках памяти принимают нулевые значения. Восстановить исходное значение ячейки после программирования невозможно. Программирование OTPROM осуществляется в специальных приборах – программаторах, обеспечивающих заданные изготовителем технические условия программирования.

Контроллеры с OTPROM относительно дешевы. Их применение целесообразно при серийном изготовлении изделий даже сравнительно небольшими партиями.

Репрограммируемая память

Репрограммируемая память (*EPROM - Erasable PROM)* аналогична OTPROM, но допускает стирание информации и повторное программирование. Стирание информации в памяти осуществляется с помощью интенсивного ультрафиолетового излучения. Корпуса таких микросхем имеют специальные окна, закрытые кварцевым стеклом, через которые излучение попадает на кристалл. Число циклов стирания и программирования EPROM относительно небольшое (20….100).

Стоимость контроллеров с EPROM относительно велика и изделия с такой памятью рекомендуется использовать только на стадии проектирования или при изготовлении относительно малых (опытных) партий изделий.

Память с электрическим стиранием

Память с электрическим стиранием (*EEPROM - Electrically EPROM)* программируется пользователем и может многократно стираться. Стирание и повторное программирование EEPROM осуществляется по отдельным ячейкам (побайтно), что позволяет вносить в разработанные программы даже небольшие коррективы. Число циклов стирания и программирования, допускаемых EEPROM, обычно порядка 10000. Для реализации технологии необходимо относительно высокое напряжение программирования в пределах 10…20В, которое подается на кристалл извне или формируется на кристалле встроенным преобразователем напряжения (генератор накачки).

Контроллеры с EEPROM относительно дешевы (по сравнению с EPROM), но емкость такой памяти ограничена из-за сложности ячеек. Кристаллы с такой памятью программ применяются довольно редко, только в относительно простых системах на стадиях проектирования и серийного производства.

Флэш - память

Флэш-память *(Flash memory)* относится к классу EEPROM, но использует особую технологию построения запоминающих ячеек. В отличие от EEPROM, она может стираться только целиком, либо достаточно большими блоками. Возможность перезаписи отдельных ячеек памяти в ней отсутствует. Кристаллы с флэш - памятью всегда содержат встроенные генераторы накачки и при соответствующей аппаратной и программной поддержке позволяют реализовать режим «программирования в системе» - программирование без извлечения микроконтроллера из изделия. Необходимость использования программатора в этом случае отпадает.

Современные технологии изготовления *Flash Memory* обеспечивают гарантированное число циклов стирания/записи до 1000….10000, срок хранения до 10 лет.

Флэш-память программ микроконтроллера может быть переписана самим микроконтроллером. Для организации этого режима в структуре памяти предусматривается специальный раздел начальной загрузки, где располагается специальная программа.

Например, микроконтроллер ATmega163 имеет 8K 16-битных ячеек Flash-памяти программ. Общий объем памяти 16 Кбайт. При этом всё адресное пространство флэш-памяти программ ($0000…$1FFF) разделено на два раздела (рис. 2.1): раздел программы начальной загрузки *(Boot Program Section*) размер которой может находиться в пределах от 256 до 2048 байт и раздел прикладной программы (*Application Program Section*).

Оба раздела могут быть программно заблокированы от записи. Кроме того, в системе команд микроконтроллера предусмотрена специальная инструкция spm (*store program memory*), осуществляющая запись данных в *Application Program Section*. Эта команда может быть использована только в *Boot Program Section* .

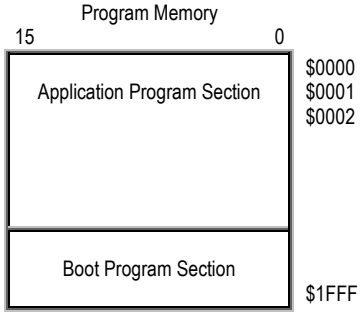


Рис. 2.1. Организация flash-памяти программ микроконтроллера ATmega 163

В *Boot Program Section* записывается программа *Flash-resident Boot Loader* , позволяющая процессорному ядру микроконтроллера организовать перепрограммирование раздела прикладной программы непосредственно в изделии. Программа *Boot Loader* может использовать различные способы связи микроконтроллера с внешними устройствами для ввода кода программы.

Весь массив 8К ячеек *flash memory* микроконтроллера ATMEGA163 разделен на 128 страниц по 64 слова. Размер раздела начальной загрузки может быть задан с помощью специальных fuse битов BOOTSZ, как показано в таблице 2.1.

Таблица 2.1.

Конфигурирование секции начальной загрузки *(Boot Section*)

BOOTSZ1	BOOTSZ0	Объем (слов)	К-во страниц	Адреса
1	1	128	2	$1F80 - $1FFF
1	0	256	4	$1F00 - $1FFF
0	1	512	8	$1E00 - $1FFF
0	0	1024	16	$1C00 - $1FFF

2.2. Память данных

Память данных предназначена для хранения результатов вычислений в процессе работы микроконтроллера. Она организована, как и память программ, в виде множества ячеек, каждая из которых имеет свой адрес. В процессе работы микроконтроллер обращается к ячейкам памяти данных при выполнении команд загрузки (чтения) и записи.

Память данных микроконтроллера может быть двух типов: SRAM и EEPROM.

СТАТИЧЕСКАЯ ПАМЯТЬ

Статическая память *(SRAM – Static Random Access Memory)* энергозависима. Она обеспечивает хранение информации только при наличии напряжения питания не менее определенной величины (порядка 1….3В). Некоторые микроконтроллеры при данном напряжении не работают, но данные в памяти сохраняются. Для обеспечения длительной сохранности данных в такой системе необходим резервный источник (аккумулятор или батарея), подключающийся при отключении основного. Отдельные микроконтроллеры, например МК DS50000 фирмы *Dallas Semiconductor*, даже имеют в своем корпусе

автономный источник резервного питания, обеспечивающий сохранение данных в течении нескольких лет.

Каждая ячейка статической памяти в микроконтроллере имеет свой адрес. Некоторые ячейки могут иметь специальное назначение.

Например, микроконтроллер *ATmega 163* имеет 1024 восьмибитные ячейки встроенной статической памяти типа *SRAM*. Объем статической памяти микроконтроллера равен 1кбайт. Сама память занимает адресное пространство $60…..$45F (рис. 2.2). Размещенные в том же адресном пространстве микроконтроллера 32 ячейки с адресами $0…1F используются микроконтроллером как регистры общего назначения (*General Purpose Registers*), а 64 ячейки с адресами $2F…$5F – как регистры ввода/вывода (*Input Output Registers*).

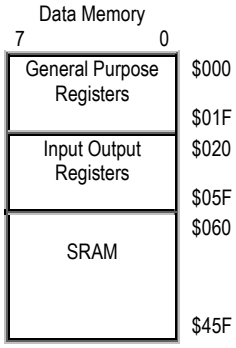


Рис. 2.2.Статическая память данных микроконтроллера ATmega 163

Регистры общего назначения и регистры ввода/вывода используются при работе процессорного ядра.

Данные сохраняются в SRAM при напряжении питания в пределах от 4.0 до 5,5В. При этих же значениях напряжения работоспособно и микропроцессорное ядро микроконтроллера ATmega 163.

Память с электрическим стиранием

Память данных типа EEPROM может быть включена в общее пространство памяти данных или организована в виде отдельного массива с возможность считывания и записи каждого отдельного байта. В отличие от SRAM она допускает только ограниченное (порядка 100000) циклов перезаписи. Время записи в EEPROM значительно больше, чем время записи в статическую память.

Содержимое EEPROM может быть разрушено при снижении напряжения питания в процессе записи. Разрушение данных EEPROM, происходит по двум причинам. Во-первых, для операций записи необходимо, чтобы напряжение питания было не ниже определенного уровня, гарантирующего правильное их выполнение. Во-вторых, само ядро микроконтроллера при слишком низком напряжении питания, может неправильно выполнять команды. Защита EEPROM от разрушения при снижении напряжения питания в контроллерах решается как программными, так и аппаратными средствами.

Например, EEPROM микроконтроллера ATmega163 имеет объем 512 байт. Она обеспечивает 100000 циклов стирания/записи. Обращение к EEPROM ведется через

специальные регистры ввода/вывода: регистр адреса EEPROM, регистр данных EEPROM и регистр управления EEPROM

Время записи, в зависимости от напряжения питания, может составлять от 1,9 до 3,8 мс. Существует специальная функция, которая позволяет пользовательскому программному обеспечению обнаружить момент окончания записи *(EEPROM Write Complete*). Случайная запись в EEPROM предотвращается выполнением специальной процедуры.

Для защиты EEPROM от разрушения в микроконтроллере предусмотрен детектор снижения напряжения питания. При снижении напряжения питания ниже уровня 2,9В операция записи не производится. В этом случае микроконтроллер переходит в режим ожидания, при котором процессорное ядро не может выполнять инструкции программы.

2.3. Специализированные ячейки флэш-памяти

В энергонезависимой flash-памяти микроконтроллеров могут присутствовать специализированные биты и байты, предназначенные для защиты программы пользователя и конфигурирования изделия. Эти ячейки памяти не отображаются в общем массиве памяти программ, имеют оригинальные имена и индивидуально программируются. Специализированные flash-ячейки имеет и микроконтроллер *ATmega163*. В его структуре предусмотрены:

- Биты блокировки (*lock-bits*) LB1 и LB2, позволяющие запретить чтение программы, EEPROM и программирование кристалла. Запрограммированные биты блокировки можно стереть только при очистке *flash –памяти* программ. При этом уничтожается и сама программа.

Таблица 2.2.

Режимы защиты программной информации *AVR*

LB1	LB2	Тип защиты
1	1	Защита отсутствует
0	1	Запрет программирования Flash и EEPROM
0	0	Запрет программирования и чтения Flash и EEPROM

- Биты предохранители *(fuse-bits*) позволяют задавать некоторые конфигурационные особенности микроконтроллера. Состав *fuse*-битов каждого конкретного микроконтроллера обусловлен особенностями построения узлов сброса, тактирования и программирования кристалла. *ATmega163,* поддерживающий режим последовательного программирования имеет fuse-бит SPIEN, который может запретить этот режим перезаписи кристалла. Кроме того, fuse-биты CKSEL[0..3] задают источник тактового сигнала микроконтроллера и время задержки старта микроконтроллера после сброса, а схемой контроля питания управляют fuse-биты BODEN и BODLEVEL. Fuse-биты BOOTSZ, как показано ранее, задают объем секции начальной загрузки в памяти программ.
- Байты сигнатуры (*signature-byte*) служат для идентификации типа кристалла, программируются изготовителем и доступны только для чтения.

3. ПРОЦЕССОРНОЕ ЯДРО

3.1. Основные элементы

Каждый производитель микроконтроллеров для серии выпускаемых им изделий разрабатывает и патентует своё оригинальное процессорное ядро *(MCU – Microprocessor Core Unit).* Однако в большинстве из них присутствуют одни и те же элементы:

- регистр инструкций,
- программный счетчик,
- арифметико-логическое устройство,
- регистры общего назначения,
- регистр состояния
- регистры ввода/вывода,
- стек.

Программный счетчик

Программный счетчик *(PC - Program counter)* – регистр, предназначенный для хранения адреса ячейки памяти программ, в которой находится выполняемая в данный момент инструкция. Разрядность программного счетчика определяется количеством ячеек в памяти программы. При выполнении команды содержимое программного счетчика изменяется. В простейшем случае оно увеличивается на единицу. Но некоторые команды сами способны записывать данные в программный счетчик. В этом случае новое содержимое программного счетчика и определяется данными, заложенными в выполняемой инструкции.

Например, программный счетчик у микроконтроллеров с ядром AVR имеет разрядность 16 бит. В общем случае он позволяет адресовать до 64К ячеек памяти. Микроконтроллер ATmega163 с ядром AVR имеет только 8К ячеек памяти программ, поэтому в нем используется только 13 младших бит программного счетчика ядра AVR (2^{13} = 8К).

Регистр инструкций

Регистр инструкций (*IR - Instruction register*) – регистр, предназначенный для хранения считанной из памяти программ инструкции. Считанная из памяти программ инструкция декодируется дешифратором команд и исполняется микропрограммным автоматом ядра. Разрядность регистра инструкций определяется разрядностью памяти программ.

Например, ядро AVR в своей структуре имеет 16-битный регистр инструкций. Эта разрядность обеспечивает ему работу с 16-битной памятью программ.

Арифметико-логическое устройство

Арифметико-логическое устройство (*ALU - Arithmetic Logic Unit*) – логическая схема, непосредственно осуществляющая преобразование одной или двух переменных в соответствии с инструкцией занесенной в регистр команд. Стандартное ALU способно выполнять простейшие арифметические и логические операции над одной или двумя переменными.

Типовые арифметические операции ALU:

- сложение (*addition*),
- вычитание (*subtract),*
- инкремент (*increment*),

- декремент (*decrement*).

Типовые логические операции:

- инверсия (NOT),
- логическое сложение (OR),
- логическое умножение (AND),
- исключающее ИЛИ (exclusive OR).

Некоторые производители интегрируют на кристалл также встроенный умножитель двух переменных.

В частности ALU ядра AVR способно выполнить арифметические операции: сложение, вычитание, инкремент, декремент, а также и логические операции: логическое сложение, логическое умножение, исключающее ИЛИ и очистка регистра. Оно снабжено внутренним умножителем, способным выполнять перемножение 8-битных целых и дробных чисел без знака и со знаком.

Все арифметические и логические операции (за исключением умножения) выполняются за один такт работы процессорного ядра. Перемножение двух переменных выполняется за два такта.

Регистры общего назначения

Регистры общего назначения предназначены для временного хранения данных в процессе вычислений. Разрядность регистров определяет разрядность вычислений и, в конечном счете, разрядность самого микроконтроллера. Количество регистров может быть произвольным. Обычно в этих регистрах хранится информация, обрабатываемая в арифметико-логическом устройстве и полученный в нем результат вычислений. На некоторые из регистров могут быть возложены ещё какие либо дополнительные функции. В большинстве архитектур один из регистров отличается от других большими возможностями. Он обычно называется аккумулятор или рабочий регистр. В этом регистре может храниться одна из переменных, обрабатываемых в арифметико-логическом устройстве, и туда же помещается результат операции.

Например, ядро AVR имеет 32 регистра общего назначения(*GPR - General purpose registers*). Каждому регистру (рис. 3.1) соответствует дополнительно адрес в памяти данных, отображающий их в первых 32 ячейках пространства данных. Хотя они не используются как физические ячейки SRAM, такая организация памяти обеспечивает гибкое обращение к регистрам.

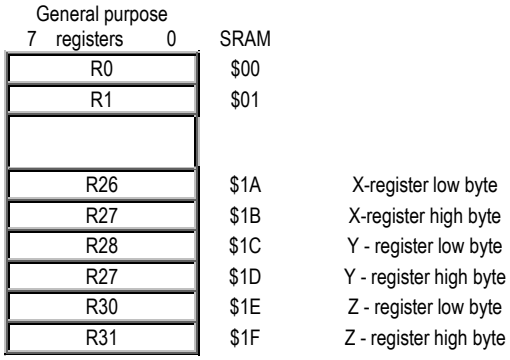


Рис.3.1. Файл регистров общего назначения ядра AVR

Все инструкции, в которых происходит обращение к регистрам, выполняются процессорным ядром AVR в течении одного тактового цикла.

Самые общие арифметические и логические инструкции между двумя регистрами или с одним регистром используют для записи результата тоже регистры этого регистрового файла.

Шесть из 32 регистров (R26 …. R31), кроме обычной для прочих регистров функций, выполняют функцию 16-битных указателей адреса при косвенной адресации памяти данных и памяти программ . Эти три регистра косвенной адресации определяются как регистры X, Y и Z. При этом в четных регистрах хранятся младшие байты 16-битных переменных, а в нечетных – старшие (рис. 3.3).

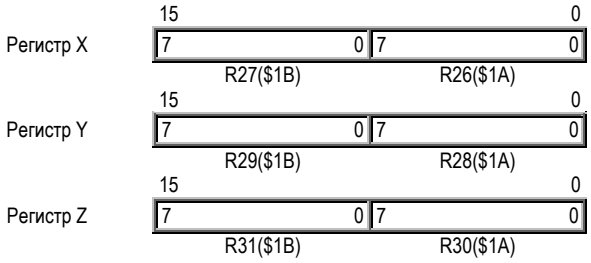


Рис. 3.3 Регистры косвенной адресации архитектуры AVR

Регистры ввода/вывода

Регистры ввода/вывода предназначены для управления функциональными блоками микроконтроллера, энергонезависимой памятью данных и программ. В различных операциях регистры могут участвовать целиком или отдельными битами. Отдельный бит регистра обычно именуется флагом *(flag*). Обращение к регистрам в различных архитектурах организуется различным образом. Обычно обращение к ним осуществляется как к элементам процессорного ядра по присвоенным в архитектуре именам и адресам, в ряде случаев к ним обращаются с помощью специальных команд ввода/вывода.

Например, в микроконтроллере *ATmega163* предусмотрено 64 регистра ввода/вывода (*Input/output registers*). Все они размещены в пространстве ввода/вывода процессорного ядра по адресам от $00 по $3F. Для обращения к регистрам можно использовать специальные инструкции (IN) и вывода (OUT). При этом в инструкции необходимо указывать адрес регистра. Одновременно регистры ввода/вывода представлены также в адресном пространстве памяти данных (рис. 2.2) и к ним можно адресоваться как к обычным ячейкам SRAM с адресами от $20 до $5F. Адрес SRAM получается простым добавлением $20 к непосредственному адресу регистра. В дальнейшем, при описании регистров, адрес SRAM приводится в круглых скобках после непосредственного адреса регистра ввода/вывода. Регистры побитно адресуются специальными командами работы с битами: установки бита в регистре и очистки бита в регистре. Состояние каждого отдельного бита этих регистров может быть проверено командами переходов. Каждый регистр микроконтроллера имеет своё оригинальное имя. Все регистры микроконтроллера ATmega163 представлены в следующей таблице.

Таблица 3.1.

Регистры ввода/вывода микроконтроллера *ATmega163*

I/O адрес (SRAM адрес)	*Имя*	Функция регистра
$3F ($5F)	SREG	*Status Register*
$3E ($5E)	SPH	*Stack Pointer High*
$3D ($5D)	SPL	*Stack Pointer Low*
$3B ($5B)	GIMSK	*General Interrupt Mask Register*
$3A ($5A)	GIFR	*General Interrupt Flag Register*
$39 ($59)	TIMSK	*Timer/Counter Interrupt Mask Register*
$38 ($58)	TIFR	*Timer/Counter Interrupt Flag Register*
$37 ($57)	SPMCR	*SPM Control Register*
$36 ($56)	TWCR	*2-wire Serial Interface Control Register*
$35 ($55)	MCUCR	*MCU general Control Register*
$34 ($54)	MCUSR	*MCU general Status Register*
$33 ($53)	TCCR0	*Timer/Counter0 Control Register*
$32 ($52)	TCNT0	*Timer/Counter0 (8-bit)*
$31 ($51)	OSCCAL	*Oscillator Calibration Register*
$30 ($50)	SFIOR	*Special Function I/O Register*
$2F ($4F)	TCCR1A	*Timer/Counter1 Control Register A*
$2E ($4E)	TCCR1B	*Timer/Counter1 Control Register B*
$2D ($4D)	TCNT1H	*Timer/Counter1 High-byte*
$2C ($4C)	TCNT1L	*Timer/Counter1 Low-byte*
$2B ($4B)	OCR1AH	*Timer/Counter1 Output Compare Register A High-byte*
$2A ($4A)	OCR1AL	*Timer/Counter1 Output Compare Register A Low-byte*
$29 ($49)	OCR1BH	*Timer/Counter1 Output Compare Register B High-byte*
$28 ($48)	OCR1BL	*Timer/Counter1 Output Compare Register B Low-byte*
$27 ($47)	ICR1H T/C 1	*Input Capture Register High-byte*
$26 ($46)	ICR1L T/C 1	*Input Capture Register Low-byte*
$25 ($45)	TCCR2	*Timer/Counter2 Control Register*
$24 ($44)	TCNT2	*Timer/Counter2 (8-bit)*
$23 ($43)	OCR2	*Timer/Counter2 Output Compare Register*
$22 ($42)	ASSR	*Asynchronous Mode Status Register*
$21 ($41)	WDTCR	*Watchdog Timer Control Register*
$20 ($40)	UBRRHI	*UART Baud Rate Register High-byte*
$1F ($3F)	EEARH	*EEPROM Address Register High-byte*
$1E ($3E)	EEARL	*EEPROM Address Register Low-byte*
$1D ($3D)	EEDR	*EEPROM Data Register*
$1C ($3C)	EECR	*EEPROM Control Register*
$1B ($3B)	PORTA	*Data Register, Port A*
$1A ($3A)	DDRA	*Data Direction Register, Port A*
$19 ($39)	PINA	*Input Pins, Port A*

Окончание табл. 3.1.

I/O адрес (SRAM адрес)	*Имя*	Функция регистра
$18 ($38)	PORTB	*Data Register, Port B*
$17 ($37)	DDRB	*Data Direction Register, Port B*
$16 ($36)	PINB	*Input Pins, Port B*
$15 ($35)	PORTC	*Data Register, Port C*
$14 ($34)	DDRC	*Data Direction Register, Port C*
$13 ($33)	PINC	*Input Pins, Port C*
$12 ($32)	PORTD	*Data Register, Port D*
$11 ($31)	DDRD	*Data Direction Register, Port D*
$10 ($30)	PIND	*Input Pins, Port D*
$0F ($2F)	SPDR	*SPI I/O Data Register*
$0E ($2E)	SPSR	*SPI Status Register*
$0D ($2D)	SPCR	*SPI Control Register*
$0C ($2C)	UDR	*UART I/O Data Register*
$0B ($2B)	UCSRA	*UART Control and Status Register A*
$0A ($2A)	UCSRB	*UART Control and Status Register B*
$09 ($29)	UBRR	*UART Baud Rate Register*
$08 ($28)	ACSR	*Analog Comparator Control and Status Register*
$07 ($27)	ADMUX	*ADC Multiplexer Select Register*
$06 ($26)	ADCSR	*ADC Control and Status Register*
$05 ($25)	ADCH	*ADC Data Register High*

Зарезервированные (не указанные в таблице) адреса при программировании микроконтроллера *ATmega163* использовать не допускается.

Регистр состояния

Регистр состояния *(Status register)* предназначен для хранения отдельных признаков результата, полученного при выполнении различных арифметических и логических операций в арифметико-логическом устройстве. Регистр обычно рассматривается состоящим из отдельных бит (флагов), каждый из которых несет в себе определенную информацию о каком либо одном признаке результата. Типовыми флагами регистра состояния являются:

- флаг переноса *(Carry)* – устанавливается при возникновении переноса из старшего разряда результата;
- флаг переполнения *(Overflow)* – устанавливается при переполнении разрядной сетки; при алгебраическом сложении двух двоичных чисел признаком переполнения является наличие переноса в знаковый разряд суммы при отсутствии переноса из её знакового разряда (положительное переполнение) или наличие переноса из знакового разряда суммы при отсутствии переноса в её знаковый разряд (отрицательное переполнение); если и в знаковый, и из знакового разряда суммы есть переносы или нет этих обоих переносов, то переполнение отсутствует.

- флаг отрицательного результата *(Negative)* - устанавливается, когда результат операции является отрицательным числом; отрицательным обычно считается число, содержащее единицу в знаковом (старшем) разряде.
- флаг нулевого результата *(Zero)* – устанавливается, когда результат операции равен нулю;
- флаг полупереноса *(Half Overflow*) – устанавливается при возникновении переноса из младшей тетрады 8-битного числа в старшую (из третьего разряда в четвертый).

Разрядность регистра состояния обычно равна разрядности ядра, но некоторые его биты могут быть не задействованы или задействованы в работе других узлов ядра.

Например, 8-битный регистр состояния ядра AVR (*SREG – Status register*) размещен в пространстве регистров ввода/вывода (табл. 3.1) по адресу $3F ($5F) и его биты определяются в соответствии с рис. 3.4.

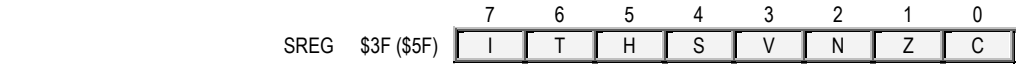


Рис. 3.4. Регистр состояния ядра AVR

- Бит 7 – I *(Global Interrupt Enable)* - бит разрешения глобального прерывания. Для разрешения прерывания должен быть установлен в состояние 1. Работа бита будет рассмотрена позднее.
- Бит 6 – T *(Bit Copy Storage)* - бит сохранения копии. Ряд инструкций копирования используют его как бит-источник или бит-приемник.
- Бит 5 – H (*Half Carry Flag) - флаг* полупереноса.
- Бит 4 – S (*Sign Bit, S = N V)* - флаг знака. Бит S всегда находится в состоянии, определяемом операцией исключающего ИЛИ (*exclusive OR*) между флагом отрицательного значения N и флагом переполнения V.
- Бит 3 – V (*Two's Complement Overflow Flag)* – флаг переполнения (дополнения до двух).
- Бит 2 – N (*Negative Flag)* - флаг отрицательного значения..
- Бит 1 – Z (*Zero Flag)* - флаг нулевого значения.
- Бит 0 – C (*Carry Flag)* - флаг переноса.

Стек

Стек – память данных, организованная по принципу: последний зашел – первый вышел (LIFO - *Last In - First Out*). Такая память предназначается, обычно, для оперативного сохранения содержимого отдельных регистров при переходах к подпрограммам. Одним из таких регистров является программный счетчик. Извлечение из стека содержимого регистров производится в порядке, обратном порядку записи. Запись в стек и извлечение из стека не требует знания адреса ячеек памяти, в которые записываются данные.

Стек может быть организован либо в специально созданных в ядре ячейках памяти, либо в области SRAM. В последнем случае в ядре предусматривается специальный регистр – указатель стека (*Stack Pointer*). Указатель стека хранит адрес последней записанной ячейки памяти в области стека. Поскольку при последовательной записи адреса ячеек всегда изменяются последовательно в сторону уменьшения, а при

изображении памяти SRAM ячейки с меньшими номерами рисуются сверху, то говорят, что стек при записи растет вверх.

Количество ячеек памяти (количество уровней стека), которые используются в стеке для хранения данных, именуется глубиной стека. Глубина стека в различных архитектурах ядра может быть различна: от 2-х ячеек до размера SRAM. Например, в ядре PIC12 фирмы *Microchip* заложен 2-х уровневый стек, разрядность ячеек которого равна разрядности памяти программ (12 бит), а в ядре PIC16 той же фирмы - 8 уровневый стек с разрядностью ячеек 13 бит.

В микроконтроллерах AVR стек организуется в памяти данных типа SRAM. Например, микроконтроллер *ATmega163* оснащен 16-разрядным указателем стека, размещенным в двух регистрах SPH (*Stack Pointer High*) и SPL (*Stack Pointer Low)* пространства ввода/вывода по адресам $3E ($5E) и $3D ($5D). Поскольку рассматриваемый микроконтроллер имеет SRAM с адресами от $000 до $045F (рис. 2.2), то в нем используется только 11 бит указателя стека SP0….SP10 (рис. 3.5).

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

Рис. 3.5. Регистры указателя стека микроконтроллера ATmega163

Указатель стека (SP) указывает на область в статической памяти данных, в которой размещаются стек. Начальное значение указателя стека должно задаваться в программе до первого вызова подпрограмм. Содержимое указателя стека уменьшается на единицу, при каждом занесении данных в стек, и на две единицы при занесении в стек содержимого программного счетчика. Указатель стека увеличивается на единицу, при извлечении данных из стека, и на две единицы при извлечении из стека содержимого программного счетчика. Для работы со стеком в системе команд микроконтроллера предусматриваются специальные инструкции.

3.2. Система команд

Мнемонические обозначения

Каждая архитектура микроконтроллера имеет собственную систему команд. Система команд микроконтроллера описывается на специальном языке символического кодирования. При этом каждая инструкция из системы команд представляется простым трех- или четырехбуквенным мнемоническим символом. Мнемонические символы ассоциируются с функциями команды и однозначно соответствуют действиям процессорного ядра при выполнении этой инструкции. Каждая архитектура имеет фиксированный набор символических команд, созданных и описанных его производителем. Некоторые символические обозначения, используемые в системе команд ядра AVR:

- adc *add with carry* сложить с переносом,
- add *addition* сложить,
- and and операция И,
- clr *clear* очистить,
- dec *decrement* отнять 1,
- in *input* ввод,

- inc *increment* прибавить 1,
- jmp *jump* переход
- ldi *load immediate* загрузить непосредственно,
- mul *multiply* умножить,
- or or операция ИЛИ,
- out *output* вывод,
- ret *return* возврат,
- sub *subtract* вычесть.

Программа, записанная на языке символического кодирования, процессорным ядром не воспринимается. При вводе её в память программ осуществляется перевод каждой инструкции в двоичные коды. При этом каждая символическая команда заменяется её двоичным эквивалентом. Эту операцию можно сделать вручную, используя таблицы машинных кодов микроконтроллера или на компьютере по специальной программе - *ассемблер*. Ассемблер сравнивает каждую мнемоническую команду со списком команд и заменяет её двоичным кодом. Процесс преобразования программы с языка символического кодирования в двоичные коды называется *компиляцией*.

Адресация данных

Адреса операндов, задействованных в выполнении любой инструкции программы, в явном или в неявном виде должны быть указаны в коде этой инструкции. Операнды могут находиться в ячейках памяти данных, в регистрах общего назначения, в регистрах ввода/вывода и даже в ячейках памяти программ. В системе команд микроконтроллера, как правило, различные способы данных, позволяющие программисту задавать адреса операндов в разнообразных инструкциях. В зависимости от архитектуры код инструкции может занимать одну или несколько ячеек программной памяти. При этом для хранения адресов могут отводиться отдельные биты в ячейке или целые ячейки.

Например, в архитектуре AVR большинство команд занимает только одну ячейку памяти программ. В этой 16-битной ячейке содержится двоичный код операции (*OP - Operand*) и адрес регистра или ячейки памяти, задействованных при выполнении этой операции. Вместе с тем, ряд инструкций имеют размерность 32бита. В этом случае 16-битный адрес одного из операндов хранится в 16 младших разрядах кода.

Непосредственная адресация

При непосредственной адресации одним из операндов команды является числовая константа (К). Она хранится непосредственно в коде инструкции. Вторым операндом инструкции должен быть какой либо один из регистров общего назначения.

Например, команда загрузки регистра Rd константой R (*Load Immediate)* на языке ассемблера AVR-микроконтроллера записывается в виде :

```
ldi Rd,K
```

По этой инструкции константа К записывается в указанный регистр Rd. В сокращенном виде эти действия можно представить в виде: Rd <- K.

Формат команды показан на рис. 3.6. В 16-битном коде инструкции номер регистра d занимает 5 бит, константа К – 8 бит, а код операции ldi – 3 бита.

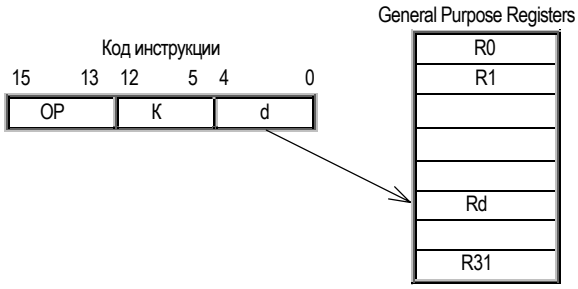


Рис. 3.6. Формат инструкции с непосредственной адресацией данных

Прямая адресация

При прямой адресации операндами могут быть 8-битные переменные, хранящиеся в регистрах общего назначения, ячейках памяти данных и памяти программ или в регистрах ввода/вывода, а также отдельные биты этих регистров. Эти операнды указываются прямо в инструкции.

Операции с одним регистром общего назначения

Операнд инструкции содержится в одном из регистров общего назначения - Rd (регистр-приемник результата ; $0 \le d \le 31$). Номер регистра содержится в коде инструкции.

Например,

inc	Rd	*Increment register*	Rd <- Rd + 1
dec	Rd	*Decrement register*	Rd <- Rd - 1
clr	Rd	*Clear register*	Rd <- $00
ser	Rd	*Set register*	Rd <- $FF

Формат инструкции показан на рис. 3.7. В 16-битном коде адрес регистра занимает 5 младших бит.

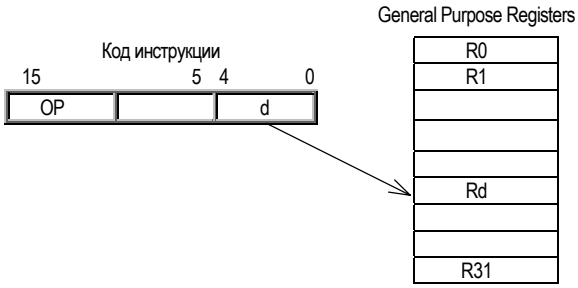


Рис. 3.7. Формат инструкции с прямой адресацией одного регистра общего назначения

Операции с двумя регистрами общего назначения

Операнды инструкции содержатся в двух регистрах общего назначения:

- Rd - регистр-приемник результата ($0 \le d \le 31$)
- Rr - регистр-источник.

Например,

add	Rd, Rr	*addition two registers*	Rd <- Rd + Rr
sub	Rd, Rr	*subtract two registers*	Rd <- Rd - Rr
and	Rd, Rr	*logical and two registers*	Rd <- Rd ·Rr
or	Rd, Rr	*logical or two registers*	Rd <- Rd v Rr

При выполнении инструкций результат операции записывается в регистр-приемник Rd. Номера регистров содержатся в коде инструкции (рис. 3.8).

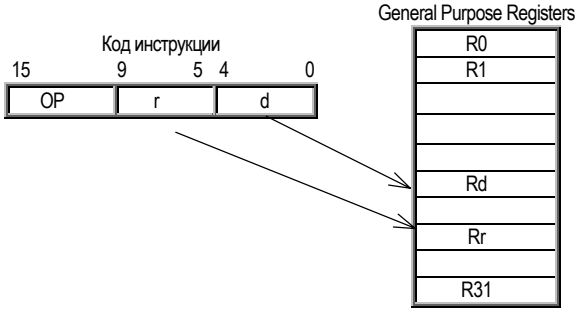


Рис. 3.8. Формат инструкции с прямой адресацией двух регистров общего назначения

Битовые операции с регистром общего назначения

Операндом команды является бит в регистре общего назначения. Номер бита (b) и номер регистра (d или r) задается в коде инструкции.

Например,

bld	Rd, b	*bit load from T to register*	Rd(b) <- T
bst	Rr, b	*bit store from register to T*	T <- Rr(b)

Вторым операндом в инструкциях является бит сохранения копии T *(Copy Storage)* регистра статуса SREG.

В 16-битном формате команды (рис. 3.9) номер бита и номер регистра занимают младшие 8 бит.

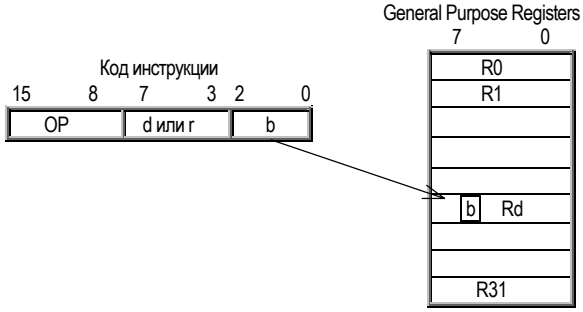


Рис. 3.9. Формат инструкции с прямой адресацией бита в регистре общего назначения

Операции с регистром ввода/вывода

Операнды команды содержатся в одном из регистров общего назначения Rd и в одном из регистров ввода/вывода (с номером $P).

Например,

in	Rd, P	*Input port*	Rd <- P
out	P, Rr	*Output port*	P <- Rr

В 16-битном формате команды номера регистров занимают младшие 11 бит (рис. 3.10).

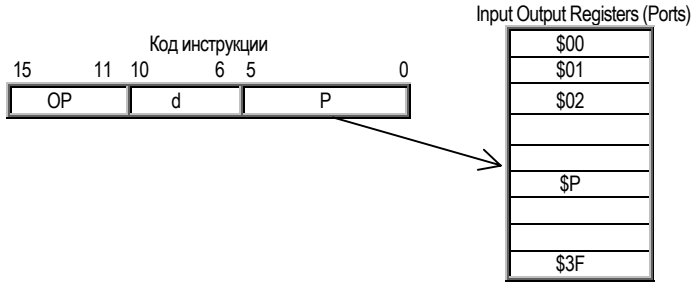


Рис.3.10. Формат команды с прямой адресацией регистра ввода/вывода

Битовые операции с регистром ввода/вывода

Операндом команды является бит в ввода/вывода. Побитно адресуемыми регистрами в пространстве ввода вывода являются регистры с номерами $00….$1F.

Например,

sbi	P*,b	*Set Bit in I/O Register*	I/O(P,b) <- 1
cbi	P*,b	*Clear Bit in I/O Register*	I/O(P,b) <- 0

Номер бита (b) и номер побитно адресуемого регистра P* задается в коде инструкции (рис. 3.11).

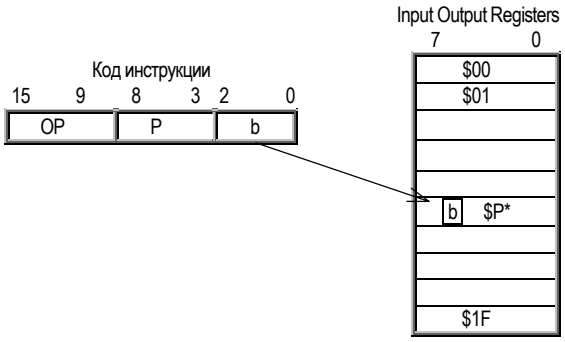


Рис. 3.11. Формат команды с прямой адресацией в регистре ввода/вывода

Операции с памятью данных

Операнды команды содержатся в одном из регистров общего назначения Rd или Rr и в одной из ячеек памяти данных (с номером $K).

Например,

lds	Rd, k	*load direct from SRAM*	Rd <- (k)
sts	k, Rr	*store direct to SRAM*	(k) <- Rr

В 32-битном коде инструкции (рис. 3.12) адрес ячейки памяти занимает младшие 16 бит (младшее 16-битное слово), а номер регистра хранится в битах с номерами 16…19 (четыре младших бита старшего 16-битного слова)

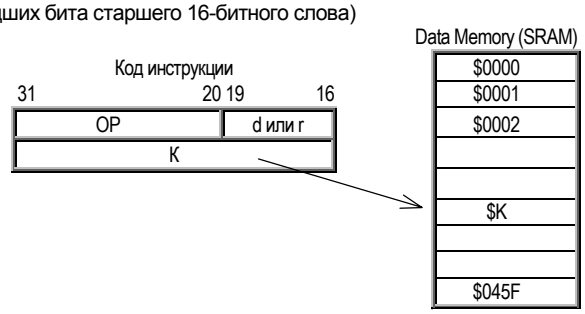


Рис.3.12. Формат команды с прямой адресацией памяти данных

Операции с памятью программ

В инструкции указывается адрес ячейки памяти программ. Обычно таким образом адресуются различные команды переходов и обращения к подпрограммам.

Например,

jmp	k	*Jump*	PC <- k
call	k	*Call Subroutine*	PC <- k

Выполнение программы начинается с адреса, записанного непосредственно в команде

В 32-битном формате команды адрес ячейки памяти программ занимает младшие 16 бит (рис. 3.13).

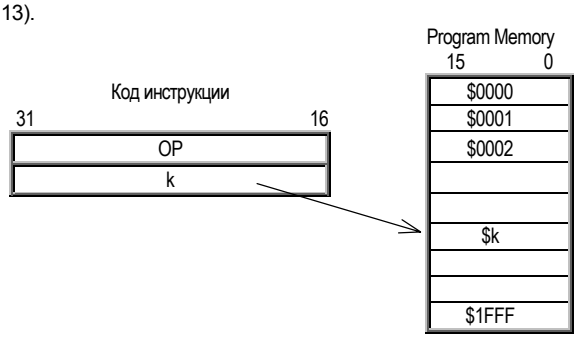


Рис.3.13. Формат команды с прямой адресацией памяти программ

Косвенная адресация

Косвенным образом могут адресоваться ячейки памяти данных или памяти программ.

Операции с памятью данных

Операнд содержится в ячейке памяти данных. Адрес операнда находится в одном из регистров косвенной адресации X, Y или Z.

Например,

```
ld    Rd, X    load indirect     Rd <-  (X)
ld    Rd, Y    load indirect     Rd <-  (Y)
ld    Rd, Z    load indirect     Rd <-  (Z)
```

В 16-битном формате команды адрес ячейки памяти не фигурирует (рис. 3.14).

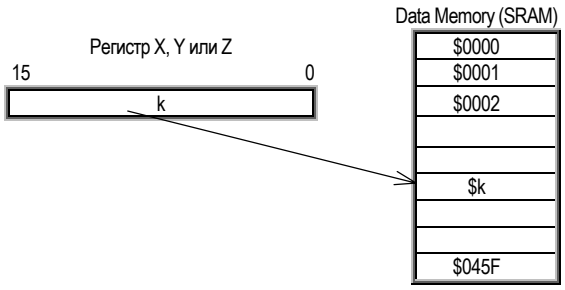


Рис.3.14. Косвенная адресация памяти данных

Адресация памяти данных с постинкрементом

Операнд содержится в ячейке памяти данных. Адрес операнда находится в одном из регистров косвенной адресации X, Y или Z. После выполнения операции регистр косвенной адресации (X, Y или Z) инкрементируется (рис. 3.15).

Например:

```
ld    Rd, X+    load indirect and post-increment     Rd <- (X), X <- X + 1
ld    Rd, Y+    load indirect and post-increment     Rd <- (Y), Y <- Y + 1
```

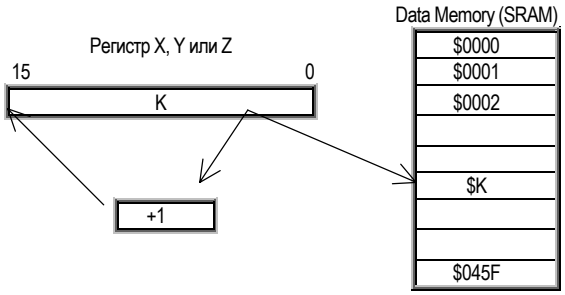


Рис. 3.15. Косвенная адресация памяти данных с постинкрементом

Адресация памяти данных с преддеккрементом

Операнд содержится в ячейке памяти данных. Адрес операнда находится в одном из регистров косвенной адресации X, Y или Z. Перед выполнением операции регистр косвенной адресации (X, Y или Z) декрементируется (рис. 3.16).

Например:

ld Rd, - X *load indirect and pre-decrement* X <- X - 1, Rd <- (X)

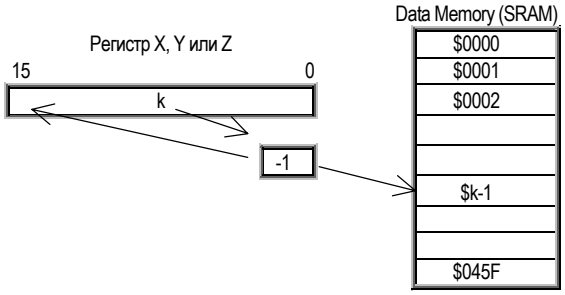


Рис.3.16. Косвенная адресация памяти данных с преддекрементом

Адресация памяти данных со смещением

Операнд содержится в ячейке памяти данных. Адрес операнда вычисляется суммированием содержимого регистра Y или Z с 6 битами адреса, содержащимися в инструкции (рис. 3.17).

Например,

ldd Rd,Y+q load indirect with displacement Rd <- (Y + q)
ldd Rd, Z+q *load indirect with displacement* Rd <- (Z + q)

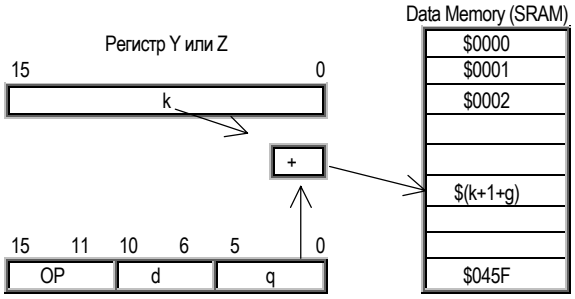


Рис.3.17. Косвенная адресация памяти данных со смещением

Косвенная адресация памяти программ

Адрес константы в памяти программ определяется содержимым регистра Z или выполнение программы продолжается с адреса, записанного в регистре Z (программный счетчик загружается содержимым регистра Z).

Например, команды, обращающиеся за данными к памяти программ:

lpm	load program memory	R1:R0 <- (Z)
spm	*store program memory*	(Z) <- R1:R0

Инструкция SPM, модифицирующая программу может быть использована только в разделе *Boot Program Section* памяти программ (рис.2.1).

Команды переходов

ijmp	indirect jump to (z)	PC <- Z

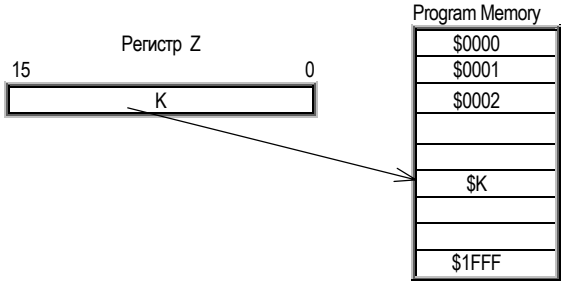


Рис. 3.18. Косвенная адресация памяти программ

Относительная адресация

Относительная адресация используется при обращении к памяти программ. В инструкциях перехода с помощью относительной адресации модифицируется содержимое программного счетчика.

Например,

rjmp	k	*relative jump*	PC <- PC + k + 1
rcall	k	*relative subroutine call*	PC <- PC + k + 1

При этом Выполнение программы продолжается с адреса PC + k + 1. Значение относительного адреса может быть от -2048 до 2047 (рис. 3.19).

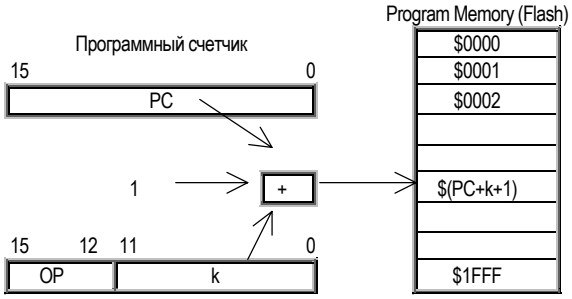


Рис.3.19. Относительная адресация памяти программ

Классификация команд

Множество инструкций микроконтроллера образует систему команд. Система команд, как правило, не меняется для всех микроконтроллеров одного семейства. В полном списке инструкций при описании системы обычно выделяются отдельные группы, родственных по применению инструкций, например, арифметические и логические команды, команды пересылки и др. Это деление довольно условно и полностью определяется политикой разработчика.

При описании системы команд приводится полная информация о каждой инструкции: способах адресации, использовании флагов регистра состояния, времени исполнения.

В качестве примера рассмотрим систему команд микроконтроллера *ATmega 163*.

Система команд микроконтроллера содержит 130 команд, условно разделенных на четыре группы:

- команды пересылки данных (*data transfer instructions*),
- арифметические и логические команды (*arithmetic and logic instructions*),
- команды работы с битами *(bit and bit-test instructions*),
- команды ветвления (*branch instructions*).

При описании системы команд использованы следующие обозначения:

- Rd - регистр-приемник результата ($0 \le d \le 31$),
- Rd* - регистр-приемник результата с номером более 16 ($16 \le d \le 31$),
- Rr - регистр-источник ($0 \le r \le 31$),
- Rdl: регистры R24, R26, R28, R30 (для инструкций ADIW и SBIW),
- P- адрес регистра ввода/вывода,
- P*- адрес побитно адресуемого регистра ввода/вывода ($00-$1F)
- K - символьная или численная константа (8 бит)
- k - адресная константа
- b - номер бита в регистре (3 бита)
- s - номер бита в регистре статуса (3 бита)
- X, Y, Z - регистры косвенной адресации (X=R27:R26, Y=R29:R28; Z=R31:R30

Команды пересылки

Команды пересылки осуществляют перемещение данных между ячейками памяти и регистрами процессорного ядра. Один из операндов, участвующих в инструкции, является источником данных, второй – приемником. При пересылке из источника в приемник копия данных всегда остается в источнике. Таким образом, все команды пересылки практически осуществляют копирование данных.

Одним из операндов в любой команде пересылки обычно является регистр общего назначения процессорного ядра. Вторым может быть любой регистр или ячейка памяти.

Инструкции, как правило, не влияют на флаги в регистре состояния процессорного ядра.

В системе команд микроконтроллеров AVR предусмотрено 34 инструкции, осуществляющие пересылку данных. В инструкциях используются все, за исключением битовых, способы адресации данных; одним из операндов в любой инструкции является регистр общего назначения. Инструкция ldi (*load immediate*), использующая непосредственную адресацию, может быть использована только в старшей половине файла регистров общего назначения (R16….R31), остальные команды работают с любым из регистров файла. Инструкцию spm (*store program memory*), осуществляющую запись

кода в память программ, можно использовать только в *Boot Program Section* памяти программ.

Время выполнения инструкций, работающих с регистрами, равно 1 такту. Инструкции, обращающиеся к ячейкам памяти данных, выполняются за 2 такта, а обращающиеся к ячейкам памяти программ – за 3 такта.

Таблица. 3.1.

Команды пересылки AVR-микроконтроллеров

Мнемо-ника	Операнды	Описание инструкции	Выполняемая операция	Флаги	Такты
mov	Rd, Rr	*Move Between Registers*	Rd <- Rr	None	1
movw	Rd, Rr	*Copy Register Word*	Rd+1:Rd <- Rr+1:Rr	None	1
ldi	Rd*, K	*Load Immediate*	Rd <- K	None	1
ld	Rd, X	*Load Indirect*	Rd <- (X)	None	2
ld	Rd, X+	*Load Indirect and Post-Inc.*	Rd <- (X), X <- X + 1	None	2
ld	Rd, - X	*Load Indirect and Pre-Dec.*	X <- X - 1, Rd <- (X)	None	2
ld	Rd, Y	*Load Indirect*	Rd <- (Y)	None	2
ld	Rd, Y+	*Load Indirect and Post-Inc.*	Rd <- (Y), Y <- Y + 1	None	2
ld	Rd, - Y	*Load Indirect and Pre-Dec.*	Y <- Y - 1, Rd <- (Y)	None	2
ldd	Rd, Y+q	*Load Indirect with Displacement*	Rd <- (Y + q)	None	2
ld	Rd, Z	*Load Indirect*	Rd <- (Z)	None	2
ld	Rd, Z+	*Load Indirect and Post-Inc.*	Rd <- (Z), Z <- Z+1	None	2
ld	Rd, -Z	*Load Indirect and Pre-Dec*	Z <- Z - 1, Rd <- (Z)	None	2
ldd	Rd, Z+q	*Load Indirect with Displacement*	Rd <- (Z + q)	None	2
lds	Rd, k	*Load Direct from SRAM*	Rd <- (k)	None	2
ST	X, Rr	*Store Indirect*	(X) <- Rr	None	2
st	X+, Rr	*Store Indirect and Post-Inc.*	(X) <- Rr, X <- X + 1	None	2
st	- X, Rr	*Store Indirect and Pre-Dec.*	X <- X - 1, (X) <- Rr	None	2
st	Y, Rr	*Store Indirect*	(Y) <- Rr	None	2
st	Y+, Rr	*Store Indirect and Post-Inc.*	(Y) <- Rr, Y <- Y + 1	None	2
st	- Y, Rr	*Store Indirect and Pre-Dec.*	Y <- Y - 1, (Y) <- Rr	None	2
std	Y+q,Rr	*Store Indirect with Displacement*	(Y + q) <- Rr	None	2
st	Z, Rr	*Store Indirect*	(Z) <- Rr	None	2
st	Z+, Rr	*Store Indirect and Post-Inc.*	(Z) <- Rr, Z <- Z + 1	None	2
st	-Z, Rr	*Store Indirect and Pre-Dec*	Z <- Z - 1, (Z) <- Rr	None	2
std	Z+q,Rr	*Store Indirect with Displacement*	(Z + q) <- Rr	None	2
sts	k, Rr	*Store Direct to SRAM*	(k) <- Rr	None	2
lpm		*Load Program Memory*	R0 <- (Z)	None	3
lpm	Rd, Z	*Load Program Memory*	Rd <- Z)	None	3

Окончание табл. 3.1.

Мнемо-ника	Операнды	Описание инструкции	Выполняемая операция	Флаги	Такты
lpm	Rd, Z+	*Load Program Memory and Post-Inc.*	Rd <- Z), Z=Z+1	None	3
spm		*Store Program Memory*	(Z) <- R1:R0	None	-
in	Rd, P	*In Port*	Rd <- P	None	1
out	P, Rr	*Out Port*	P <- Rr	None	1
push	Rr	*Push Register on Stack*	STACK <- Rr; SP<-SP-1	None	2
pop	Rd	*Pop Register from Stack*	SP<- SP+1, Rd <-STACK	None	2

Арифметические и логические команды

В группу арифметических команд входят команды выполняющие сложение, вычитание, декремент и инкремент данных, логическое умножение, логическое сложение, операцию ИСКЛЮЧАЮЩЕЕ ИЛИ, инверсию переменной. Обычно к этой группе относят также инструкции сравнения данных. В микроконтроллере *ATmega163* реализованы также функции арифметического умножения целых чисел и дробных чисел, без знака и со знаком.

Все инструкции этой группы, как правило, приводят к изменению состояния флагов регистра состояния в соответствии с результатами, выполняемой операции.

В микроконтроллерах с архитектурой AVR предусмотрено довольно много (31) инструкций, выполняющих арифметические и логические преобразования данных. Инструкции используют исключительно прямую регистровую или непосредственную адресацию данных. Операнды хранятся в регистрах общего назначения, в один из них (регистр-приемник) всегда направляется и результат вычислений.

В результате выполнения инструкций изменяются флаги регистра SREG (*Status Register)*, а флаг переноса C (*Carry*), кроме того, непосредственно участвует в выполнении ряда операций.

Арифметические команды сложения и вычитания выполняют сложение и вычитание одно- и двухбайтных операндов. Команды adc *(add with carry two registers)* и sbc *(subtract with carry two registers)* используют при вычислениях флаг переноса C.

Инструкции логического умножения (and) и логического сложения (or) , ИСКЛЮЧАЮЩЕЕ ИЛИ (eor*)* преобразуют только однобайтные данные.

Инструкция дополнения до единицы com *(one's complement),* фактически выполняет операцию инверсии, а инструкция дополнения до двух neg (*two's complement*) – меняет знак числа.

Инструкции установки (set) позволяют установить как отдельные, так и все биты выбранного регистра в единичное состояние, а команды очистки (clear) – в нулевое.

Инструкции инкремента регистра inc *(increment*) и декремента регистра dec (*decrement)* используют прямую адресацию одного выбранного регистра.

Тест на нуль или минус tst (*test for zero or minus*) фактически не меняет содержимого регистра, но устанавливает соответствующие флаги при равенстве операнда нулю или при его отрицательном значении.

Команды сравнения (*compare*) также не меняют содержимого регистров, а оценивают разность операндов и устанавливают соответствующие флаги в регистре состояния.

Шесть инструкций умножения (*multiply*) выполняют умножение целых и дробных операндов с учетом и без учета знака. 16-битный результат умножения всегда записывается в регистры общего назначения R0:R1.

Большинство арифметических и логических команд выполняются за один такт. Исключение составляют только команды с непосредственной адресацией, работающие с двухбайтными словами: adiw (*add immediate to word*) и sbiw (*subtract immediate from word*), и команды умножения, выполняющиеся за два такта.

Таблица 3.2.

Арифметические и логические команды микроконтроллера *Atmega163*

Мнемо-ника	*Операнды*	*Описание*	*Операция*	*Флаги*	*Такт*
add	Rd, Rr	*Add without Carry two Registers*	Rd <- Rd + Rr	Z,C,N,V,H	1
adc	Rd, Rr	*Add with Carry two Registers*	Rd <- Rd + Rr + C	Z,C,N,V,H	1
adiw	Rdl, K	*ADD Immediate from Word*	Rdh:Rdl <- Rdh:Rdl+K	Z,C,N,V,S	2
sub	Rd, Rr	*Subtract without Carry two Registers*	Rd <- Rd – Rr	Z,C,N,V,H	1
subi	Rd*, K	*Subtract Constant from Register*	Rd <- Rd – K	Z,C,N,V,H	1
sbc	Rd, Rr	*Subtract with Carry two Registers*	Rd <- Rd – Rr – C	Z,C,N,V,H	1
sbci	Rd*, K	*Subtract with Carry Constant from Register*	Rd <- Rd – K – C	Z,C,N,V,H	1
sbiw	Rdl, K	*Subtract Immediate from Word*	Rdh:Rd <- Rdh:Rdl–K	Z,C,N,V,S	2
and	Rd, Rr	*Logical AND Registers*	Rd <- Rd Rr	Z,N,V	1
andi	Rd*, K	*Logical AND Register and Constant*	Rd <- Rd K	Z,N,V	1
or	Rd, Rr	*Logical OR Registers*	Rd <- Rd v Rr	Z,N,V	1
ori	Rd*, K	*Logical OR Register and Constant*	Rd <- Rd v K	Z,N,V	1
eor	Rd, Rr	*Exclusive OR Registers*	Rd <- Rd Rr	Z,N,V	1
com	Rd	*One's Complement*	Rd <- $FF – Rd	Z,C,N,V	1
neg	Rd	*Two's Complement*	Rd <- $00 – Rd	Z,C,N,V,H	1
sbr	Rd*, K	*Set Bit(s) in Register*	Rd <- Rd v K	Z,N,V	1
ser	Rd	*Set Register*	Rd <- $FF	None	1
cbr	Rd*, K	*Clear Bit(s) in Register*	Rd <- Rd ($FF – K)	Z,N,V	1
clr	Rd	*Clear Register*	Rd <- $00	Z,N,V	1
inc	Rd	*Increment*	Rd <- Rd + 1	Z,N,V	1
dec	Rd	*Decrement*	Rd <- Rd –1	Z,N,V	1

Окончание табл. 3.2.

Мнемо-ника	*Операнды*	*Описание*	*Операция*	*Флаги*	*Такт*
tst	Rd	*Test for Zero or Minus*	Rd <- Rd Rd	Z,N,V	1
cp	Rd, Rr	*Compare*	Rd – Rr	Z, N,V,C,H	1
cpc	Rd, Rr	*Compare with Carry*	Rd – Rr – C	Z, N,V,C,H	1
cpi	Rd*, K	*Compare Register with Immediate*	Rd – K	Z, N,V,C,H	1
mul	Rd, Rr	*Multiply Unsigned*	R1:R0 <- Rd x Rr	Z,C	2
muls	Rd, Rr	*Multiply Signed*	R1:R0 <- Rd x Rr	Z,C	2
mulsu	Rd, Rr	*Multiply Signed with Unsigned*	R1:R0 <- Rd x Rr	Z,C	2
fmul	Rd, Rr	*Fractional Multiply Unsigned*	R1:R0<-(Rd x Rr)<<1	Z,C	2
fmuls	Rd, Rr	*Fractional Multiply Signed*	R1:R0<-(Rd x Rr)<<1	Z,C	2
fmulsu	Rd, Rr	*Fractional Multiply Signed with Unsigned*	R1:R0<-(Rd x Rr)<<1	Z,C	2

Битовые команды

Битовые команды *(Bit And Bit-Test Instructions)* позволяют обращаться непосредственно к отдельным битам регистров процессорного ядра и выполнять с выбранными битами простейшие операции: пересылки, установки и сброса. Операндами команд могут быть как биты регистров общего назначения, так и биты регистров ввода/вывода. Битовые команды могут влиять на отдельные флаги регистра признаков.

Битовые команды микроконтроллеров семейства AVR работают с отдельными битами регистров общего назначения и регистров ввода/вывода. Биты регистров ввода/вывода могут быть установлены или сброшены. Биты регистров общего назначения могут быть сдвинуты в соседние ячейки, как влево (в сторону старших разрядов), так и вправо (в сторону младших разрядов). В операциях сдвига, кроме разрядов регистров, может участвовать и бит переноса. Специальные команды предусмотрены для установки и сброса отдельных флагов регистра состояния. Команда swap меняет местами тетрады (полубайты) регистров общего назначения.

Таблица 3.3.

Битовые команды микроконтроллера *ATmega163*

Мнемо-ника	*Операнды*	Описание	*Операция*	*Флаги*	*Такт*
sbi	P*,b	*Set Bit in I/O Register*	I/O(P,b) <- 1	None	2
cbi	P*,b	*Clear Bit in I/O Register*	I/O(P,b) <- 0	None	2
lsl	Rd	*Logical Shift Left*	Rd(n+1) <- Rd(n), Rd(0) <- 0	Z,C,N,V	1
lsr	Rd	*Logical Shift Right*	Rd(n) <- Rd(n+1), Rd(7) <- 0	Z,C,N,V	1
rol	Rd	*Rotate Left Through Carry*	Rd(0) <- C, Rd(n+1) <- Rd(n), C <- Rd(7)	Z,C,N,V	1

Окончание табл.3.3.

Мнемо-ника	*Операнды*	Описание	*Операция*	*Флаги*	*Такт*
ror	Rd	*Rotate Right Through Carry*	Rd(7) <- C,Rd(n)<- Rd(n+1),C <- Rd(0)	Z,C,N,V	1
asr	Rd	*Arithmetic Shift Right*	Rd(n) <- Rd(n+1), n=0..6	Z,C,N,V	1
swap	Rd	*Swap Nibbles*	Rd(3..0) <- Rd(7..4), Rd(7..4) <- Rd(3..0)	None	1
bset	s	*Flag Set*	SREG(s) <- 1 SREG(s)		1
bclr	s	*Flag Clear*	SREG(s) <- 0 SREG(s)		1
bld	Rd, b	*Bit load from T to Register*	Rd(b) <- T	None	1
bst	Rr, b	*Bit Store from Register to T*	T <- Rr(b)	T	1
sec		*Set Carry*	C <- 1	C	1
clc		*Clear Carry*	C <- 0	C	1
sen		*Set Negative Flag*	N <- 1	N	1
cln		*Clear Negative Flag*	N <- 0	N	1
sez		*Set Zero Flag*	Z <- 1	Z	1
clz		*Clear Zero Flag*	Z <- 0	Z	1
sei		*Global Interrupt Enable*	I <- 1	I	1
cli		*Global Interrupt Disable*	I <- 0	I	1
ses		*Set Signed Test Flag*	S <- 1	S	1
cls		*Clear Signed Test Flag*	S <- 0	S	1
sev		*Set Twos Complement Overflow*	V <- 1	V	1
clv		*Clear Twos Complement Overflow*	V <- 0	V	1
set		*Set T in SREG*	T <- 1	T	1
clt		*Clear T in SREG*	T <- 0	T	1
she		*Set Half Carry Flag in SREG*	H <- 1	H	1
clh		*Clear Half Carry Flag in SREG*	H <- 0	H	1

Инструкции ветвления

Инструкции ветвления изменяют содержимое программного счетчика. Они используются при организации переходов и подпрограмм. Инструкции условных переходов и условных вызовов подпрограмм в процессе исполнения проверяют выполнение различных условий. Если условие не выполняется, программный счетчик просто

инкрементируется. Такими условиями может быть равенство содержимого двух регистров, единичное или нулевое состояние заданных битов в регистрах общего назначения, регистрах ввода/вывода или в регистре состояния, Например, команда cpse (*compare, skip if equal*) проверяет равенство содержимого регистров, команда sbrc Rr, b (*skip if bit in register cleared*) осуществляет переход при условии равенства нулю содержимого бита b в регистре Rr, а инструкция brmi k *(branch if minus)* проверяет флаг отрицательного результата (N) в регистре состояния SREG.

Время выполнения команд ветвления зависит от результата проверки. Оно, для различных инструкций, может меняться, в пределах от одного до четырех тактов.

Таблица 3.4.

Команды переходов микроконтроллера *ATmega163*

Мнемо-ника	Операнды	Описание	Операция	Флаги	Такт
rjmp	k	*Relative Jump*	PC <- PC + k + 1	None	2
ijmp		*Indirect Jump to (Z)*	PC <- Z	None	2
jmp	k	*Jump*	PC <- k	None	3
rcall	k	*Relative Subroutine Call*	PC <- PC + k + 1	None	3
call	k	*Call Subroutine*	PC <- k	None	4
icall		*Indirect Call to (Z)*	PC <- Z	None	3
ret		*Subroutine Return*	PC <- STACK	None	4
reti		*Interrupt Return*	PC <- STACK	I	4
cpse	Rd,Rr	*Compare, Skip if Equal*	if (Rd = Rr) PC <- PC + 2 or 3	None	1/ 2 / 3
sbrc	Rr, b	*Skip if Bit in Register Cleared*	if (Rr(b)=0) PC <- PC + 2 or 3	None	1 / 2
sbrs	Rr, b	*Skip if Bit in Register is Set*	if (Rr(b)=1) PC <- PC + 2 or 3	None	1 / 2
sbic	P*, b	*Skip if Bit in I/O Register Cleared*	if (P(b)=0) PC <- PC + 2 or 3	None	1 / 2
sbis	P*, b	*Skip if Bit in I/O Register is Set*	if (P(b)=1) PC <- PC + 2 or 3	None	1 / 2
brbs	s, k	*Branch if Status Flag Set*	if (SREG(s) = 1) then PC <- PC+k + 1	None	1 / 2
brbc	s, k	*Branch if Status Flag Cleared*	if(SREG(s) = 0) then PC <- PC+k + 1	None	1 / 2
breq	k	*Branch if Equal*	if (Z = 1) then PC <- PC + k + 1	None	1 / 2
brcs	k	*Branch if Carry Set*	if (C = 1) then PC <- PC + k + 1	None	1 / 2
brne	k	*Branch if Not Equal*	if (Z = 0) then PC <- PC + k + 1	None	1 / 2
brcc	k	*Branch if Carry Cleared*	if (C = 0) then PC <- PC + k + 1	None	1 / 2

Окончание табл. 3.4.

Мнемо-ника	Операнды	Описание	Операция	Флаги	Такт
brsh	k	*Branch if Same or Higher*	if (C = 0) then PC <- PC + k + 1	None	1 / 2
brlo	k	*Branch if Lower*	if (C = 1) then PC <- PC + k + 1	None	1 / 2
brmi	k	*Branch if Minus*	if (N = 1) then PC <- PC + k + 1	None	1 / 2
brpl	k	*Branch if Plus*	if (N = 0) then PC <- PC + k + 1	None	1 / 2
brge	k	*Branch if Greater or Equal, Signed*	if (N ⊕ V= 0) then PC <- PC + k + 1	None	1 / 2
brlt	k	*Branch if Less Than Zero, Signed*	if (N ⊕ V= 1) then PC <- PC + k + 1	None	1 / 2
brhs	k	*Branch if Half Carry Flag Set*	if (H = 1) then PC <- PC + k + 1	None	1 / 2
brhc	k	*Branch if Half Carry Flag Cleared*	if (H = 0) then PC <- PC + k + 1	None	
brts	k	*Branch if T Flag Set*	if (T = 1) then PC <- PC + k + 1	None	1 / 2
brtc	k	*Branch if T Flag Cleared*	if (T = 0) then PC <- PC + k + 1	None	1 / 2
brvs	k	*Branch if Overflow Flag is Set*	if (V = 1) then PC <- PC + k + 1	None	1 / 2
brvc	k	*Branch if Overflow Flag is Cleared*	if (V = 0) then PC <- PC + k + 1	None	1 / 2
brie	k	*Branch if Interrupt Enabled*	if (I = 1) then PC <- PC + k + 1	None	1 / 2
brid	k	*Branch if Interrupt Disabled*	if (I = 0) then PC <- PC + k + 1	None	1 / 2

Инструкции управления

Инструкции управления используются для управления вычислениями и режимами работы процессорного ядра. Пустая операция nop заполняет пробелы между инструкциями в программной памяти или используется в качестве выдержки. Команда sleep переводит процессорное ядро в режим пониженного энергопотребления. Инструкция wdr сбрасывает сторожевой таймер.

Таблица 3.5.

Инструкции управления микроконтроллера *ATmega163*.

Мнемоника	Описание	Операция	Такт
nop	*No Operation*	None	1
sleep	*Sleep (see specific description)*	None	3
wdr	*Watchdog Reset*	None	1

4. ЯЗЫК АССЕМБЛЕРА

Язык ассемблера (*assembler language*) – язык программирования микропроцессорных систем, ориентированный на определенную архитектуру системы. Программа, написанная на языке ассемблера, переводится в машинные коды с помощью специального компилятора.

Язык ассемблера использует систему команд процессорного ядра и специальные директивы, указывающие программе ассемблеру, как организовать различные секции программы, где располагать данные, как связать отдельные процедуры и т. д. Из-за множества архитектурных отличий различных систем единого языка ассемблера не существует. Каждый разработчик создает свой язык и свое программное обеспечение для работы с ним. Все примеры далее написаны на языке ассемблера Atmel AVR Assembler.

Компилятор транслирует исходные коды с языка ассемблера в объектный код. Полученный объектный может быть непосредственно запрограммирован в микроконтроллеры AVR.

4.1. Выражения

Программа на языке ассемблера состоит из отдельных строк. Строка кода не должна быть длиннее 120 символов. Ассемблер Atmel AVR не различает строчные и заглавные буквы.

Любая строка может начинаться с метки, которая является набором символов, заканчивающимся двоеточием. Метки используются для указания места, в которое передаётся управление при переходах, а также для задания имён переменных.

Входная строка может иметь одну из четырёх форм:

- [метка:] директива [операнды] [Комментарий]
- [метка:] инструкция [операнды] [Комментарий]
- Комментарий
- Пустая строка

Комментарий имеет следующую форму:

- ; [Текст]

Позиции в квадратных скобках необязательны. Текст после точки с запятой (;) и до конца строки игнорируется компилятором. Метки, инструкции и директивы более детально описываются ниже.

Примеры:

```
label: .equ var1=100     ; Устанавливает var1 равным 100 (директива)
.equ var2=200            ; Устанавливает var2 равным 200
test: rjmp test          ; Бесконечный цикл (инструкция)
  ; Строка с одним только комментарием
  ; Ещё одна строка с комментарием
```

Компилятор не требует, чтобы метки, директивы, комментарии или инструкции находились в определённой колонке строки.

Команды микроконтроллера и директивы языка ассемблера оперируют выражениями. Выражением считается набор: операндов (*operands*), связанных между собой операторами (*operators*) и функциями (*functions*).

ОПЕРАНДЫ

Операндами языка ассемблера могут быть:

- Определяемые пользователем метки.

  Метка может располагаться перед командой/директивой или входить в директиву.

  Если метка располагается перед командой (или перед директивой) микроконтроллера, то после неё ставится символ : (двоеточие). Двоеточие указывает ассемблеру, что метка задает состояние программного счетчика в отмеченном месте программы.

  Например,

  Lab23: mov r5,r7 ; строка в программе отмечена меткой Lab23

  Если метка входит в директиву то, она рассматривается как один из операндов этой директивы и двоеточием не отмечается.

  Например,

  .set pina = $19 ; метка pina с помощью директивы .set связывается с числом $19

- Определяемые пользователем с помощью директивы set переменные.

  Директива .set связывает метку и переменную. Эта метка может использоваться далее в программе вместо переменной. Метка, указывающая на переменную в соответствии с директивой set, может быть впоследствии изменена.

  Например,

  .set pina = $19 ; метка pina связывается с числом $19

  .set porta = pina + 2 ; метка porta связывается с числом $19+2=$1B

  ……

  out porta,r2 Пересылка данных из регистра r2 в porta (по адресу $1B)

- Определяемые пользователем с помощью директивы equ константы.

  Директива equ связывает метку с константой. Эта метка может использоваться далее в программе. Метка, указывающая на константу в соответствии с директивой set, не может быть впоследствии изменена.

  Например,

  .equ pina = $19 ; метка pina связывается с числом $19

  .equ porta = pina + 2 ; метка porta связывается с числом $19+2=$1B

  ……

  out porta,r2 ; пересылка данных из регистра r2 в porta (по ;адресу $1B)

- Целые константы; заданные в одном из следующих форматов:
  - десятичный (*decimal*), например, 255;
  - шестнадцатеричный (*hexadecima*l) начинается с символа $ или 0x, например, $0a или 0x0a;
  - двоичный (*binary*) – начинается с символов 0b, например, 0b00001010;
  - восьмеричный *(octal*) начинается с символа 0, например, 077.

ОПЕРАТОРЫ

Все операторы ассемблера перечислены в таблице. Порядок выполнения операторов в выражении определяется приоритетом. Операторы с большим приоритетом выполняются в первую очередь. Кроме того, отдельные выражения могут быть заключены в круглые скобки. Такие выражения также вычисляются в первую очередь.

Таблица 4.1.

Операторы языка Atmel AVR Assembler

Символ	Описание		Приоритет
!	*Logical Not*	Логическое НЕ	14
~	*Bitwise Not*	Побитное НЕ	14
-	*Unary Minus*	Смена знака	14
*	*Multiplication*	Умножение	13
/	*Division*	Деление	13
+	*Addition*	Сложение	12
-	*Subtraction*	Вычитание	12
<<	*Shift left*	Сдвиг влево	11
>>	*Shift right*	Сдвиг вправо	11
<	*Less than*	Меньше	10
<=	*Less than or equal*	Меньше или равно	10
>	*Greater than*	Больше	10
>=	*Greater than or equal*	Больше или равно	10
==	*Equal*	Равно	9
!=	*Not equal*	Не равно	9
&	*Bitwise And*	Побитное И	8
^	*Bitwise Xor*	Побитное ИСКЛЮЧАЮЩЕЕ ИЛИ	7
|	*Bitwise Or*	Побитное ИЛИ	6
&&	*Logical And*	Логическое И	5
||	*Logical Or*	Логическое ИЛИ	4

Логическое НЕ

Оператор | возвращает 1, если выражение равно 0, в противном случае возвращает 0

Например: ldi r16,!$f0 ; в регистр r16 пересылается $00

Побитное НЕ

Оператор ~ возвращает побитную инверсию выражения.

Например: ldi r16,~$f0 ; в регистр r16 пересылается $0f

Смена знака

Оператор - возвращает отрицание арифметического выражения

Например: ldi r16,-2 ; пересылается число –2 ($fe) в регистр r16

Умножение

Оператор * возвращает произведение выражений

Например: ldi r30,label*2 ; пересылается в регистр r30 произведение label*2

Деление

Оператор / возвращает целую часть от деления левой части выражения на правую.

Наример: ldi r30,label/2 ; пересылается в регистр r30 целая часть выражения label/2 .

Сложение

Оператор + возвращает сумму двух выражений

Пример: ldi r30,c1+c2 ; пересылается в регистр r30 сумма c1+c2

Вычитание

Оператор - возвращает разность двух выражений.

Например: ldi r17,c1-c2 ;пересылается в r17 разность c1-c2

Сдвиг влево

Оператор << возвращает левое выражение сдвинутое влево на количество бит, указанных в правом выражении.

Например: ldi r17,1<<bitmask ; Пересылается в регистр r17 число 1 сдвинутое влево на bitmask бит.

Сдвиг вправо

Оператор >> возвращает выражение в левой части сдвинутое вправо на количество бит, указанных в правой части.

Например: ldi r17,c1>>c2 ;пересылается в регистр r17 число c1 сдвинутое вправо на c2 бит.

Меньше

Оператор < возвращает 1, если выражение в левой части меньше, чем выражение в правой; 0 – в противном случае

Например: ori r18,bitmask*(c1<c2)+1 ;выполняется логическое ИЛИ регистра r18 с результатом вычислений bitmask*(c1<c2)+1

Меньше или равно

Оператор <= возвращает 1, если выражение в левой части меньше или равно, чем выражение в правой части; 0 – в противном случае

Например: ori r18,bitmask*(c1<=c2)+1 ; логическое ИЛИ r18 с результатом вычисления выражения.

Больше

Оператор > возвращает 1, если выражение со знаком слева больше, чем выражение со знаком справа; 0 – в противном случае.

Например: ori r18,bitmask*(c1>c2)+1 ; логическое ИЛИ r18 с выражением .

Больше или равно

Оператор >= возвращает 1, если выражение в левой части больше или равно, выражению в правой; 0 - в противном случае..

Например: ori r18,bitmask*(c1>=c2)+1 ; логическое ИЛИ r18 с выражением.

Равно

Оператор == возвращает 1, если выражение в левой части равно, выражению в правой; 0 - в противном случае..

Например: andi r19,bitmask*(c1==c2)+1 ; логическое И r19 с выражением

Не равно

Оператор |= возвращает 1, если выражение в левой части не равно, выражению в правой; 0 - в противном случае..

Например: .set flag=(c1|=c2) ; установка переменной flag

Побитное И

Оператор & возвращается поразрядное логическое И между двумя выражениями

Например: ldi r18,High(c1&c2) ; пересылка в r18 выражения

Побитное ИСКЛЮЧАЮЩЕЕ ИЛИ

Оператор ^ возвращается поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ между двумя выражениями

Например: ldi r18,Low(c1^c2) ;пересылка в r18 выражения

Побитное ИЛИ

Оператор | возвращается поразрядное ИЛИ между двумя выражениями

Пример: ldi r18,Low(c1|c2) ; пересылка в r18 выражения

Логическое И

Оператор && возвращается 1, если оба выражения отличны от нуля; 0 – в остальных случаях

Например: ldi r18,Low(c1&&c2) ; пересылка в r18 выражения

Логическое ИЛИ

Оператор || возвращает 1, если один или оба из выражения отличны от нуля, 0 – в остальных случаях.

Например: ldi r18,Low(c1||c2) ; пересылка в r18 выражения

ФУНКЦИИ

Функции, определенные в языке ассемблера

- low(expression) возвращает младший байт выражения;
- high(expression) возвращает второй байт выражения;
- byte2(expression) то же что high;
- byte3(expression) возвращает третий байт выражения;
- byte4(expression) возвращает четвертый байт выражения;
- lwrd(expression) возвращает биты 0-15 выражения;
- hwrd(expression) возвращает биты 16-31 выражения;
- page(expression) возвращает биты 16-21 выражения;
- exp2(expression) возвращает 2 в степени, определяемой выражением в скобках;
- log2(expression) возвращает целую часть вычислений log2 от выражения в скобках.

4.2. Директивы

Директивы ассемблера не транслируются в коды операций, они используются для размещения программы в памяти, определяют макрокоманды, инициализируют память данных и выполняют ещё целый ряд различных вспомогательных операций, облегчающих процесс программирования. Полный список директив ассемблера AVR приведен в таблице 4.2.

Таблица 4.2.

Директивы языка Atmel AVR Assembler

Директивы	Мнемоника	Наименование
byte	*Reserve byte to a variable*	Резервирование байта для переменной
cseg	*Code Segment*	Сегмент памяти программ
db	*Define constant byte(s)*	Определение однобайтной константы
def	*Define a symbolic name on a register*	Определение символического названия регистра
device	*Define which device to assemble for*	Определение устройства
dseg	*Data Segment*	Сегмент памяти данных
dw	*Define Constant word(s)*	Определение слова
endmacro	*End macro*	Конец макроса
equ	*Set a symbol equal to an expression*	Задание имени константы
eseg	*EEPROM Segment*	Сегмент EEPROM
exit	*Exit from file*	Конец файла
include	*Read source from another file*	Подключение файла

list	*Turn listfile generation on*	Включение генерации листинга
listmac	*Turn Macro expansion in list file on*	Включение макроса в листинг
nolist	*Turn listfile generation off*	Отключение генерации листинга
org	*Set program origin*	Задание начального адреса памяти
set	*Set a symbol to an expression*	Задание имени выражения

Примечание: Любой директиве в тексте программы предшествует точка.

Выбор микроконтроллера

device

Директива device (*Define which device to assemble for*) сообщает ассемблеру о используемом контроллере. При использовании этой директивы генерируется предупреждение, если команда в программе не поддерживается точно выбранным контроллером. Если сегмент программы или сегмент EEPROM в программе больше, чем в выбранном устройстве, ассемблер также формирует предупреждение. Если директива не используется, то считается, что все команды ассемблера поддержаны и нет никаких ограничений на EEPROM и память программ.

Синтаксис:

.device at90s1200 |at90s2313 | at90s2323 | at90s2333 | at90s2343 | at90s4414 | at90s4433 | at90s4434 | at90s8515 | at90s8534 | at90s8535 | attiny11 | attiny12 | attiny22 | atmega603 | atmega103 | atmega163

Например,

```
.device atmega163       ; выбираем atmega163
```

Организация и резервирование памяти

Программа на языке ассемблера обычно состоит из нескольких сегментов. Область памяти, в которой размещаются программы, называют сегментом кода, область памяти данных – сегментом данных, а область энергонезависимой памяти данных – сегментом EEPROM. Сегмент кода описывается директивой CSEG, сегмент данных – директивой DSEG, а сегмент EEPROM – ESEG.

cseg

Директива cseg *(Code segment*) определяет начало сегмента кода. Файл ассемблера может состоять нескольких сегментов, которые связаны в одну программу. В пределах сегмента кода cseg нельзя использовать директиву byte. Для размещения программ в сегменте кода используется директива org (см. далее). Директива cseg не имеет параметров.

Синтаксис:

```
.cseg
```

Например,

```
.dseg                   ; начало сегмента данных
```

```
vartab: .byte 4          ; резервируется 4 байта в sram
.cseg                    ; начало сегмента кода
const: .dw 2             ; запись $0002 в память программ
mov r1,r0                ; пересылка
```

dseg

Директива dseg (*Data segment*) определяет начало сегмента данных. Файл ассемблера может содержать несколько сегментов данных. Сегмент данных обычно состоит только из директив byte. Для размещения переменных в определенных байтах SRAM может использоваться директива org. Директива не имеет никаких параметров.

Синтаксис:

```
.dseg
```

Например,

```
.dseg                    ; начало сегмента данных
var1:  .byte 1           ; резервируем 1 байт по адресу  var1
table:  .byte tab_size   ; резервируем tab_size бай.
.cseg
ldi r30,low(var1)        ; запись в младший байт регистра z
ldi r31,high(var1)       ; запись в старший байт регистра z
ld r1,z                  ; пересылка из  var1 в r1
```

eseg -

Директива eseg (*EEPROM Segment*) определяет начало сегмента EEPROM. Файл ассемблера может содержать несколько сегментов EEPROM. Сегмент EEPROM обычно состоит только из директив db и dw. Для размещения данных в сегменте EEPROM может использоваться директива org. Директива eseg не имеет параметров.

Синтаксис:

```
.eseg
```

Например,

```
.dseg                    ; начало сегмента данных
var1:   .byte 1          ; резервируется 1 байт для var1
table:  .byte tab_size   ; резервируется tab_size байт
.eseg
eevar1: .dw 0xffff       ; инициализируется 1 слово в eeprom
```

byte

Директива byte (*Reserve bytes to a variable*) резервирует байты в памяти данных (SRAM). Директиве должна предшествовать метка.. Директива имеет один параметр, который указывает на число резервируемых байт. Директива может быть использована только пределах сегмента данных (см. директиву dseg).

Синтаксис:

```
label: .byte expression
```

Например,

```
.dseg
var1:   .byte 1          ; резервируется 1 байт для var1
table:  .byte tab_size   ; резервируется tab_size байт
.cseg
ldi r30,low(var1)        ; загрузка младшего байта регистра z
ldi r31,high(var1)       ; загрузка старшего байта регистра z
ld r1,z                  ; загрузка var1 из регистра 1
```

Определение переменных

Директивы определения переменных позволяют включить данные в программу. Они указывают ассемблеру, что в соответствующем месте памяти программ или EEPROM располагается переменная, устанавливают её тип (байт или двухбайтное слово), задают начальные значения и ставят в соответствие переменным метки, которые впоследствии можно использовать при обращении к данным.

db

Директива db (*Define constant byte(s) in program memory and EEPROM*) определяет константы в памяти программ или EEPROM. Директиве db должна предшествовать метка. Директива db использует список выражений, содержащий, по крайней мере, одно выражение. Директива db должна быть размещена в сегменте кода или в сегменте EEPROM.

Список выражения - последовательность выражений, разграниченных запятыми. Каждое выражение должно иметь значение в пределах –128….255. Если выражение является отрицательным числом, оно будет представлено в дополнительном коде.

Если директива db расположена в сегменте программы cseg и список выражений содержит более чем одно выражение, то выражения упаковываются по два байта в каждом слове памяти программ. Если список выражений содержит нечетное число выражений, последнее выражение займет слово в памяти программ, даже если следующая строка в коде ассемблера снова содержит db директиву.

Синтаксис:

```
label:  .db expressionlist
```

Например,

```
.cseg
consts: .db 0, 255, 0b01010101, -128, $aa
.eseg
const2: .db 1,2,3
```

dw

Директива dw (*Define constant word(s) in program memory and EEPROM*) резервирует слово (2 байта) в памяти программ или EEPROM. Директиве должна предшествовать метка. Директива использует список выражений, состоящий не менее чем из одного выражения. Директива dw должна быть размещена в сегменте кода или в сегменте EEPROM.

Список выражений - последовательность выражений, разграниченных запятыми. Каждое выражение должно находится в пределах от -32768 до 65535. Отрицательные числа записываются в дополнительном коде.

Синтаксис:

label: .dw expressionlist

Например,

```
.cseg
varlist:  .dw 0, 0xffff, 0b1001110001010101, -32768, 65535
.eseg
eevarlst: .dw 0,0xffff,10
```

def

Директива def (*Set a symbolic name on a register*) присваивает символические имена регистрам. Один регистр может иметь несколько символических имен. Символическое имя регистра может быть переопределено позже в программе.

Синтаксис:

.def symbol=register

Например,

```
.def temp=r16
.def ior=r0
.cseg
ldi temp,0xf0          ; запись $f0 в регистр temp
in ior,$3f             ;чтение регистра статуса sreg в регистр ior
eor temp,ior           ; исключающее ИЛИ регистров temp и ior
```

equ

Директива equ (*Set a symbol equal to an expression*) связывает константу с меткой. Эта метка может использоваться далее в программе. Метка, указывающая на константу в соответствии с директивой set, не может быть впоследствии переопределена.

Синтаксис

.equ label = expression

Директива .equ обычно используется для присвоения символических имен всем регистрам микроконтроллера, отдельным битам регистров и ячеек памяти, как это сделано, например, в приложении 1 в файле описания файле ATmega163def.inc.

set

Директива set (*Set a symbol equal to an expression*) присваивает имя выражению. Это имя может использоваться далее в программе. Имя, присвоенное выражению в соответствии с директивой set, может быть впоследствии изменено.

Синтаксис

.set label = expression

Например,

```
.set pina = $19        ; имя pina присваивается числу $19
```

```
.set porta = pina + 2      ; имя porta присваивается выражению $19+2=$1b
……
out porta,r2               ;Пересылка данных из регистра r2 в porta (по адресу $1B)
```

Управление файлами

include

Директива include *(Include another file*) вставляет в текст программы заданный файл. Включенный файл может также содержать директивы include.

Синтаксис:

.include "filename"

Например,

.include “m163def.inc “ ;подключаем файл описания микроконтроллера *ATmega163*

Макросы

Макросом называется участок программы, которому присвоено имя и который ассемблируется всякий раз, когда ассемблер встречает это имя в тексте программы. Макрос начинается директивой macro и заканчивается директивой endmacro.

macro

Директива macro (*begin macro*) сообщает ассемблеру о начале макроса. Параметром директивы является название макроса. Макрос может содержать до 10 параметров. Эти параметры упоминаются как @0 - @9 в пределах макроопределения. При вызове макроса параметры перечисляются списком через запятую.

По умолчанию в листинге программы показывается только вызов макроса. Чтобы включить в листинг макроопределение необходимо использовать директиву LISTMAC. Макроопределения отмечаются в поле кода операции знаком «+».

Синтаксис:

.macro macroname

Например,

```
.macro subi16          ; начало определения мароса
subi @1,low(@0)        ; вычитание младшего байта
sbci @2,high(@0)       ; вычитание старшего байта
.endmacro              ; конец макроопределения
.cseg                  ; начало сегмента кода
subi16 0x1234,r16,r17  ; вычитание 0x1234 из r17:r16
```

endmacro

Директива endmacro *(End macro*) определяет конец макроса. Директива не имеет никаких параметров.

Синтаксис:

.endmacro

Например,

```
.macro subi16          ; начало определения макроса
subi r16,low(@0)       ; вычитание low byte
sbci r17,high(@0)      ; вычитание high byte
.endmacro
```

Управление листингом

list

Директива list (*Turn the list file generation on)* сообщает ассемблеру о необходимости генерации листинга программы. Листинг – это файл, содержащий исходный текст программы, адреса и коды инструкций. По умолчанию генерация листинга считается включенной. Вместе с директивой NOLIST она позволяет генерировать листинги отдельных частей исходного файла.

Синтаксис:

```
.list
```

Например,

```
.nolist                ; отключение генерации листинга
.include "macro.inc"   ; подключение файла
.include "const.def    ; подключение файла
.list                  ; включение генерации листинга
```

listmac

Директива listmac (*Turn macro expansion on*) включает в листинг программы текст макроса. По умолчанию в листинге показывается только вызов макроса с параметрами.

Синтаксис:

```
.listmac
```

Например,

```
.macro macx            ; определение макроса
add  r0,@0
eor  r1,@1
.endmacro              ; конец определения
.listmac               ; включение макроса в листинг программы
macx r2,r1             ; выполнение макроса
```

nolist

Директива nolist (*Turn listfile generation off*) выключает генерацию листинга. По умолчанию генерация листинга считается включенной. Директива может использоваться также вместе с директивой LIST для генерации листинга отдельных частей программы.

Синтаксис:

```
.NOLIST
```

Например,

```
.NOLIST                ; отключение генерации листинга программы
.INCLUDE "macro.inc    ; подключение файла
```

```
.INCLUDE "const.def"   ; подключение файла
.LIST                  ; включение генерации листинга
```

Установка адресных счетчиков

ORG

Директива ORG (Set program origin) устанавливает счетчик адреса на определенное значение. Устанавливаемое значение задается как параметр. Если директива ORG записана в сегменте данных, то она устанавливает счетчик адреса SRAM. если в сегменте кода – то счетчик адреса программы, если в сегменте EEPROM – то счетчик адреса EEPROM. Если директиве предшествует метка на той же самой строке исходного текста, nj метке будет присвоено значение параметра. По умолчанию значения счетчиков адреса программы и EEPROM нулевые, а счетчика адреса данных - 32 (после регистров, занимающим адреса 0-31).

Синтаксис:

```
.org expression
```

Например,

```
.dseg                  ; сегмент данных
.org 0x37              ; адрес $37
variable: .byte 1      ; резервирование байта
.cseg                  ; сегмент кода
.org 0x10              ; установка программного счетчика на адрес $10
mov r0,r1               ;
```

Выход

exit

Директива exit (*Exit this file*) прекращает трансляцию файла. Обычно ассемблер выполняет трансляцию до конца файла. Если директива exit появляется в подключенном файле ассемблирование продолжается от строки, следующей за директивой include.

Синтаксис:

```
.exit
```

Например,

```
.exit          ; конец трансляции
```

5. ТАКТОВЫЙ ГЕНЕРАТОР

Работа процессорного ядра синхронизируется тактовым генератором. Именно период работы генератора определяет время, необходимое для выполнения элементарных операций в ядре. Простейшие операции, как правило, выполняются за один такт. Операции, связанные с извлечением данных из памяти, выполняются за два-три такта.

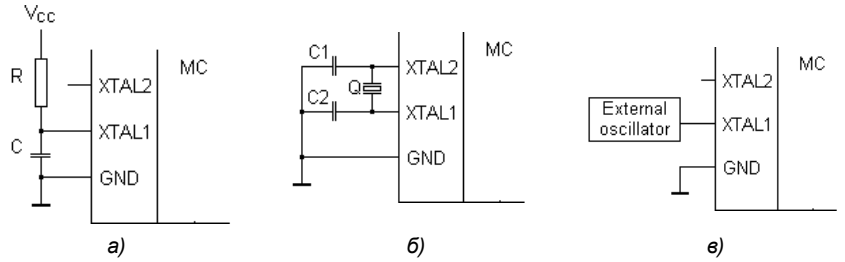
Схемы тактовых генераторов могут быть различны.

Простейшим и наиболее дешевым считается внутренний (реализованный на кристалле) RC-генератор. Он строится без внешних компонентов, но на частоту колебаний

такого генератора влияет напряжение питания и температура. Для повышения стабильности к схеме внутреннего тактового генератора подсоединяются внешние высокостабильные времязадающие компоненты: кварцевые или керамические резонаторы. В ряде случаев микроконтроллеры используются с внешними схемами тактовых генераторов. Это также обеспечивает высокую точность отсчета времени и, при необходимости, позволяет синхронизировать работу сразу нескольких устройств. Некоторые схемы, могут работать с различными генераторами. Так, например, микроконтроллер *ATmega163* может использоваться с различными внутренними и внешними генераторами.

Внутренний RC-генератор обеспечивает работу процессорного ядра AVR с частотой 1 МГц. Какие либо навесные компоненты в этом режиме к микросхеме не подсоединяются.

Внешний RC-генератор является простым решением, позволяющим изменять тактовую частоту. Но частота такого RC-генератора, как и генератора внутреннего, зависит от напряжения питания и температуры. Кроме того, частота может быть различной у различных изделий. Внешняя времязадающая RC-цепь генератора подключается к выводам микроконтроллера по схеме рис. 5.1 (а).



а) *б)* *в)*

Рис. 5.1. Подключение RC-цепей, резонаторов и внешних генераторов к контроллеру ATmega163

Сопротивление резистора R допускается варьировать в диапазоне от 3 до 100 кОм, а емкость С не должна быть менее 20 пФ. Типовые значения параметров приведены в таблице 5.1.

Таблица 5.1.

Типовые параметры внешнего RC-генератора

R [кОм]	C [пФ]	F [кГц]
100	70	100
31.5	20	1000
6.5	20	4000

Кварцевый или керамический резонатор подключается к микроконтроллеру по схеме рис. 5.1 (б). Частота колебаний в этом случае определяется резонансной частотой кристалла. Емкость конденсаторов С1, С2 подбирается в диапазоне 13…40 пФ.

Внешний генератор просто подключается к выводу XTAL1 микроконтроллера (рис 5.1 в). В качестве внешнего генератора можно использовать любую схему генератора с ТТЛ-выходом. Внешние генераторы, как правило, обладают высокой стабильностью и предсказуемостью параметров.

Выбор той или иной схемы тактового генератора программируется на стадии занесения программы в Flash-память контроллера. В контроллере *ATmega163* для хранения этой информации во флэш-памяти предназначены специальные биты-предохранители (*fuse bits)* CKSEL[3..0]. В процессе работы микроконтроллера эти биты не могут быть изменены. Программирование fuse-битов осуществляется в соответствии с таблицей 5.2.

Таблица 5.2.

Программирование тактового генератора

Вид генератора	CKSEL [3…0]
Внешний кварцевый или керамический резонатор	1111 - 1010
Внешний низкочастотный кристалл	1001 – 1000
Внешний RC-генератор	0111 – 0101
Внутренний RC-генератор	0100 – 0010
Внешний генератор	0001 - 0000

Fuse-биты CKSEL [3…0] используются также для задания некоторых временных параметров при сбросе микроконтроллера, поэтому в таблице 5.2. для описания каждого генератора предусмотрено много различных комбинаций этих битов.

6. СИСТЕМА СБРОСА

Сбросом считается перевод микроконтроллера в исходное состояние. При этом все регистры микропроцессорного ядра устанавливаются во вполне определенные начальные состояния, и микроконтроллер переходит к выполнению программы с фиксированного начального адреса. Таким адресом обычно является адрес $0.

6.1. Источники сброса

Причинами (источниками) сброса могут являться различные воздействия: включение питания и кратковременные его изменения, сигналы формируемые аппаратно вне и внутри микроконтроллера, а также инструкции программы. В частности, инструкция безусловного перехода на адрес $0 всегда приводит к сбросу устройства.

Источники сброса микроконтроллера *ATmega 163*:

- Сброс при включении питания (*Power-on Reset*.). Сброс происходит, если напряжение питания ядра ниже определенного порога (V_{POT}).
- Внешний сброс (*External Reset*). Сброс происходит при поступлении сигнала низкого уровня длительностью более 500 ns на внешний контакт RESET микросхемы.
- Сброс сторожевым таймером (*Watchdog Reset*). Сброс происходит по команде сторожевого таймера.
- Сброс при кратковременном провале напряжения питания (*Brown-out Reset*). ••••• ••••••••••, •••• ••••••••••• ••••••• •••••••••••• • •••••••• •••••• ••••••••••• •••• •••••••••••• •••••• ($V_{ВОТ}$).

По любой из этих причин микроконтроллер переходит к выполнению программы с адреса $0. В этой ячейке обычно размещают инструкцию jmp с адресом программы инициализации системы.

Структурная схема блока управления сбросом приведена на рис. 6.1.

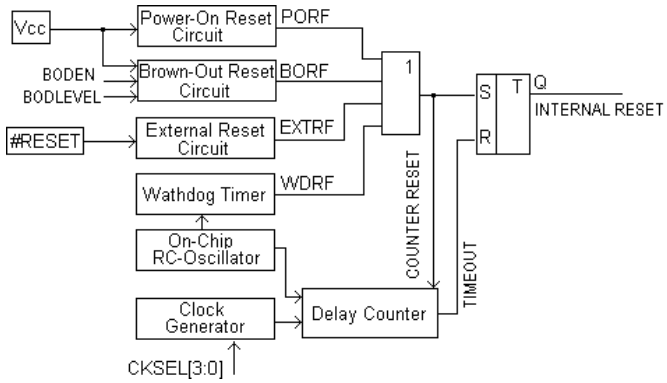


Рис. 6.1 Функциональная схема управления сбросом

Все сигналы сброса детектируются на кристалле специальными схемами.

- Схема сброса при включении питания (*Power-On Reset Circuit)* контролирует напряжение питания Vcc и запускается при Vcc > V_{POT}. При номинальном напряжении питания Vcc=5В, типовое значение порогового напряжения V_{POT} = 1,4 В.
- Схема сброса при кратковременном провале напряжения питания *(Brown-Out Reset Circuit)* сравнивает напряжение питания Vcc с уровнем V_{BOT}. Уровень сигнала V_{BOT} программируется с помощью специального бита программирования уровня BODLEVEL. Этот бит относится к группе *fuse*-битов микроконтроллера, позволяющих задавать их некоторые конфигурационные особенности. Fuse-биты программируются при занесении программы в память и в процессе работы микроконтроллера не могут быть изменены. Схемой контроля питания (*Brown-Out Reset Circuit)* управляют *fuse*-биты BODEN и BODLEVEL. При BODLEVEL = 1 уровень V_{BOT} равен 2,7В, а при BODLEVEL = 0 – 4В.
- Схема внешнего сброса (*External Reset Circuit)* управляется внешним сигналом низкого уровня #RESET.

Сигналы с этих схем и сигнал со сторожевого таймера (*Watchdog Timer)* фиксируются в регистре состояния процессорного ядра MCUSR (*MCU Status Register*) и объединенные по схеме ИЛИ, устанавливают RS-триггер. Длительность импульса внутреннего сброса (*Internal Reset)* на выходе триггера задается счетчиком задержки (*Delay Counter*). Фактически сброс микроконтроллера происходит по окончании импульса внутреннего сброса. Время задержки от момента поступления одного из сигналов сброса до окончания импульса сброса зависит от периода тактового генератора СК *(Clock*), от состояния *fuse*-бита BODLEVEL и может регулироваться *fuse*-битами CKSEL[3…0]. Время задержки при различных комбинациях битов CKSEL приведено в таблице 6.1.

Таблица 6.1.

Программирование временной задержки при сбросе

CKSEL [3…0]	BODLEVEL=0	BODLEVEL=1	Рекомендуемое использование
0000	4,2 мс + 6 СК	5,8 мс + 6 СК	Внешний генератор, быстрое включение
0001	30 μс + 6 СК	10 μс + 6 СК	Внешний генератор, BOD включен
0010	67 мс + 6 СК	92 мс + 6 СК	Внутренний RC генератор, медленное включение
0011	4,2 мс + 6 СК	5,8 мс + 6 СК	Внутренний RC генератор, быстрое включение
0100	30 μс + 6 СК	10 μс + 6 СК	Внутренний RC генератор, BOD включен
0101	67 мс + 6 СК	92 мс + 6 СК	Внешний RC генератор, медленное включение
0110	4,2 мс + 6 СК	5,8 мс + 6 СК	Внешний RC генератор, быстрое включение
0111	30 μс + 6 СК	10 μс + 6 СК	Внешний RC генератор,BOD включен
1000	67 мс+ 32К СК	92мс + 32К СК	Внешний низкочастотный кристалл

Окончание табл. 6.1.

CKSEL [3…0]	BODLEVEL=0	BODLEVEL=1	Рекомендуемое использование
1001	67 мс + 1К СК	92 мс + 1К СК	Внешний низкочастотный кристалл
1010	67 мс + 16К СК	92 мс + 16К СК	Кварцевый резонатор, медленное включение
1011	4,2 мс + 16К СК	5,8 мс + 16К СК	Кварцевый резонатор, быстрое включение
1100	30 μс +16К СК	10 μс +16К СК	Кварцевый резонатор, BOD включен
1101	67 мс + 1К СК	92 мс + 1К СК	Керамический резонатор, медленное включение
1110	4,2 мс + 1К СК	5,8 мс + 1К СК	Керамический резонатор, быстрое включение
1111	30 μс +1К СК	10 μс +1К СК	Керамический резонатор, BOD включен

Регистр состояния процессорного ядра MCUSR (*MCU Status Register*) позволяет определить источник сброса. Он доступен программно в пространстве регистров ввода/вывода по адресу $34($54) (рис. 6.2).

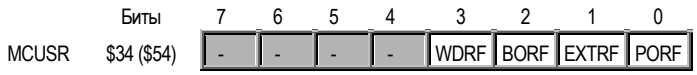


Рис.6.2. Регистр MCUSR

- Бит 3 – WDRF (*Watchdog Reset Flag*) – флаг сторожевого таймера. Бит устанавливается при сбросе процессорного ядра сторожевым таймером, сбрасывается при включении питания или путем записи логического нуля.
- Бит 2 – BORF *(Brown-out Reset Flag*) – флаг сброса при кратковременном провале питания. Бит устанавливается при сбросе от кратковременного провала напряжения питания, сбрасывается при включении питания или путем записи логического нуля.
- Бит 1 – EXTRF (*External Reset Flag*) – флаг внешнего сброса. Флаг устанавливается при сбросе от внешнего источника, сбрасывается при включении питания или путем записи логического нуля.
- Бит 0 – PORF (*Power-on Reset Flag*) – флаг сброса при включении питания. Бит устанавливается при включения питания, сбрасывается только путем записи логического нуля.

Флажки сброса используются для определения причины сброса. Если регистр очищен программно до момента сброса, то источник сброса может быть найден чтением регистра целиком или его отдельных флажков.

6.2. Сторожевой таймер

Сторожевой таймер *(Watchdog*) синхронизирован от отдельного внутреннего генератора на кристалле, работающего с частотой 1 МГц (при напряжении питания Vcc=5 В).

Задержка сброса устанавливается с помощью предделителя *(Prescaler)*. Настройка предделителя осуществляется установкой или сбросом битов WDP0….WDP2 регистра управления сторожевым таймером WDTCR (*WDT Control Register*). По истечении установленного времени задержки сторожевой таймер подает сигнал сброса на микроконтроллер.

Таблица. 6.2.

Интервалы задержки сторожевого таймера микроконтроллера *ATmega163*

WDP2	WDP1	WDP0	Количество циклов	Задержка сброса
0	0	0	16K	15 ms
0	0	1	32K	30 ms
0	1	0	64K	60 ms
0	1	1	128K	0.12 s
1	0	0	256K	0.24 s
1	0	1	512K	0.49 s
1	1	0	1,024K	0.97 s
1	1	1	2,048K	1.9 s

Сброс может быть предотвращен инструкцией wdr (`Watchdog Reset`).

Бит WDE (*Watchdog Enable*) в регистре WDTCR (рис. 6.3) позволяет подключать или отключать сторожевой таймер. При разрешении работы сторожевого таймера его состояние не определено и прежде, чем разрешать его включение, необходимо выполнить инструкцию wdr. В ином случае контроллер может быть сброшен прежде, чем будет выполнена команда wdr, прописанная после разрешения. Для предотвращения случайных ошибок запрет сторожевого таймера должен оформляться специальной процедурой выключения.

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

Рис. 6.3. Регистр управления сторожевым таймером WDTCR

- Бит 4 – WDTOE (*Watchdog Turn-off Enable*) - разрешение отключения сторожевого таймера. Бит должен быть установлен в состояние 1 при очистке бита WDE. В ином случае сторожевой таймер не будет запрещен. Установленный бит аппаратно очищается после четырех тактовых циклов.
- Бит 3 – WDE (*Watchdog Enable*): разрешение сторожевого таймера. Если бит WDE установлен в состояние 1 (сторожевой таймер разрешен) и если бит WDE очищен, то функционирование сторожевого таймера запрещено. Бит WDE может быть очищен, только если до этого установлен бит WDTOE. Для запрещения разрешенного сторожевого таймера необходимо выполнить следующую процедуру:
  - В одной операции записать 1 в биты WDTOE и WDE. Логическая 1 должна быть записана в WDE, даже если этот бит был установлен перед началом операции запрета сторожевого таймера.
  - За время последующих четырех тактовых циклов записать 0 в WDE.
- Биты 2..0 - WDP2, WDP1, WDP0 (*Watch Dog Timer Prescaler 2, 1 and 0*) - биты установки коэффициента предварительного деления сторожевого таймера. Состояния битов WDP2, WDP1 и WDP0 определяют коэффициент предварительного деления тактовой частоты разрешенного сторожевого таймера. Коэффициенты и соответствующие им промежутки времени представлены в таблице 6.2.

7. СИСТЕМА ПРЕРЫВАНИЙ

7.1. Алгоритм обработки прерываний

Сигнал запроса на прерывание вырабатывается периферийным устройством при его готовности к обмену информацией. Сигнал может появиться в произвольный момент времени.

Процессорное ядро может реагировать на запросы прерываний по-разному. В каждой архитектуре реализуется своя оригинальная система обслуживания прерываний. Однако общий алгоритм обработки запросов всегда содержит одни и те же действия ядра:

- при поступлении запроса на прерывание завершается выполнение текущей инструкции программы;
- записывается в стек содержимое программного счетчика и текущее состояние некоторых наиболее важных регистров общего назначения;
- идентифицируется прерывающее устройство;
- осуществляется переход к выполнению подпрограммы обслуживания прерывания для идентифицированного устройства;
- по окончании подпрограммы восстанавливает состояние прерванной программы за счет извлечения из стека содержимого регистров общего назначения и программного счетчика;
- возобновляется выполнение прерванной программы.

В различных системах обработки прерывания отдельные перечисленные действия реализуются аппаратно или программно.

Пользователю системы всегда предоставляется возможность отключить (замаскировать) отдельные источники прерываний. Для этого в схемах предусматриваются специальные программируемые регистры маски, где каждому источнику прерываний ставится в соответствие один бит.

При большом количестве источников прерываний порядок их обслуживания определяется приоритетом источника. Приоритет устанавливается в зависимости от важности и ответственности решаемой этим устройством задачи. Полагается, что запрос с большим приоритетом способен прервать подпрограммы обслуживания прерываний с меньшим приоритетом.

Наиболее простым является фиксированное распределение приоритетов. При этом каждому из источников запросов присваивается постоянный приоритет, соответствующий его порядковому номеру в схеме.

Циклическое распределение приоритетов (очередь) используется в тех случаях, когда ни одному из источников запросов исходно нельзя отдать явного предпочтения. В этом случае приоритеты входов изменяются в процессе работы контроллера после обработки любого из запросов. Вход последнего обслуженного запроса на прерывание получает низший приоритет, а приоритеты остальных входов при этом повышаются.

7.2. Вектора прерываний

Идентификация источника прерывания в системе может выполняться как программными, так и аппаратными средствами.

В первом случае, источник прерывания фиксируется установкой флажка в каком либо регистре. Поиск этого флажка осуществляет прерывающая программа, путем проверки всех возможных источников прерывания.

Во втором случае каждому источнику прерываний ставится в соответствие определенный адрес программы (вектор прерывания) и по этому адресу программист размещает команду перехода к соответствующей этому источнику подпрограмме обработки прерывания.

Например, контроллер *ATmega163* имеет 17 различных источников прерывания: два внешних и 15 внутренних. Практически каждый функциональный узел микроконтроллера является источником прерываний. В процессе работы он имеет возможность прервать выполняемую процессорным ядром программу. Каждое прерывание имеет фиксированный приоритет и отдельный вектор прерывания.

Вектора прерываний занимают в пространстве памяти программы адреса с $0 до $22. Самый младший вектор прерываний (с адресом $0) определен как сигнал сброса. Адрес каждого последующего вектора больше предыдущего на 2. Адреса $2 и $4 присвоены внешним входам запросов на прерывания (Таблица 7.1).

Таблица 7.1.

Вектора внешних прерываний контроллера ATmega 163

Вектор	Адрес	Обозначение источника	Описание источника
1	$000	RESET	Сброс (*External Pin, Power-on Reset, Brown-out Reset and Watchdog Reset*)

Вектор	Адрес	Обозначение источника	Описание источника
2	$002	INT0	Внешнее прерывание 0 (*External Interrupt 0)*
3	$004	INT1	Внешнее прерывание 1 (*External Interrupt 1*)

Полный список векторов приведен в приложении 2. Приоритет прерываний уменьшается с возрастанием адресов их векторов. Сигнал сброса (*Reset)* имеет самый высокий приоритет. Далее следуют прерывания от внешних источников, поступающие на контакты INT 0 и INT1 микроконтроллера.

Любое прерывание может быть запрещено (замаскировано) специальными битами в регистрах ввода/вывода микроконтроллера.

Типовая программа обработки векторов внешних прерываний выглядит следующим образом:

```
.cseg
.org $0
jmp RESET  ; Переход к подпрограмме сброса
jmp EXT_INT0        ; Переход к подпрограмме обработки IRQ0
jmp EXT_INT1        ; Переход к подпрограмме обработки IRQ1
…
.org $24
main:               ; Начало программы пользователя
```

Для выхода из любой подпрограммы обработки прерывания в системе команд микроконтроллера *ATmega163* предусмотрена специальная команда reti (*Interrupt Return*), восстанавливающая содержимое программного счетчика из стека. Для правильного функционирования системы прерывания в начале пользовательской программы обязательно должен быть загружен указатель стека.

Внешние прерывания вызываются с контактов INT1 и INT0. Для их обработки и программирования в контроллере задействованы отдельные биты трех регистров:

- регистра статуса SREG (*Status Register)*,
- регистра управления процессорным ядром MCUCR (*MCU Control Register)* и
- регистра маски прерываний GIMSK (*General Interrupt Mask register*) (рис. 7.1).

Биты	7	6	5	4	3	2	1	0
SREG $3F ($5F)	I							
MCUCR $35 ($55)					ISC11	ISC10	ISC01	ISC00
GIMSK $3B ($5B)	INT1	INT0						

Рис. 7.1. Регистры, задействованные в системе прерываний микроконтроллера ATmega163

Прерывания разрешаются только при единичном значении бита глобального разрешения прерываний I *(Interrupt)* в регистре статуса SREG. Сброс бита I в регистре SREG запрещает все прерывания. Если прерывания разрешены, то при появлении любого запроса бит I в регистре статуса SREG сбрасывается, а все дальнейшие прерывания

запрещаются. Программа пользователя может вновь установить этот бит, разрешив вложенные прерывания. Инструкция возвращения из подпрограммы обслуживания прерывания reti также устанавливает бит I в регистре статуса SREG, разрешая дальнейшие прерывания.

Биты ISCxx в регистре MCUCR описывают уровни и фронты прерывающих сигналов на контактах INT0 и INT1.

- Биты 3, 2 - ISC11, ISC10 (*Interrupt Sense Control 1 bit 1 and bit 0*): биты активизации входа INT1. Уровень и фронты на ножке INT1, которые инициируют прерывание, определены в таблице 5.6. Импульсы короче одного такта не гарантируют генерирование прерывания.

Таблица 7.2.

Активизация параметров входа INT1 (INT0)

ISC11 (ISC01)	ISC10 (ISC00)	Описание
0	0	Низкий уровень сигнала INT1 (INT0) генерирует запрос на прерывание.
0	1	Любое логическое изменение на INT1 (INT0) генерирует запрос на прерывание.
1	0	Задний фронт импульса на INT1 (INT0) генерирует запрос на прерывание.
1	1	Передний фронт сигнала на INT1 (INT0) генерирует запрос на прерывание.

- Биты 1, 0 – ISC01, ISC00 *(Interrupt Sense Control 1 bit 1 and bit 0)*: биты активизации входа INT0. Работают аналогично битам ISC11, ISC10.

В регистре маскирования прерываний GIMSK содержатся флаги внешних прерываний. Когда счетчик команд микроконтроллера устанавливается на конкретный вектор прерывания, соответствующий флаг в регистре GIMSK аппаратно сбрасывается. Флаги прерываний можно очистить, записав в соответствующий бит регистра логическую единицу.

- Бит 7 - INT1: внешнее прерывание INT1 разрешено. Если бит INT1 установлен и бит I в регистре статуса SREG также равен 1, то внешний вход запроса на прерывание INT1 становится активным.
- Бит 6 - INT0: внешнее прерывание INT0 разрешено. Если бит INT0 установлен и бит I в регистре статуса SREG также равен 1, то внешний вход запроса на прерывание INT0 становится активным.

8. ЭНЕРГОНЕЗАВИСИМАЯ ПАМЯТЬ ДАННЫХ

Энергонезависимая память типа EEPROM отличается от памяти данных типа SRAM существенно большим временем чтения и записи информации. Время обращения при записи обычно составляет несколько миллисекунд и к тому же сильно зависит от напряжения питания микроконтроллера. По этой причине обращение к EEPROM всегда осуществляется через регистры ввода/вывода. В ряде случаев в архитектуре микроконтроллера предусматриваются специальные сервисные функции, которая позволяют пользователю программными средствами обнаруживать момент готовности EEPROM к записи. Для индикации момента готовности EEPROM к записи новых данных

устанавливаются специальные прерывания по завершению записи EEPROM (*EEPROM Write Complete*). Случайная запись в EEPROM предотвращается выполнением специальных процедур. Для работы с EEPROM в структуре микроконтроллера всегда предусматривается несколько регистров, хранящих адрес ячейки памяти, данные, команды и флаги состояния памяти. Например, микроконтроллеру ATmega163 для записи байта данных в EEPROM необходимо 1.9 - 3.8 мс. В процессе работы с памятью он использует (рис. 8.1):

- регистр статуса микроконтроллера SREG,
- два адресных регистра EEARH и EEARL *(EEPROM Address Registers),*
- регистр данных EEDR (*EEPROM Data Register),*
- регистр управления EECR (*EEPROM Control Register*).

		7	6	5	4	3	2	1	0
SREG	$3F ($5F)	I							
EEARH	$1F ($3F)	-	-	-	-	-	-	-	EEAR8
EEARL	$1E($3E)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0
EEDR	$1D ($3D)	EEDR7	EEDR6	EEDR5	EEDR4	EEDR3	EEDR2	EEDR1	EEDR0
EECR	$1C ($3C)	-	-	-		EERIE	EEMWE	EEWE	EERE

Рис. 8.1. Регистры, задействованные в управлении EEPROM

Регистры адреса EEPROM (*EEPROM Address Registers*) EEARH и EEARL позволяют адресовать 512 байт памяти пространства EEPROM.

- Бит 0 регистра EEARH и биты 7….0 регистра - EEARL - EEAR8…EEAR0 (*EEPROM address*) образуют 9 битный адрес ячейки памяти.

Начальные значения EEAR8….EEARO не определены. Обращаться за данными к EEPROM можно только после записи адреса.

Регистр данных EEDR (*EEPROM Data Register*) хранит данные для записи в EEPROM при выполнении операции записи или данные, считанные из EEPROM, при выполнении операции чтения.

- Биты 7 .. EEDR7…. EEDR0 *(EEPROM Data*) - данные EEPROM.

Регистр управления EECR (*EEPROM Control Register*) предназначен для управления записью и чтением EEPROM.

- Бит 3 – EERIE (*EEPROM Ready Interrupt Enable*) - разрешение прерывания по готовности EEPROM. При установленных в состояние 1 бите EERIE и I-бите регистра SREG разрешается прерывание по готовности EEPROM. При очищенном бите EERIE прерывание запрещено. Запрос прерывания по готовности EEPROM при очищенном бите EEWE генерируется непрерывно. Вектор прерывания EE_RDY имеет адрес $1E.
- Бит – EEMWE (*EEPROM Master Write Enable*) - управление разрешением записи EEPROM. Бит EEMWE определяет, будет ли установленный в состояние 1 бит EEWE разрешать запись в EEPROM. При установленном в состояние 1 бите EEMWE установка бита EEWE приведет к записи в EEPROM по заданному адресу. Если же бит EEMWE будет находиться в состоянии 0, то установка бита EEWE никакого эффекта не окажет. Установленный программным путем бит EEMWE аппаратно очищается через четыре тактовых цикла.

- Бит 1 – EEWE (*EEPROM Write Enable*) - разрешение записи EEPROM. Сигнал разрешения записи EEPROM EEWE является стробом записи в EEPROM. Запись установленных данных по установленному адресу EEPROM выполняется по установке бита EEWE. При этом бит EEMWE обязательно должен быть в состоянии 1, иначе запись не произойдет. Процесс записи EEPROM выполняется следующей процедурой (очередность выполнения пунктов 2 и 3 значения не имеет):
  - Подождать сброса бита EEWE в состояние 0.
  - Записать новый адрес в EEAR (при необходимости).
  - Записать новые данные (при необходимости).
  - Установить в состояние 1 бит EEMWE регистра EECR.
  - В течение четырех тактовых циклов после установки EEMWE установить в состояние 1 бит EEWE.

  По истечении времени записи (типовое значение 2,5 мс при напряжении питания Vcc = 5 В) бит EEWE аппаратно очищается. Пользовательское программное обеспечение может тестировать состояние этого бита для определения момента сброса его в 0, чтобы начать запись следующего байта. После установки бита EEWE процессорное ядро, прежде чем начать выполнение следующей команды, останавливается на два тактовых цикла.
- Бит 0 – EERE (*EEPROM Read Enable*) - разрешение чтения EEPROM. Сигнал разрешения чтения EERE является стробом чтения EEPROM. Бит EERE должен быть установлен после записи в регистр EEAR требуемого адреса. После аппаратной очистки бита EERE считываемые данные будут располагаться в регистре EEDR. Считывание байта данных выполняется одной командой и не требует опроса бита EERE. При установленном бите EERE CPU останавливается на четыре тактовых цикла, прежде чем начнет выполнение следующей команды. Пользователю необходимо тестировать состояние бита EEWE перед началом операции чтения. Если новые данные или адрес будут записываться в регистры ввода/вывода EEPROM в то время, когда будет выполняться операция записи, то операция записи будет прервана и результат записи будет неопределенным.

Содержимое EEPROM может быть разрушено при снижении напряжения питания V_{CC} до уровня, при котором процессорное ядро и EEPROM работают неправильно.

Разрушение данных EEPROM происходит по двум причинам. Во-первых, для правильного выполнения операций записи необходимо, чтобы напряжение питания было не ниже определенного. Во-вторых, само ядро при слишком низком напряжении питания может неправильно выполнять команды.

Разрушения данных можно предотвратить при следующих условиях:

- Во время снижения напряжения питания аппаратно удерживать сигнал сброса #RESET в активном (низком) состоянии.
- В период снижения напряжения питания переводить ядро микроконтроллера в режиме пониженного энергопотребления (*Power Down Sleep)*.
- Сохранять константы в Flash-памяти программ, если нет необходимости изменять их программно. *Flash* память не изменяется процессорным ядром и, следовательно, не может быть повреждена.

9. ПОРТЫ ВВОДА-ВЫВОДА

9.1. Организация ввода/вывода

Порты ввода-вывода обеспечивают ввод и вывод данных в параллельном формате. Обычно порты ввода-вывода выполняются 8-разрядными. В режиме ввода данные с внешних контактов порта пересылаются в регистры микроконтроллера. В режиме вывода данные из регистров перемещаются на контакты микроконтроллера. Вывод данных, как правило, производится в «защелку» порта. Данные при этом присутствуют на выходных контактах до новой операции вывода в этот порт. В системе команд микроконтроллера для ввода и вывода данных обычно предусматриваются специальные команды.

В зависимости от выполняемых функций порты могут быть:

- однонаправленными, предназначенными только для выполнения одной из операций (ввод или вывод) по всем линиям;
- двунаправленными, предназначенными для выполнения любой из операций ввода-вывода по всем линиям одновременно; направление передачи может быть изменено программно в процессе работы;
- с индивидуальной настройкой линий; направление передачи данных по каждой линии программируется независимо от остальных.

Последний вариант построения схемы порта в настоящее время наиболее распространен. Например, микроконтроллер *ATmega163* имеет 32 линии ввода-вывода с индивидуальной настройкой, сгруппированных в 4 параллельных порта: порт A порт B, порт C, порт D. Направление передачи данных любого вывода любого порта может быть изменено независимо от направлений других выводов. Для работы с портами в микроконтроллере предусмотрено 12 регистров, по три на каждый из портов (рис. 5.8):

- регистры данных (*Data Register*): PORTA , PORTB, PORTC и PORTD;
- регистры направления (*Data Direction Register):* DDRA, DDRB, DDRC и DDRD;
- регистры входных контактов *(Port A Input Pins)*: PINA; PINB, PINC и PIND.

Все выводы портов имеют индивидуальные подтягивающие резисторы (*pull-up resistors*). Для подключения этих резисторов в регистре специальных функций ввода/вывода SFIOR *(Special Function Input Output Register)* предусмотрен бит PUD *(Pull-up Disabled).*

		7	6	5	4	3	2	1	0
PORTA	$1B ($3B)	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
DDRA	$1A ($3A)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
PINA	$19 ($39)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
PORTB	$18 ($38)	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
DDRB	$17 ($37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
PINB	$16 ($36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
PORTC	$15 ($35)	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
DDRC	$14 ($34)	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
PINC	$13 ($33)	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
PORTD	$12 ($32)	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
DDRD	$11 ($31)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
PIND	$10 ($30)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
SFIOR	$30 ($50)						PUD		

Рис. 9.1. Регистры портов ввода/вывода

Любая линия порта выполняет функции выхода при записи логической единицы в соответствующий бит регистра направления DDRx (x ∈ A, B, C, D).

Регистры PINx не хранят информацию и фактически не являются настоящими регистрами. Они разрешают доступ к физическим сигналам на линиях соответствующего порта. При чтении PORTx читается защелка данных выбранного порта, а при чтении PINx – значение на контактах порта. Регистры PINx доступны только для чтения, в то время как регистры PORTx и DDRx - для чтения и записи.

Внутренние подтягивающие резисторы подключаются только при PUD=0, если контакты портов сконфигурированы как входы.

Выходы портов выдерживают втекающий ток до 20mA и могут быть непосредственно подключены к светодиодным индикаторам. Однако, вытекающий ток порта не должен быть более 4 мА , а суммарная загрузка порта - не более 80 мА.

9.2. Алгоритмы обмена данными

Порты ввода-вывода предназначены для связи микроконтроллера с различными объектами и могут реализовывать различные алгоритмы обмена данными:

- асинхронный программный обмен,
- синхронный обмен,
- ввод-вывод с сигналами квитирования.

Обмен данными между портами и объектами обеспечивается специальными подпрограммами – драйверами, создаваемыми индивидуально для каждого объекта.

Асинхронный обмен

В режиме асинхронного программного обмена ввод и вывод данных производится по программе в моменты выполнения инструкций ввода и вывода данных. Предполагается, что объект всегда готов к обмену: при вводе – данные в момент выполнения инструкции in

присутствуют на линиях порта, при выводе – данные будут прочитаны с линий порта до следующего вывода.

Например, микроконтроллер ATmega163 осуществляет асинхронный вывод данных при выполнении фрагмента программы:

```
.equ porta = $1B        ;
.equ ddra = $1A         ;
.cseg
ldi r16,$FF             ;запись $FF в r16
out ddra, r16           ;включение порта A на вывод
out porta, r0           ;вывод данных из регистра r0 в порт A.
```

Время выполнения команды вывода равно двум периодам тактового сигнала.

Тот же микроконтроллер в течении двух тактов введет данные с линий порта при выполнении фрагмента программы:

```
.equ pina = $19         ;
.cseg
in r0, pina     ; ввод данных из порта pina в регистр r0.
```

По адресу $19 в пространстве ввода-вывода микроконтроллера размещен регистр pina, с входных линий которого и будут взяты данные во время выполнения инструкции.

Симплексный обмен

Симплексным считается однонаправленный обмен данными. Такой обмен обычно является синхронным. В этом случае каждое изменение данных на линиях порта сопровождается сигналом синхронизации (стробом). Строб генерируется источником данных и предназначается для задания момента записи данных в регистр приемника.

При выводе данных сигнал строба должен сформировать микроконтроллер, используя для этого специальные линии шины управления или отдельные биты портов ввода-вывода.

На рис. 9.2 показан вариант соединения микроконтроллера ATmega163 с посимвольным принтером, имеющим 8-битный вход для приема данных DATA, выход сигнала готовности READY и вход стробирования #STB. Активным на входе стробирования является сигнал низкого уровня.

Порт PORTA микроконтроллера и бит PB6 порта PORTB программируются на вывод данных, бит PB2 порта PORTB на ввод. Подпрограмма вывода должна:

- осуществить проверку готовности принтера (чтение сигнала готовности принтера READY и его анализ),
- при обнаружении сигнала READY=1 вывести данные в порт PORTA,
- подтвердить вывод данных выводом сигнала стробирования #STB=0 для записи данных в принтер.

Если принтер не готов к обмену микроконтроллер через заданный интервал времени повторяет операцию.

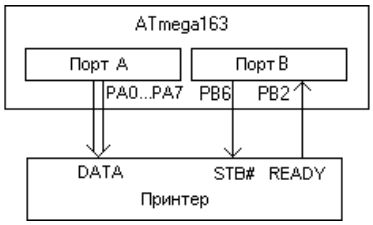


Рис. 9.2. Подключение принтера к микроконтроллеру

Режим синхронного ввода позволяет точно задать момент считывания данных с объекта. В этом режиме сигнал стробирования, подтверждающий готовность данных, поступает с объекта на микроконтроллер. При появлении строба микроконтроллер должен осуществить ввод данных с порта.

Полудуплексный обмен

Полудуплексным считается двунаправленный синхронный обмен, при котором в любой момент времени передача данных может производиться только в одном направлении. Направление передачи данных порта меняется в процессе работы в зависимости от решаемой в текущий момент задачи. На рис. 9.3 изображен вариант соединения микроконтроллера ATmega163 с объектом для обмена данными в полудуплексном режиме.

В схеме на рис. 9.3 передача данных в параллельном формате осуществляется по линиям порта PORTA. Для выдачи и приема четырех сигналов управления обменом (сигналы квитирования): строб ввода #STB IN (*Strobe Input*), строб вывода #STB OUT (*Strobe Out*), подтверждение ввода #ACK IN (*Acknowledge Input*) и подтверждение вывода #ACK OUT (*Acknowledge Out*), использованы две линии порта PORTB и входы запросов на прерывания INT0 и INT1.

При поступлении сигнала #STB IN = 0 контроллер должен выставить сигнал подтверждения #ACK IN = 0 и осуществить запись во входной регистр порта PORTA.. При низком уровне сигнала ASK IN объекту запрещается формировать новый сигнал #STB IN. По окончании записи контроллер снимает сигнал #ACK IN, разрешая повторную передачу данных.

Работа микроконтроллера в режиме вывода аналогична. Микроконтроллер выводит данные в порт PORTA, подтверждая вывод сигналом #STB OUT = 0. Объект формирует сигнал подтверждения #ACK OUT = 0, сообщая контроллеру о готовности к приему данных. При снятии сигнала #ACK OUT контроллер должен снять сигнал #STB OUT и может вновь перейти к выводу данных.

Одновременная передача данных в двух направлениях при полудуплексном обмене невозможна.

Дуплексный обмен

Дуплексным считается двунаправленный синхронный обмен, при котором в любой момент времени возможна передача данных в двух направлениях. В этом случае для передачи данных в каждом направлении выделяется свой однонаправленный порт. На рис.

9.4 изображен вариант соединения микроконтроллера *ATmega 163* с объектом для обмена данными в дуплексном режиме.

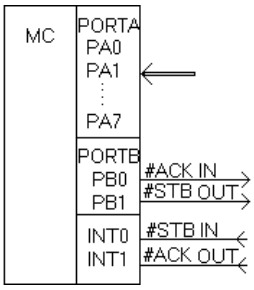


Рис. 9.3. Подключение микроконтроллера при полудуплексном обмене

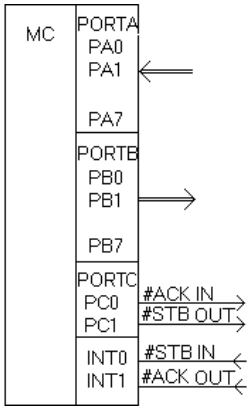


Рис. 9.4. Подключение микроконтроллера при дуплексном обмене

В схеме на рис. 9.4 порт PORTA работает на ввод данных, порт PORTB – на вывод, а линии PC0 и PC1 порта PORTC использованы для вывода сигналов квитирования #ACK IN и #ACK OUT. Для ввода сигналов квитирования от объекта #STB IN и #ACK OUT использованы входы запросов на прерывания INT0 и INT1. При таком подключении задача ввода данных в микроконтроллер получает больший приоритет, чем задача вывода.

10. АНАЛОГО-ЦИФРОВЫЕ ПРЕОБРАЗОВАТЕЛИ

Аналого-цифровой преобразователь ADC (*Analog Digital Converter)* осуществляет преобразование напряжения в цифровой код. Он предназначен для оцифровки и ввода в микроконтроллер аналоговых сигналов с различных датчиков физических величин. Схемы преобразователей различны. В зависимости от принципа построения меняются и свойства преобразователя.

10.1. Принципы аналого-цифрового преобразования

Параллельный преобразователь

В параллельном преобразователе (рис. 10.1) входной сигнал подается сразу на множество компараторов, осуществляющих сравнение сигнала с опорными напряжениями. Опорные напряжения формируются цепочкой резисторов, делящих эталонное напряжение U_0 на равные части.

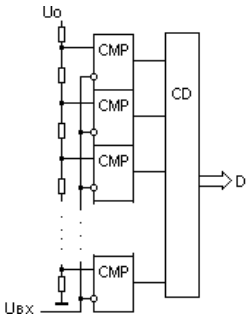


Рис. 10.1. Параллельный аналого-цифровой преобразователь.

Такие схемы ADC работают очень быстро, но сложны и используются редко.

Преобразователь последовательного приближения

Основными элементами преобразователя (рис. 10.2) является регистр последовательных приближений, код из которого с помощью цифроаналогового преобразователя преобразуется в напряжение. Компаратор CMP сравнивает входное напряжение с выходным напряжением преобразователя и через устройство управления воздействует на регистр.

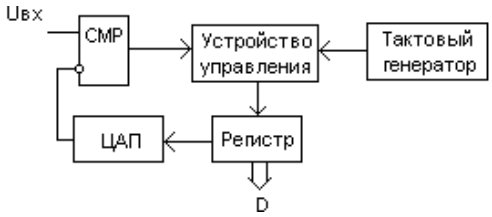


Рис. 10.2. Преобразователь последовательного приближения

Преобразование выполняется за несколько тактов. В первом такте в старший разряд регистра последовательных приближений записывается единица. Если в результате сравнения на выходе компаратора устанавливается единичный сигнал, единица в старшем разряде регистра сохраняется. В противном случае – сбрасывается. Далее, в том же порядке, формируется второй по старшинству разряд результата, потом - третий и т.д. Для получения результата необходимо *n* тактов, где число *n* равно разрядности преобразователя.

Интегрирующий преобразователь

Интегрирующий ADC для сравнения входного сигнала с эталонным использует заряд конденсатора. Сначала (рис. 10.3) конденсатор в течении фиксированного промежутка времени *T1* заряжается током, пропорциональным входному сигналу. После это он разряжается постоянным током с определенным значением. Время разряда конденсатора *T2* пропорционально значению входного напряжения. Оно фиксируется с помощью счетчика и поступает на выход схемы (рис. 10.4).

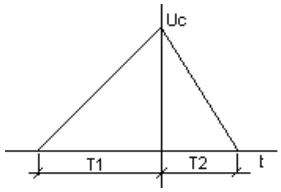


Рис. 10.3. Интегрирование сигнала в преобразователе

Интервал времени *T1* задается включением ключа *S1*. По окончании *T1* ключ *S1* размыкается, а *S2* – замыкается. Опорное напряжение U_o должно иметь знак противоположный знаку напряжения входного. Компаратор, устройство управления и счетчик определяют выходной код D, пропорциональный интервалу *T2*.

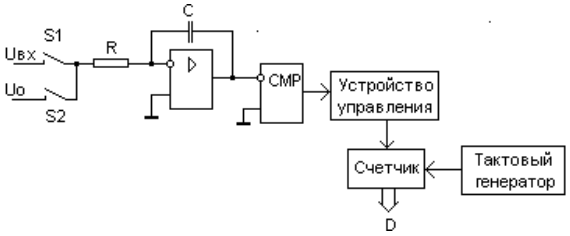


Рис. 10.4. Интегрирующий ADC

Интегрирующие схемы ADC имеют 8-16 разрядов и могут представлять результат в двоичном или двоично-десятичном коде.

Сигма-дельта преобразователь

Сигма-дельта преобразователи являются разновидностью интегрирующих ADC, в которых входной ток компенсируется коммутируемым зарядом от встроенного источника (рис. 10.5). Импульсы тока фиксированной длительности на каждом такте могут быть подключены к входу интегратора. В суммирующей точке интегратора поддерживается нулевой средний ток. Счетчик подсчитывает количество импульсов, поступающих в суммирующую точку за фиксированный период времени. Результат счета пропорционален входному напряжению.

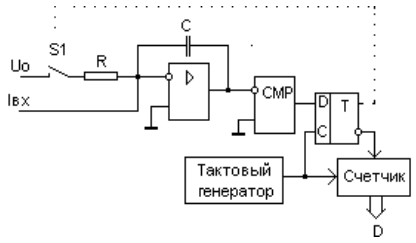


Рис. 10.5. Сигма-дельта преобразователь

10.2. Управление аналого-цифровым преобразователем

В состав микроконтроллеров обычно включают 8 - 16-битные многоканальные преобразователи с большим набором встроенных функций. При этом все функции преобразователя программируются и могут быть изменены в процессе работы.

Например, микроконтроллер ATmega163 оснащен 10-разрядным ADC последовательных приближений (рис. 10.6). ADC подсоединен к 10-канальному аналоговому мультиплексору (MUX), позволяющему подать на вход преобразователя любой из восьми входных сигналов со входов ADC0…ADC7, либо эталонное напряжение 1,22В, либо сигнал со входа AGND. Вывод AGND рекомендуется подсоединить к точке с нулевым потенциалом GND (*Ground*). ADC содержит схему выборки/хранения SHC (*Sample&Hold Comparator)* , удерживающую напряжение входа во время преобразования на неизменном уровне.

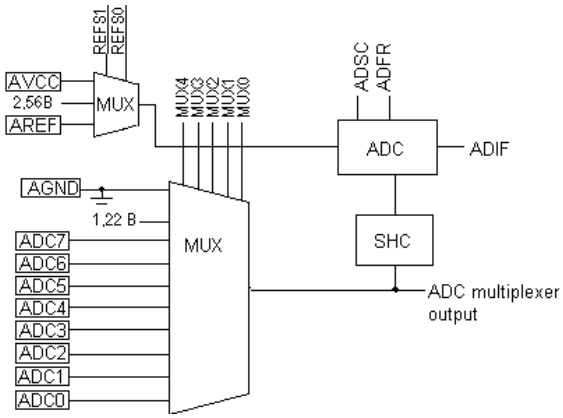


Рис. 10.6. Структура аналого-цифрового преобразователя

Аналого-цифровой преобразователь преобразует напряжение аналогового входного сигнала в 10-разрядное цифровое значение методом последовательных приближений. Минимальное значение входного напряжения равно напряжению на контакте AGND, максимальное значение не должно превышать напряжение на контакте AREF. Результат в виде 10-битного двоичного числа D равен:

$$D = \frac{U}{U_0} \cdot 2^{10},$$

где U – входное напряжение, а U_0 – опорное напряжение преобразователя.

В качестве источника опорного напряжения преобразователя можно использовать внешний сигнал с вывода AREF, внутренний источник 2,56В, либо напряжение питания аналоговой части микроконтроллера с вывода AVCC. Напряжение на выводе AVCC не должно отличаться от напряжения питания V_{cc} более, чем на ±0,3 В.

Например, если аналоговый мультиплексор подключает ко входу ADC эталонное напряжение *U =1,22В*, а в качестве опорного напряжения использовать источник U_0 *=2,56В*, то результат преобразования:

*D =1,22*1024/2,56=488=$1E8=0b111101000.*

Для управления преобразователем в микроконтроллере используются регистры:

- Регистр управления мультиплексором ADMUX (*ADC Multiplexer Selection Register);*
- Регистр управления аналого-цифровым преобразователем ADCSR (*ADC Control and Status Register)*
- Регистры данных ADCL и ADCH (*ADC Low u ADC High).*
- Регистр состояния микроконтроллера SREG *(Status Register).*

	Биты	7	6	5	4	3	2	1	0
ADMUX	$07 ($27)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
ADCSR	$06 ($26)	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADCH	$05 ($25)	SIGN	-	-	-	-	-	ADC9	ADC8
ADCL	$04 ($24)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
SREG	$3F ($5F)	I							

Рис. 10.7. Регистры, используемые аналого-цифровым преобразователем

Аналого-цифровой преобразователь может работать в двух режимах: режиме однократного преобразования и в циклическом режиме. В режиме однократного преобразования каждое преобразование инициируется пользователем. В циклическом режиме аналого-цифровой преобразователь осуществляет выборку и обновление содержимого регистра данных непрерывно. Выбор режима производится битом ADFR (*(ADC Free Run*) регистра ADCSR.

Работа аналого-цифрового преобразователя разрешается установкой в состояние 1 бита ADEN в регистре ADCSR. Преобразование начинается с установки в состояние 1 бита начала преобразования ADSC (*ADC Start Conversion*). Если в процессе выполнения преобразования производится смена канала данных, то ADC вначале закончит текущее преобразование, а потом выполнит переход к другому каналу.

Поскольку аналого-цифровой преобразователь формирует 10-разрядный результат, то по завершении преобразования результирующие данные размещаются в двух регистрах данных ADCH и ADCL. Для обеспечения соответствия результирующих данных считываемому уровню используется специальная логика защиты. Этот механизм работает следующим образом: при считывании данных первым должен быть считан регистр ADCL. Если регистр ADCL считан, обращение аналого-цифрового преобразователя к регистрам данных блокируется. Таким образом, если после считывания состояния ADCL, но до считывания ADCH, будет завершено следующее преобразование, ни один из регистров не будет обновлен и записанный ранее результат не будет искажен. Обращение аналого-цифрового преобразователя к регистрам ADCH и ADCL разрешается по завершении считывания содержимого регистра ADCH.

Аналого-цифровой преобразователь имеет свое собственное прерывание ADC (вектор $1C), которое может быть активизировано по завершению преобразования. Когда

обращение к регистрам запрещено, в процессе считывания регистров ADCL и ADCH, прерывание будет активизироваться, даже при потере результата.

Регистр ADMUX (*ADC Multiplexer Selection Register*) предназначен для управления входным аналоговым мультиплексором

- Биты 7 и 6 - REFS1..0 (*Reference Selection Bits*) – обеспечивают выбор эталонного напряжения на входе AREF аналого-цифрового преобразователя. Выбор производится в соответствии с таблицей 10.1. Изменение этих битов во время процесса преобразования приводит к ошибке. Для её исключения пользователь должен игнорировать первый результат после изменения битов. Внутренние источники напряжения не могут быть использованы, если к контакту AREF приложено внешнее напряжение.

Таблица 10.1.

Выбор источника опорного напряжения аналого-цифрового преобразователя

REFS1	REFS0	Выбор источника напряжения
0	0	AREF, внутреннее напряжение Vref отключено
0	1	AVCC с внешним конденсатором на контакте AREF
1	0	Резерв
1	1	Внутренний источник 2.56В с внешним конденсатором на AREF

- Бит 5 – ADLAR (*ADC Left Adjust Result*) – воздействует на запись результата в регистры данных ADCL и ADCH. При ADLAR=0 можно использовать упрощенное 8-битное преобразование.
- Биты 4..0 - MUX4..MUX0 (*Multiplexer bits)* – предназначены для выбора входа, коммутируемого на вход преобразователя. Выбор осуществляется в соответствии с таблицей 10.2. Изменение этих битов в процессе преобразования, когда флаг ADIF в регистре ADCSR установлен, не приводит к изменению результата.

Таблица 10.2.

Выбор входного сигнала ADC

MUX4..0	Подключаемый контакт
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7
01000. .11101	Резерв
11110	1.22V
11111	0V (AGND)

Регистр - ADCSR (*ADC Control and Status Register*) предназначен для управления работой аналого-цифрового преобразователя.

- Бит 7 – ADEN (*ADC Enable*) - разрешение работы ADC. Очистка бита запрещает работу ADC. Запрещение ADC в процессе преобразования прекращает преобразование.
- Бит 6 – ADSC (*ADC Start Conversion*) - запуск преобразования ADC. В режиме однократного преобразования для запуска каждого цикла преобразования необходимо устанавливать бит ADSC в состояние 1. В циклическом режиме бит ADSC устанавливается в состояние 1 только при запуске первого цикла преобразования. Каждый раз после первой установки бита ADSC, выполненной после разрешения или одновременно с разрешением, будет выполняться пустое преобразование. Это пустое преобразование активизирует преобразователь. ADSC будет сохранять состояние 1 в течение всего цикла преобразования и сбрасывается по его завершению. При выполнении пустого преобразования, предшествующего активизируемому, бит ADSC остается установленным до завершения активируемого преобразования. Запись 0 в этот бит эффекта не оказывает.
- Бит 5 – ADFR *(ADC Free Run Select)* - установка циклического режима работы ADC. При установленном в состояние 1 бите ADFR аналого-цифровой преобразователь будет работать в циклическом режиме. В этом режиме производятся выборки и обращения к регистрам непрерывно (одно за другим). Очистка бита приводит к прекращению циклического режима.
- Бит 4 – ADIF (*ADC Interrupt Flag*) - флаг прерывания ADC. Данный бит устанавливается в состояние 1 по завершению преобразования и обновления регистров данных. Прерывание по завершению преобразования ADC выполняется, если в состояние 1 установлены бит ADIE и I - бит регистра статуса SREG. Бит ADIF сбрасывается аппаратно при выполнении подпрограммы обработки соответствующего вектора прерывания. Кроме того, бит ADIF может быть очищен записью во флаг логической 1. Этого необходимо остерегаться при чтении-модификации-записи ADCSR, поскольку может быть запрещено отложенное прерывание. Это применимо и в случаях использования команд sbi и cbi.
- Бит 3 – ADIE (*ADC Interrupt Enable*) - разрешение прерывания ADC. При установленных в состояние 1 бите ADIE и I-бите регистра SREG активируется прерывание с вектором $1C по завершению преобразования ADC.
- Биты 2..0 - ADPS2..ADPS0 (*ADC Prescaler Select Bits)* - выбор коэффициента предварительного деления. Данные биты определяют коэффициент деления тактовой частоты микроконтроллера для получения необходимой тактовой частоты ADC.

Таблица 10.3.

Выбор коэффициента предварительного деления

ADPS2	ADPS1	ADPS0	Коэффициент деления
0	0	0	Без деления
0	0	1	2
0	1	0	4
0	1	1	8

Окончание табл. 10.3.

ADPS2	ADPS1	ADPS0	Коэффициент деления
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Регистры ADCL и ADCH являются регистрами данных. Их содержимое зависит от состояния бита ADLAR регистра ADMUX.

Когда преобразование выполнено десятибитный результат находится в этих двух регистрах. Если младший регистр ADCL считан, то регистры не изменяются до чтения старшего регистра ADCH. ADLAR бит в ADMUX воздействует на представление результата. Если ADLAR сброшен, то результат представлен в виде, изображенном на рис. 10.8.

	Биты	7	6	5	4	3	2	1	0
ADCH	$05 ($25)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCL	$04 ($24)	ADC1	ADC0						

Рис. 10.8. Представление результата в регистре данных при ADLAR=0

Если достаточным является 8-битное преобразование, то при ADLAR=0 можно считывать только старший байт результата (регистр ADCH).

11. АНАЛОГОВЫЕ КОМПАРАТОРЫ

Аналоговые компараторы осуществляют сравнение двух напряжений. Результатом сравнения является логический сигнал, фиксирующий момент равенства входных сигналов. Выход компаратора может быть использован в качестве запроса на прерывание. При этом пользователь может программировать формирование запроса по переднему или заднему фронту сигнала, либо по любому его изменению.

Схема компаратора микроконтроллера *ATmega163* приведена на рис. 11.1..

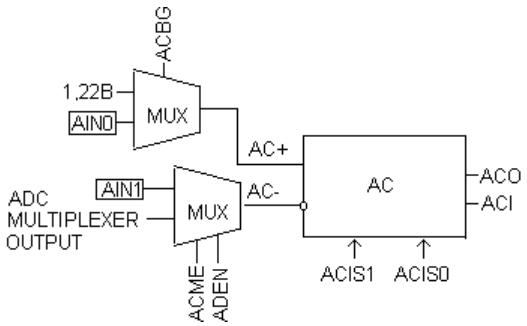


Рис.11.1. Структурная схема аналогового компаратора

Аналоговый компаратор сравнивает уровни на положительном (AC+) и отрицательном (AC-) входах. При напряжении на входе (AC+) большем, чем напряжение на входе (AC-), выход аналогового компаратора ACO устанавливается в состояние 1. Компаратор может формировать запрос на прерывание. Пользователь может задать формирование запроса по нарастающему или падающему фронту или по переключению.

В работе компаратора используются регистры (рис. 11.2):

- Регистр управления аналоговым компаратором ACSR (*Analog Comparator Control And Status Register*);
- Регистр специальных функций ввода/вывода SFIOR (*Special Function Input Output Register);*
- Регистр состояния аналого-цифрового преобразователя ADCSR *(ADC Status Register);*
- Регистр мультиплексора аналого-цифрового преобразователя ADCMUX *(ADC Multiplexer).*
- Регистр состояния микроконтроллера SREG *(Status Register).*

Биты		7	6	5	4	3	2	1	0
ACSR	$08 ($28)	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
SFIOR	$30 ($50)					ACME			
ADCSR	$06 ($26)	ADEN							
ADMUX	$07 ($27)						MUX2	MUX1	MUX0
SREG	$3F ($5F)	I							

Рис. 11.2. Регистры, задействованные в работе аналогового компаратора

Регистр ACSR (*Analog Comparator Control And Status Register*) предназначен для управления аналоговым компаратором.

- Бит 7 – ACD (*Analog Comparator Disable)* - отключает аналоговый компаратор. Когда этот бит установлен, питание на аналоговый компаратор не подается. Бит может быть установлен в любое время. При изменении бита прерывание от компаратора должно быть заблокировано очисткой бита ACIE.
- Бит 6 – ACBG (*Analog Comparator Bandgap)* - выбор эталона аналогового компаратора. Когда этот бит установлен, и BOD позволяется (fuse-бит BODEN запрограммирован), фиксированное напряжение 1.22V поступает на положительный вход аналогового компаратора. Когда бит сброшен, к положительному входу подключается контакт AIN0
- Бит 5 – ACO (*Analog Comparator Output*) - выход аналогового компаратора.
- Бит 4 – ACI (*Analog Comparator Interrupt Flag)* - флаг прерывания по аналоговому компаратору. Данный бит устанавливается в состояние 1 в случае формирования компаратором прерывания, определяемого ACIS1 и ACIS0. Подпрограмма обработки прерывания по аналоговому компаратору будет выполняться при установленном бите ACIE и установленном бите глобального прерывания в регистре SREG. Бит ACI очищается аппаратно при выполнении соответствующего вектора обработки прерывания, Бит ACI можно очистить, также, записью во флаг логической 1. При модификации

других битов регистра ACSR командами SBI или CBI бит ACI будет очищен, если он был установлен перед этими операциями.

- Бит 3 – ACIE (*Analog Comparator Interrupt Enable)* - разрешение прерывания по аналоговому компаратору. При установленном бите ACIE и установленном бите глобального прерывания регистра SREG активируется прерывание ANA_COM с вектором $20. При сброшенном бите ACIE прерывание запрещено.
- Бит 2 – ACIC (*Analog Comparator Input Capture Enable)* - разрешение входа захвата аналогового компаратора. Установленный в состояние 1 бит ACIC разрешает срабатывание функции захвата входа таймера/счетчика1 по переключению аналогового компаратора. В этом случае, выход аналогового компаратора подсоединяется непосредственно ко входной цепи логики захвата входа, что обеспечивает использование функций подавления шума и выбора вида срабатывания прерывания по захвату входа таймера/счетчика1. При сбросе бита ACIC соединения нет. Для запуска прерывания по захвату входа таймера/счетчика1 бит TICIE1 в регистре масок прерываний TIMSK должен быть установлен в состояние 1.
- Биты 1,0 - ACIS1, ACIS0 (*Analog Comparator Interrupt Mode Select*) - выбор режима прерывания по аналоговому компаратору. Эти биты определяют характер события компаратора, при котором запускается прерывание по аналоговому компаратору. Варианты установок показаны в таблице 11.1.

Таблица 11.1.

Установки битов ACIS1/ACIS0

ACIS1	ACIS0	Режим прерывания
0	0	Прерывание по переключению выхода компаратора
	01	Зарезервировано
1	0	Прерывание по падающему фронту на выходе компаратора
1	1	Прерывание по нарастающему фронту на выходе компаратора

При изменении состояния битов ACIS1/ACIS0 прерывание по аналоговому компаратору должно быть запрещено очисткой бита разрешения прерывания в регистре ACSR. В противном случае, при изменении состояния битов может произойти прерывание.

На отрицательный вход аналогового компаратора можно скоммутировать любой из входов порта PORTA (PA7 .. PA0). Для выбора входа используется мультиплексор аналого-цифрового преобразователя.

- Бит ACME *(Analog Comparator Multiplexer Enable)* в регистре специальных функций ввода вывода SFIOR предназначен для подключения мультиплексора к аналоговому компаратору. При подключении должен быть сброшен бит ADEN *(ADC Enable)* в регистре ADCSR (аналого-цифровой преобразователь выключен).
- Битами MUX2..0 в регистре мультиплексора ADMUX выбирается контакт на входе (табл. 5.11). Если бит ACME равен 0 или бит ADEN равен 1, то к отрицательному входу компаратора подключается контакт AIN1.

Таблица 11.2.

Логика подключения отрицательного входа компаратора.

ACME	ADEN	MUX2..0	Отрицательный вход аналогового компаратора
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	PA0
1	0	001	PA1
1	0	010	PA2
1	0	011	PA3
1	0	100	PA4
1	0	101	PA5
1	0	110	PA6
1	0	111	PA7

12. ТАЙМЕРЫ-СЧЕТЧИКИ

Большинство задач управления решаются в реальном времени. При этом микроконтроллер должен в определенные моменты времени выполнять определенные действия с объектом. Типовыми задачами такого плана считаются:

- подсчет импульсов сигнала за фиксированный интервал времени;
- формирование интервалов времени определенной длительности;
- формирование периодических сигналов заданной частоты;
- формирование широтно-модулированных сигналов;
- формирование временных задержек;
- измерение времени и др..

Любая из этих задач может быть выполнена программно, но в этом случае процессорное ядро вынуждено заниматься только подсчетом времени и все остальные задачи откладываются. Для разгрузки процессорного ядра от такой неэффективной работы микроконтроллеры снабжаются специальными схемами, получившими название таймеры/счетчики (*Timer/Counter*). Таймер/счетчик может быть использован для подсчета тактовых импульсов фиксированной частоты либо для подсчета любых внешних импульсных сигналов. Считается, что в первом случае устройство выполняет функции таймера, во втором – счетчика.

В современных микроконтроллерах, кроме простейших функций подсчета импульсов, на таймеры/счетчики возлагаются обычно дополнительные функции:

- функция входа захвата IC (*Input Capture),*
- функция выхода сравнения OC (*Output Compare*),
- широтно-импульсной модуляции PWM *(Pulse-Width Modulation*)

В режиме захвата содержимое таймера/счетчика в момент времени, задаваемый каким либо внешним событием на входе захвата, запоминается в регистре данных и становится доступным для процессорного ядра. Одновременно формируется запрос на прерывание, сообщающий программе о готовности данных.

В режиме сравнения содержимое таймера/счетчика сравнивается с некоторым фиксированным числом, хранящемся в одном из регистров микроконтроллера. В момент равенства данных формируется сигнал на выходе захвата.

В режиме широтно-импульсной модуляции таймер/счетчик формирует последовательность импульсов определенной частоты, в которой длительность импульсов может быть изменена программно от 0 до периода последовательности.

Количество таймеров/счетчиков, интегрируемых на кристалл микроконтроллера, может быть различно. Например, микроконтроллер ATmega163 имеет три универсальных таймера/счетчика: два 8-битных (*Timer/Counter0* и *Timer/Counter2*) и один 16-битный (*Timer/Counter1*).

Для регулирования частоты входного сигнала все таймеры снабжаются предделителями (*Prescaler*). Таймер/счетчик 0 и таймер/счетчик 1 имеют общий предделитель, обеспечивающий изменение частоты на входе каждого счетчика в диапазоне от 1/8 до 1/1024 входной тактовой частоты СК *(Clock).*

Тактовый сигнал таймера/счетчика 2 по умолчанию равен частоте синхронизации СК. Программно его можно подключить к отдельному тактовому генератору. Для этого к определенным выводам микроконтроллера (TOSC1 и TOSC2) подключается кристалл резонатора. Генератор оптимизирован на частоту кристалла 32.768 кГц. Это позволяет использовать таймер/счетчик 2 в качестве часов реального времени RTC (*Real Time Clock).*

12.1. Простейший 8-битный счетчик

8-разрядный таймер/счетчик 0 (*Timer/Counter0)* тактируется сигналом синхронизации процессорного ядра (СК) или от встроенного предделителя (Preskaller), или от внешнего контакта ТО (рис. 12.1).

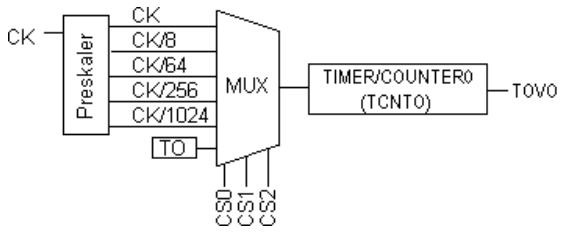


Рис.12.1. Таймер/счетчик 0 микроконтроллера ATmega163

Таймер счетчик предназначен для выполнения простейших операций: его содержимое программно доступно процессорному ядру для чтения и записи, а при переполнении счетчика вырабатывается запрос на прерывание программы.

Для управления счетчиком используются (рис. 12.2):

- регистр управления TCCR0 (*Timer/Counter0 Control Register*),
- регистр данных TCNT0 *(Timer/Counter0),*
- регистр флагов прерывания TIFR (*Timer/Counter Interrupt Flag Register*),
- регистр маски прерывания TIMSK (*Timer/Counter Interrupt Mask Register*),
- регистр состояния микроконтроллера SREG *(Status Register).*

Если таймер/счетчик осуществляет счет внешних импульсов, то последние синхронизируются сигналом синхронизации процессорного ядра.

	Биты	7	6	5	4	3	2	1	0
TCCR0	$33 ($53)	-	-	-	-	-	CSO2	CSO1	CSO0
TCNT0	$34 ($54)	MSB-	-	-	-	-	-	-	LSB
TIMSK	$39 ($59)	OCIE2	TOIE2	TICIE1	OCIE1A-	OCIE1B-	TOIE1	-	TOIE0
TIFR	$38 ($58)	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0
SREG	$3F ($5F)	I							

Рис. 12.2. Регистры, участвующие в управлении таймером/счетчиком 0

Регистр TCCR0 (*Timer/Counter 0 Control Register*) управляет тактовой частотой таймера/счетчика 0.

- Биты 2..0 - CS02, CS01, CS00 (*Clock Select0, Bit 2,1 and 0*) – задают коэффициент деления предделителя таймер/счетчика 0 и тип внешнего сигнала (табл. 12.1).

Таблица 12.1.

Коэффициенты деления предделителя таймера/счетчика 0

CS02	CS01	CS00	Описание
0	0	0	Остановка таймера
0	0	1	CK
0	1	0	CK / 8
0	1	1	CK / 64
1	0	0	CK / 256
1	0	1	CK / 1024
1	1	0	Внешний контакт T0, задний фронт
1	1	1	Внешний контакт T0, передний фронт

Timer/Counter0 является суммирующим счетчиком, доступным для чтения и записи. Регистр TCNT0 хранит содержимое счетчика.

Регистр - TIMSK (*Timer/Counter Interrupt Mask Register*) является регистром маски прерываний.

- Бит 0 - TOIE0 (*Timer/Counter0 Overflow Interrupt Enable*) – разрешение прерывания при переполнении счетчика. Когда бит TOIE0 установлен, а также установлен бит I в регистре SREG (*Status Register*), то прерывание при переполнении Timer/Counter0 разрешается. Соответствующий вектор прерывания TIMER0 OVF имеет адрес $012. Прерывание произойдет при установке бита TOV0 в регистре TIFR (*Timer/Counter Interrupt Flag Register*).

Регистр– TIFR *(Timer/Counter Interrupt Flag Register)* является регистром флагов прерывания.

- Бит 0 - TOV0 (*Timer/Counter0 Overflow Flag*) – флаг переполнения таймера/счетчика 0. Бит устанавливается при переполнении таймера/счетчика 0 и сбрасывается аппаратно при выполнении соответствующего прерывания ($012) или программно при записи 0. Прерывание происходит, если бит I в регистре SREG установлен, а также

установлен бит TOIE0 (*Timer/Counter0 Overflow Interrupt Enable*) в регистре TIMSK.

12.2. Захват, сравнение и широтно-импульсная модуляция

16-битный таймер/счетчик *Timer/Counter1* микроконтроллера *ATmega163* доступен процессорному ядру для чтения и записи, он может считать импульсы синхронизации СК, импульсы с выхода предделителя или импульсы с внешнего вывода T1. При внешнем тактировании входной сигнал синхронизируется частотой тактового генератора СК. Для правильной работы таймера/счетчика 1 с внешним тактовым сигналом минимальное время между двумя переключениями внешнего этого сигнала должно быть не менее одного периода частоты синхронизации процессорного ядра.

Шесть 16-битных регистров счетчика/таймера обеспечивают хранение данных в режимах входного захвата, выходного сравнения и широтно-импульсной модуляции. Структурная схема таймера/счетчика 1 изображена на рис. 12.3.

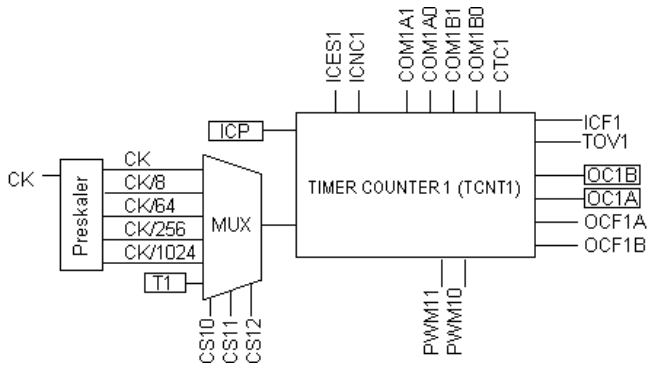


Рис. 12.3. Таймер/счетчик 1 микроконтроллера ATmega163

16-битный таймер счетчик считает импульсы со входа T1 или тактовые импульсы СК. Переключением входа управляют биты CS10….CS12. При переполнении счетчика формируется флаг запроса на прерывание TOV1.

При воздействии на вход захвата содержимое счетчика копируется в специальный регистр захвата. В этом случае устанавливается флаг запроса на прерывание ICF1.

Содержимое счетчика в процессе работы сравнивается с содержимым специальных регистров сравнения. В момент равенства формируются флаги запросов на прерывания OCF1A и OCF1B и изменяются состояния выходов OC1A и OC1B.

В режиме широтно-импульсной модуляции счетчик считает от 0 до максимального значения. Далее направление счета меняется и отсчет ведется до нуля. Содержимое счетчика сравнивается с содержимым регистров сравнения. Результат сравнения отражается на выходах OC1A и OC1B.

Работу счетчика обеспечивают 8-битные регистры, доступные процессорному ядру в пространстве регистров ввода/вывода (рис. 12.4):

- регистры управления таймера счетчика TCCR1A (*Timer/Counter1 Control Register A)* и TCCR1B *(Timer/Counter1 Control Register B);*
- регистры данных TCNT1H *(Timer/Counter1 High)* и TCNT1L (*Timer/Counter1 Low);*
- регистры выходного сравнения OCR1AH (*Timer/Counter1 Output Compare Register A High)* и OCR1AL *(Timer/Counter1 Output Compare Register A Low);*
- регистры выходного сравнения OCR1BH (*Timer/Counter1 Output Compare Register B High)* и OCR1BL *(Timer/Counter1 Output Compare Register B Low);*
- регистры входного захвата - ICR1H - (*Timer/Counter1 Input Capture Register High)* и ICR1L (*Timer/Counter1 Input Capture Register Low);*
- регистр флагов прерывания TIFR (*Timer/Counter Interrupt Flag Register*),
- регистр маски прерывания TIMSK (*Timer/Counter Interrupt Mask Register*),
- регистр состояния микроконтроллера SREG *(Status Register).*

	7	6	5	4	3	2	1	0
TCCR1A $2F ($4F)	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	PWM11	PWM10
TCCR1B $2E ($4E)	ICNC1	ICES1	-	-	CTC1	CS12	CS11	CS10
TCNT1H $2D ($4D)	MSB							
TCNT1L $2C ($4C)								LSB
OCR1AH $2B ($4B)	MSB							
OCR1AL $2A ($4A)								LSB
OCR1BH $29 ($49)	MSB							
OCR1BL $28 ($48)								LSB
ICR1H $27 ($47)	MSB							
ICR1L $26 ($46)								LSB
TIMSK $39 ($59)	OCIE2	TOIE2	TICIE1	OCIE1A-	OCIE1B:	TOIE1	-	TOIE0
TIFR $38 ($58)	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0
SREG $3F ($5F)	I							

Рис.12.4. Регистры, участвующие в управлении таймером/счетчиком 1

В регистре масок прерываний TIMSK (*Timer/Counter Interrupt Mask Register)* устанавливаются разрешения/запрещения прерываний таймера/счетчика1:

- Бит 5 - TICIE1 (*Timer/Counter1 Input Capture Interrupt Enable*) – бит разрешения прерывания при срабатывании входа захвата.
- Бит 4 - OCIE1A (*Timer/Counter1 Output CompareA Match Interrupt Enable)* – бит разрешения прерывания при равенстве содержимого счетчика и содержимого регистра сравнения OCR1A.
- Бит 3 - OCIE1B *(Timer/Counter1 Output CompareB Match Interrupt Enable)* - бит разрешения прерывания при равенстве содержимого счетчика и содержимого регистра сравнения OCR1B.
- Бит 2 - TOIE1 (Timer/Counter1 Overflow Interrupt Enable) – бит разрешения прерывания при переполнении таймера/счетчика 1.

В регистре флагов прерываний T1FR (*Timer/Counter Interrupt Flag Register*) фиксируются события, являющиеся источниками прерываний:

- Бит 5 - ICF1 *(Input Capture Flag 1)* – флаг прерывания при возникновении захвата.
- Бит 4 - OCF1A *(Output Compare Flag 1A)* – флаг прерывания при равенстве содержимого счетчика и содержимого регистра сравнения OCR1A.
- Бит 3 - OCF1B *(Output Compare Flag 1B)* - флаг прерывания при равенстве содержимого счетчика и содержимого регистра сравнения OCR1B.
- Бит 2 - TOV1 *(Timer/Counter1 Overflow Flag)* – флаг прерывания при переполнении таймера/счетчика 1.

В регистре управления TCCR1B находятся биты для переключения входа счетчика/таймера 1.

- Биты 2..0 - CS12, CS11, CS10 (Clock Select1, Bit 2,1, and 0)

Таблица 12.2.

Управление входом таймера/счетчика 1

CS12	CS11	CS10	Описание
0	0	0	Останов таймера/счетчика 1.
0	0	1	CK
0	1	0	CK / 8
0	1	1	CK / 64
1	0	0	CK / 256
1	0	1	CK / 1024
1	1	0	Внешний контакт T1, задний фронт импульса
1	1	1	Внешний контакт T1, передний фронт импульса

Регистры данных TCNT1H *(Timer/Counter1 High)* и TCNT1L (*Timer/Counter1 Low)* образуют 16-разрядный регистр, содержащий текущее значение 16-разрядного таймера/счетчика 1. Для того чтобы чтение/запись старшего и младшего байтов этого регистра происходило одновременно, обращение к ним реализовано посредством 8-разрядного регистра временного хранения (TEMP):

- Если процессорное ядро производит запись в старший байт (TCNT1H), то записываемые данные размещаются в регистре TEMP. Затем, когда процессорное ядро производит запись в младший байт (TCNT1L) данные младшего байта объединяются с байтом данных регистра TEMP и все 16 битов одновременно переписываются в регистр таймера/счетчика TCNT1. Следовательно, при 16-разрядных операциях обращение к старшему байту (TCNT1H) должно выполняться первым. При использовании таймера/ счетчика1 в качестве 8-разрядного таймера достаточно производить запись только младшего байта.
- Если CPU считывает младший байт (TCNT1L), то содержимое TCNT1L направляются непосредственно в процессорное ядро, а содержимое старшего байта (TCNT1H) размещается в регистре TEMP. При считывании старшего байта (TCNT1H) его содержимое будет изъято из регистра TEMP. Следовательно, при 16-разрядных операциях первым должно выполняться обращение к младшему байту (TCNT1L). При использовании

таймера/счетчика1 в качестве 8-разрядного таймера достаточно производить запись только младшего байта.

Если основная программа и подпрограммы обработки прерываний используют обращение к регистрам посредством TEMP, то прерывания должны быть запрещены на время обращения из основной программы.

Таймер/счетчик 1 выполнен в виде суммирующего счетчика с возможностью чтения/записи. Если в таймер/счетчик1 занесено некоторое значение и выбран источник тактового сигнала, то он через один тактовый цикл продолжит счет с записанного значения.

Вход захвата

Функция захвата заключается в копировании содержимого таймера/счетчика 1 в регистр входа захвата ICR1. Процесс копирования запускается внешним событием на входе захвата ICP. Реальные установки захвата события определяются регистром управления таймером/счетчиком 1 TCCR1B (*Timer/Counter1 Control Register*). Кроме того, для воздействия на вход захвата может быть использован аналоговый компаратор. Если разрешена функция подавления шума, действительные условия переключения события захвата тестируются, прежде чем захват будет активирован.

Регистры входа захвата ICR1H - (*Timer/Counter1 Input Capture Register High)* и ICR1L (*Timer/Counter1 Input Capture Register Low)* образуют 16-битный регистр ICR1, доступный только для чтения.

При обнаружении на входе захвата ICP нарастающего или падающего фронта сигнала (определяемого установкой бита ICES1) текущее состояние таймера/счетчика 1 пересылается в регистр входа захвата ICR1. Одновременно устанавливается в состояние 1 флаг захвата входа ICF1 *(Input Capture Flag 1)* в регистре флагов прерывания TIFR (*Timer Interrupt Flags Register).*

Поскольку регистр входа захвата ICR1 16-разрядный, то для обеспечения одновременного чтения его старшего и младшего байтов ICR1H и ICR1L используется регистр временного хранения TEMP. При считывании данных младшего байта содержимое ICR1L пересылается в процессорное ядро, а содержимое старшего байта ICR1H размещается в регистре TEMP. Чтение старшего байта заключается в переносе в процессорное ядро содержимого регистра временного хранения TEMP. Следовательно, при чтении всего 16-разрядного регистра операцию чтения необходимо начинать с младшего байта ICR1L. Регистр TEMP используется также при обращении к TCNT1, OCR1A и OCR1B. Если основная программа и подпрограммы обработки прерываний, используют обращение к регистрам посредством TEMP, то прерывания на это время должны быть запрещены. Для управления входом захвата используются отдельные биты регистра TCCR1B *(Timer/Counter1 Control Register B):*

- Бит 7 - ICNC1 (*Input Capture1 Noise Canceler*) - установка режима подавления шума на входе захвата 1. При сброшенном в состояние 0 бите ICNC1 функция подавления шума входного триггера захвата запрещена. Вход захвата в этом случае реагирует на первый же импульс, поступивший на контакт входа захвата IC1. При установленном в состояние 1 бите ICNC1 импульс, поступивший на вход захвата IC1 подвергается серьезной проверке - состояние входа IC1 опрашивается последовательно четыре раза. Все четыре выборки должны иметь одинаковый (высокий/низкий), определяемый битом ICES1, уровень. Частота опроса соответствует частоте синхронизации процессорного ядра.

- Бит 6 - ICES1 *(Input Capture1 Edge Select)* - выбор фронта срабатывания на входе захвата 1. При сброшенном в состояние 0 бите ICES1 содержимое таймера/счетчика 1 пересылается в регистр захвата входа ICR1 по падающему фронту на входе захвата IC1. При установленном в 1 бите ICES1 содержимое таймера/счетчика1 пересылается в регистр захвата входа ICR1 по нарастающему фронту на входе захвата IC1.

ВЫХОДЫ СРАВНЕНИЯ

Таймер/счетчик 1 поддерживает два выхода сравнения, используя регистры сравнения выходов A и B - OCR1A и OCR1B в качестве источников данных, сравниваемых с содержимым таймера/счетчика 1. Функции сравнения включают очистку счетчика по совпадению сравнения A и воздействие на выходы сравнения выхода при двойном совпадении.

Регистры выходов сравнения OCR1AH (*Timer/Counter1 Output Compare Register A High)*, OCR1AL (*Timer/Counter1 Output Compare Register A Low*), OCR1BH (*Timer/Counter1 Output Compare Register B High)* и OCR1BL *(Timer/Counter1 Output Compare Register B Low)* образуют два 16-разрядных регистра OCR1A и OCR1B.

Регистры OCR1A и OCR1B хранят данные, постоянно сравниваемые с состоянием таймера/счетчика 1. Действие, запускаемое совпадением при сравнении определяется содержимым регистра управления и состояния таймера/счетчика1. Совпадение может произойти, только если таймер/счетчик 1 досчитает до значения содержимого OCR1A или OCR1B. Если в TCNT1 и OCR1A или OCR1B программно будут записаны одинаковые значения, то совпадение при сравнении сгенерировано не будет.

Совпадение при сравнении устанавливает флаг прерывания по совпадению в тактовом цикле процессорного ядра, следующем за самим совпадением.

Поскольку регистры сравнения выхода OCR1A и OCR1B являются 16-битными, то для обеспечения одновременного занесения старшего и младшего байтов данных в регистры OCR1A/B используется регистр временного хранения TEMP. Когда процессорное ядро записывает старший байт, то данные временно сохраняются в регистре TEMP. Когда же записывается младший байт в OCR1AL или OCR1BL, то одновременно содержимое регистра TEMP переписывается в OCR1AH или OCR1BH. Следовательно, при 16-разрядных операциях старшие байты регистров OCR1A/B должны записываться первыми.

Кроме того, регистр TEMP используется при обращении к TCNT1 и ICR1. Если основная программа и подпрограммы обработки прерываний, используют обращение к регистрам посредством TEMP, то на это время прерывания должны быть запрещены. Для управления выходами захвата используются отдельные биты регистров TCCR1A и TCCR1B.

Регистр управления TCCR1A (*Timer/Counter 1 Control Register A*):

- Биты 7,6 - COM1A1, COM1A0 (*Compare Output Mode1A, bits 1 and 0*) – биты задания режима выхода. Определяют характер выходного сигнала при сравнении таймера/счетчика 1 с регистром OCR1A. Выходной сигнал поступает на выход OC1A (*Output Compare A*). Конфигурирование управления представлено в таблице 12.3.

Таблица 12.3.

Выбор режима сравнения

COM1X1	COM1X0	Описание
0	0	Таймер/счетчик1 отключен от вывода OC1X
0	1	Переключение выходной линии OC1X
1	0	Очистка выходной линии OC1X (на линии низкий уровень)
1	1	Установка выходной линии OC1X (на линии высокий уровень)

Примечание: X = A или B.

- Биты 5,4 - COM1B1, COM1B0 (*Compare Output Mode1B, bits 1 and 0*) – биты задания режима выхода B. Биты определяют характер сигнала на выходе при сравнении таймера/счетчика 1 с регистром OCR1B. Сигнал выхода поступает на вывод OC1B (*Output Compare B*). Конфигурирование управления представлено в таблице 5.13. При изменении битов COM1X1/COM1X0 прерывания по сравнению выхода 1 должны быть запрещены очисткой битов разрешения прерывания в регистре TIMSK. В противном случае при изменении битов может произойти прерывание
- Бит 3 - FOC1A (*Force Output Compare1A*) – воздействие на выход OC1A. Запись логической единицы в этот бит приводит к изменению на выводе OC1A относительно значений установленных битами COM1A1 и COM1A0. Если COM1A1 и COM1A0 биты занесены в том же самом цикле, что и FOC1A, то новые значения не будут вступать в силу. Бит может использоваться, чтобы изменить состояние выхода OCR1A. При этом никакое прерывание не генерируется и таймер не очищается. FOC1A бит всегда читается как нуль.
- Бит 2 - FOC1B *(Force Output Compare 1B)* воздействие на выход OC1B. Запись логической единицы в этот бит приводит к изменению на выводе OC1B относительно значений установленных битами COM1B1 и COM1B0. Если COM1B1 и COM1B0 биты занесены в том же самом цикле, что и FOC1B, то новые значения не будут вступать в силу. Бит может использоваться, чтобы изменить состояние выхода OCR1B. При этом никакое прерывание не генерируется и таймер не очищается. FOC1B бит всегда читается как нуль.

Регистр TCCR1B - (*Timer/Counter1 Control Register B):*

- Бит 3 - CTC1 (*Clear Timer/Counter1 on Compare Match*) - очистка таймера/счетчика 1 по совпадению. При установленном в состояние 1 бите CTC1 таймер/счетчик 1 при совпадении сбрасывается в состояние $0. Если бит CTC1 очищен, таймер/счетчик1 продолжает отсчет и не реагирует на совпадение при сравнении. Поскольку совпадение при сравнении детектируется в течение тактового цикла процессорного ядра следующего за совпадением, то поведение функции будет различно при различных коэффициентах предделителя. При коэффициенте предварительного деления 1 и установленном в регистре сравнения A состоянии C таймер будет считать в соответствии с установкой бита CTC1: . | C-1 | C | C+1 | 0 | 1 | ... При установленном коэффициенте предварительного деления 8 таймер

будет считать следующим образом: ... | C-1, C-1, C-1, C-1, C-1, C-1, C-1, C-1 | C, C, C, C, C,C, C, C | C+1, 0, 0, 0, 0, 0, 0, 0, 0 | ….

РЕЖИМ ШИРОТНО-ИМПУЛЬСНОЙ МОДУЛЯЦИИ

Таймер/счетчик 1 может быть использован в качестве 8, 9 или 10-разрядного широтно-импульсного модулятора (*PWM - Pulse Width Modulator)*. Работой PWM управляют отдельные биты регистра управления TCCR1A (*Timer/Counter1 Control Register A*):

° Биты 1,0 - PWM11, PWM10 *(Pulse Width Modulator Select Bits*) - биты выбора режима широтно-импульсной модуляции. Данные биты определяют установку режима ШИМ, как это показано в таблице 12.4.

Таблица 12.4.

Выбор ШИМ режима

PWM11	PWM10	Описание
0	0	Работа таймера/счетчика1 в ШИМ режиме запрещена
0	1	Работа таймера/счетчика1 в 8-разрядном ШИМ режиме
1	0	Работа таймера/счетчика1 в 9-разрядном ШИМ режиме
1	1	Работа таймера/счетчика1 в 10-разрядном ШИМ режиме

При установленном режиме широтно-импульсной модуляции таймер/счетчик 1 и регистры сравнения выхода A и B (OCR1A и OCR1B), образуют сдвоенный 8, 9 или 10-разрядный автономный генератор широтно-импульсной последовательности с выходами на выводы OC1A и OC1B. Таймер/счетчик1 в этом случае работает как реверсивный счетчик, считающий от $0000 до значения TOP (таблица 12.5), при котором направление счета меняется и отсчет ведется до нуля, после чего цикл повторяется.

Таблица 12.5.

TOP значения таймера и частота ШИМ

Разрешение ШИМ	TOP значения таймера	Частота ШИМ
8-разрядное	$00FF (255)	f_{TC1} /510
9-разрядное	$01FF (511)	f_{TC1} /1022
10-разрядное	$03FF (1023)	f_{TC1} /2046

Когда состояние счетчика совпадет с содержимым 10 младших битов OCR1A или OCR1B, выводы OC1A/OC1B устанавливаются или очищаются, в соответствии с установками битов COM1A1/ COM1A0 или COM1B1/COM1B0 в регистре управления таймером/счетчиком 1 TCCR1A (табл. 12.6).

Таблица 12.6.

Выбор функции сравнения в ШИМ режиме

COM1X1	COM1X0	Выходной сигнал на OCX1
0	0	Не подключен
0	1	Не подключен
1	0	Очищается по совпадению при счете вверх. Устанавливается по совпадению при счете вниз (не инвертированный ШИМ)
1	1	Очищается по совпадению при счете вниз. Устанавливается по совпадению при счете вверх (инвертированный ШИМ)

Примечание: X = A или B.

В PWM режиме младшие 10 разрядов регистра OCR1A/OCR1B, при записи, пересылаются в ячейки временного хранения. Они фиксируются по достижении таймером/счетчиком 1 значения TOP. Таким способом обеспечивается защита от появления ложных выбросов (*glitches*) при несинхронной записи OCR1A/OCR1B (рис. 12.5).

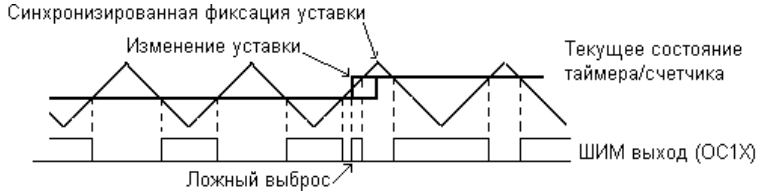


Рис. 12.5. Эффект несинхронной фиксации OCR1

При чтении OCR1A или OCR1B, в промежутке времени между записью и фиксацией, будет считано содержимое ячейки временного хранения. Это означает, что считываться из OCR1A/B всегда будет значение записанное первым.

Когда OCR1 содержит $0 или TOP, вывод OC1A/OC1B остается на низком уровне, соответственно установкам COM1A1/COM1A0 или COM1B1/COM1B0. Это отображено в таблице 12.7.

Таблица 12.7.

Состояния выходов в ШИМ режиме при OCR1X = $0 или TOP

COM1X1	COM1X0	OCR1X	Состояние выводов OC1X
1	0	$0	L
1	0	TOP	H
1	1	$0	H
1	1	TOP	L

Примечание: X = A или B.

В ШИМ режиме флаг переполнения таймера 1 (TOV1) устанавливается при смене направления счета по достижении значения $0. Прерывание по переполнению

таймера/счетчика 1 работает так же как и в обычном режиме таймера/счетчика, т.е. оно выполняется когда флаг TOV1 (*Timer Overflow Flag*) в регистре TIFR установлен, установлен бит I в регистре SREG и разрешены прерывания по переполнению таймера 1.

12.3. Часы реального времени

Часы реального времени RTC (*Real Time Clock)* являются разновидностью таймера/счетчика. Задачей RTC в схеме микроконтроллера обычно считается формирование интервалов времени равных или кратных одной секунде.

8-разрядный таймер/счетчик 2 (*Timer/Counter 2*) микроконтроллера *ATmega163* может работать от генератора тактовых импульсов процессорного ядра СК. Вместе с тем, программно его можно подключить и к отдельному тактовому генератору. Для этого к определенным выводам микроконтроллера (TOSC1 и TOSC2) подключается кристалл резонатора. Генератор оптимизирован на частоту кристалла 32768 Гц. При использовании делителя частоты с коэффициентом деления $2^{15} = 32768$ такой генератор позволяет сформировать секундные интервалы времени. 8-битный таймер/счетчик 2, снабженный 7-битным предделителем, и реализует в схеме микроконтроллера режим часов реального времени.

При внешнем тактировании таймер/счетчик работает в асинхронном режиме, независимо от тактовых импульсов СК.

Таймер/счетчик 2 снабжен выходом сравнения. В процессе работы содержимое регистра сравнения сравнивается с содержимым счетчика. В момент равенства устанавливается флаг запроса на прерывание OCF2 и модифицируется состояние выхода OC2 (рис. 12.4). Возможна его работа также и в режиме широтно-импульсной модуляции.

Запрос на прерывание TOV2 вырабатывается при переполнении таймера/счетчика 2.

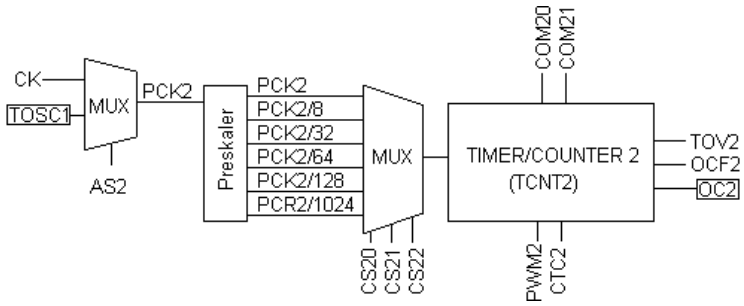


Рис. 12.4. Таймер/счетчик 2

В работе таймера/счетчика 2 задействованы следующие регистры:

- регистр управления TCCR2 *(Timer/Counter Control Register 2);*
- регистр флагов прерывания TIFR *(Timer/Counter Interrupt Flags Register);*
- регистр маски прерывания TIMSK (*Timer/Counter Interrupt Mask Register*),
- регистр состояния микроконтроллера SREG *(Status Register).*
- регистр счетчика TCNT2 (*Timer/Counter 2*)

- регистр выхода сравнения OCR2 *(Timer/Counter2 Output Compare Register)*
- асинхронный регистр состояния ASSR (*Asynchronous Status Register)*

		7	6	5	4	3	2	1	0
TCCR2	$25 ($45)	FOC2	PWM2	COM21	COM20	CTC2	CS22	CS21	CS20
TCNT2	$24 ($44)	MSB							LSB
OCR2	$23 ($43)	MSB							LSB
ASSR	$22 ($22)					AS2	TCN2UB	OCR2UB	TCR2UB
TIMSK	$39 ($59)	OCIE2	TOIE2	TICIE1	OCIE1A-	OCIE1B:	TOIE1	-	TOIE0
TIFR	$38 ($58)	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0
SREG	$3F ($5F)	I							

Рис. 12.6. Регистры таймера/счетчика 2

В регистре маски прерываний TIMSK (*Timer/Counter Interrupt Mask Register*) хранятся биты разрешения прерываний:

• бит 7 - OCIE2 (Timer/Counter2 Output Compare Match Interrupt Enable) – бит разрешения прерывания выхода сравнения.

- бит 6 - TOIE2 (Timer/Counter2 Overflow Interrupt Enable) – бит разрешения прерывания при переполнении таймера/счетчика 2.

В регистре флагов прерываний TIFR *(Timer/Counter Interrupt Flags Register)* хранятся флаги запросов на прерывания:

- бит 7 - OCF2 (Output Compare Flag 2) – флаг прерывания выхода сравнения.
- бит 6 - TOV2 (Timer/Counter2 Overflow Flag) – флаг прерывания по переполнению таймера/счетчика 2.

8-битный регистр переменной TCNT2 (*Timer/Counter 2*) содержит данные счетчика, доступен для чтения и записи. Если данные записаны в регистр и выбран источник тактового сигнала, то счет продолжается с записанного значения.

Работу счетчика в асинхронном режиме обеспечивает асинхронный регистр состояния ASSR (*Asynchronous Status Register)*.

- Бит 3 - AS2 (*Asynchronous Timer/Counter 2*) – асинхронный режим таймера/счетчика 2. Когда бит AS2 сброшен таймер/счетчик тактируется внутренним генератором СК. Когда бит AS2 установлен, таймер/счетчик 2 тактируется с контакта TOSC1.
- Бит 2 - TCN2UB (*Timer/Counter 2 Update Busy*) – бит модификации таймера/счетчика 2. Когда таймер/счетчик 2 работает асинхронно и регистр TCNT2 записывается, этот бит устанавливается. Когда регистр TCNT2 модифицируется из регистра временного хранения, этот бит аппаратно сбрасывается. Логический 0 в этом бите указывает, что регистр TCNT2 можно модифицировать.
- Бит 1 - OCR2UB *(Output Compare Register 2 Update Busy)* – бит модернизации регистра выхода сравнения. Когда таймер/счетчик 2 работает асинхронно и регистр OCR2 записывается, этот бит устанавливается. Когда регистр OCR2 модифицируется из регистра временного хранения, бит очищается. Логический 0 в этом бите указывает, что регистр OCR2 готов к модификации.

- Бит 0 - TCR2UB (*Timer/Counter Control Register2 Update Busy*) – бит модернизации регистра контроля таймера/счетчика. Когда таймер/счетчик 2 работает асинхронно и регистр TCCR2 записывается, этот бит устанавливается. Когда регистр TCCR2 модифицируется из регистра временного хранения, бит очищается. Логический 0 в этом бите указывает, что регистр TCCR2 готов к модификации.

Если запись выполняется в любой из трех регистров таймера/счетчика 2, в то время как его бит модификации установлен, модифицированное значение может быть разрушено.

Механизмы для чтения регистров TCNT2 и OCR2. При чтении TCNT2 читается фактическое значение таймера. При чтении OCR2 - читается значение во временном регистре хранения.

При переключении между асинхронным и синхронным режимами работы таймера/счетчика 2 содержимое регистров TCNT2, OCR2 и TCCR2 может быть разрушено. Безопасная процедура для переключения генератора тактовых импульсов заключается в следующем:

- Отключить прерывания таймера/счетчика 2, очищая биты OCIE2 и TOIE2.
- Выбрать генератор тактовых импульсов, воздействуя на бит AS2.
- Записать новые значения в регистры TCNT2, OCR2, и TCCR2.
- При переключении к асинхронной операции необходимо дождаться сброса TCN2UB, OCR2UB или TCR2UB.
- Разрешить прерывания, если это необходимо.

Внешний генератор оптимизирован для использования с резонатором на частоту 32.768 кГц. Подключение резонатора к внешнему контакту TOSC1 также может привести к неправильной работе таймера/счетчика 2. Частота синхронизации процессорного ядра должна быть больше частоты внешнего генератора не менее, чем в 4 раза.

При записи в любой из регистров TCNT2, OCR2, или TCCR2 данные передаются в регистры временного хранения. Каждый из трех упомянутых регистров имеет индивидуальный регистр временного хранения.

Регистр выхода сравнения OCR2 *(Timer/Counter 2 Output Compare Register)* доступен для чтения и записи, хранит константу для сравнения в счетчике. Регистр TCCR2 *(Timer/Counter2 Control Register)* содержит отдельные биты для управления выходом сравнения:

- • Бит 7 - FOC2 (*Force Output Compare*) – воздействие на выход 2. Запись 1 в этот бит приводит к изменению состояния выхода сравнения OC2 относительно значения, заданного битами COM21 и COM20. Если биты COM21 и COM20 записываются в одном цикле с битом FOC2, то новые установки не будут вступать в силу до следующего сравнения. Бит может использоваться, чтобы изменить состояние выхода OC2. При этом никакое прерывание не генерируется и таймер не очищается. FOC2 бит всегда читается как нуль.
- Бит 6 - PWM2 (*Pulse Width Modulator Enable*) – включение широтно-импульсного модулятора. Когда этот бит установлен включается режим PWM.
- Бит 5,4 - COM21, COM20 (*Compare Output Mode, bits 1 and 0*) – биты режима выхода сравнения. Биты задают режим выхода сравнения в соответствии с таблицей 12.7.

Таблица 12.7.

Режимы выхода сравнения OC2

COM21	COM20	Описание
0	0	Таймер/счетчик1 отключен от вывода OC2
0	1	Переключение выходной линии OC2
1	0	Очистка выходной линии OC2 (на линии низкий уровень)
1	1	Установка выходной линии OC1X (на линии высокий уровень)

- Бит 3 – CTC2 (*Clear Timer/Counter 2 on Compare Match*) - очистка таймера/счетчика 2 по совпадению. При установленном в состояние 1 бите CTC2 таймер/счетчик 2 при совпадении сбрасывается в состояние $0. Если бит CTC1 очищен, таймер/счетчик1 продолжает отсчет и не реагирует на совпадение при сравнении. Поскольку совпадение при сравнении детектируется в течение тактового цикла процессорного ядра следующего за совпадением, то поведение функции будет различно при различных коэффициентах предделителя. При коэффициенте предварительного деления 1 и установленном в регистре сравнения A состоянии C таймер будет считать в соответствии с установкой бита CTC2: . | C-1 | C | 0 | 1 | ... При установленном коэффициенте предварительного деления 8 таймер будет считать следующим образом: .. | C-1, C-1, C-1, C-1, C-1, C-1, C-1, C-1 | C, C, C, C, C, C, C, C | 0, 0, 0, 0, 0, 0, 0, 0 | 1, 1, 1, ...
- Биты 2,1,0 - CS22, CS21, CS20 (*Clock Select Bits 2,1, and 0*) – биты выбора тактового сигнала. Тактовый сигнал таймера/счетчика выбирается по таблице 5.19.

Таблица 12.8.

Выбор тактового сигнала таймера/счетчика 2

CS22	CS21	CS20	Описание
0	0	0	Timer/Counter2 остановлен
0	0	1	PCK2
0	1	0	PCK2 / 8
0	1	1	PCK2 / 32
1	0	0	PCK2 / 64
1	0	1	PCK2 / 128
1	1	0	PCK2 / 256
1	1	1	PCK2 / 1024

В режиме PWM таймер/счетчик 2 может работать до переполнения при значении $FF или в реверсивном режиме *(up/down)*.

Два режима PWM выбираются с помощью бита CTC2 в регистре TCCR2.Если режим PWM установлен, а бит CTC2 очищен, то таймер/счетчик 2 работает в реверсивном режиме, считая от $0 до $FF и обратно. Если бит CTC2 установлен, то таймер счетчик считает с $0 до $FF, а затем сбрасывается.

Состояние выхода определяется битами COM21/COM20. В соответствии с таблицей 12.9.

Таблица 12.9.

Режимы PWM таймера/счетчика 2

CTC2	COM21	COM20	Эффект при сравнении	Частота
0	0	0	Нет соединения	
0	0	1	Нет соединения	
0	1	0	Установка 0 при сравнении в режиме сложения; установка 1 при сравнении в режиме вычитания	$f_{TCK}/510$
0	1	1	Установка 1 при сравнении в режиме сложения; установка 0 при сравнении в режиме вычитания (инверсная PWM)	$f_{TCK}/510$
1	0	0	Нет соединения	
1	0	1	Нет соединения	
1	1	0	Установка 0 при сравнении, установка 1 при переполнении	$f_{TCK}/256$
1	1	1	Установка 1 при сравнении, установка 0 при переполнении	$f_{TCK}/256$

Когда регистр OCR1 содержит $0 или TOP, вывод OC1A/OC1B остается на низком уровне, соответственно установкам COM21/COM20. Это отображено в таблице 12.10.

Таблица 12.10.

Состояния выходов в режиме PWM при OCR2 = $00 или $FF

COM21	COM20	OCR2	Состояние выводов OC2
1	0	$00	L
1	0	$FF	H
1	1	$00	H
1	1	$FF	L

В реверсивном режиме PWM флаг переполнения TOV2 устанавливается, когда таймер/счетчик переходит в режим вычитания. В режиме работы с переполнением, он устанавливается при переполнении, как обычный таймер/счетчик. Прерывание TOV2 работает, если установлен бит I глобального разрешения прерывания в регистре статуса SREG.

13. ПОСЛЕДОВАТЕЛЬНЫЙ ВВОД-ВЫВОД

13.1. Интерфейс UART

Асинхронный последовательный интерфейс UART *(Universal Asynchronous Receiver Transmitter* – универсальный асинхронный приемопередатчик) обеспечивает полудуплексный режим обмена по трем линиям. В обмене всегда участвуют только два устройства, одно из которых является передатчиком, второе – приемником.

В режиме асинхронной передачи каждое слово данных передается автономно и передача может быть начата в любой момент времени. Стандартный формат асинхронной передачи изображен на рис. 13.1.

Рис. 13.1. Формат асинхронной передачи

Передача начинается со стартового (нулевого) бита. Затем передается от 5 до 8 бит данных. Передача заканчивается необязательным битом четного (или нечетного) паритета и одним (полутора или двумя) единичными стоповыми битами. После этого в любой момент времени может быть начат цикл передачи следующего слова.

Подразумевается, что приемник и передатчик работают на одной скорости, измеряемой числом бит в секунду (бод). Внутренний генератор синхронизации приемника запускается при обнаружении стартового бита. В идеальном случае эти импульсы располагаются в середине битовых интервалов.

Формат асинхронной передачи позволяет выявлять возможные ошибки:

- если обнаружен стартовый бит и генератор синхронизации запущен, а по первому импульсу синхронизации фиксируется уровень логической единице, стартовый бит считается ложным;
- если по импульсам синхронизации, соответствующим стоп-битам, в приемнике фиксируется логический нуль, сообщение считается ошибочным (ошибка кадра);
- если контрольный бит не соответствует принятому соглашению о паритете, фиксируется ошибка паритета.

Контроль формата позволяет обнаружить обрыв линии по отсутствию стоп-бита.

Для асинхронной передачи принят стандартный ряд скоростей: 50, 75, 110,150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 бит/с.:

В простейшем случае асинхронный приемопередатчик имеет две сигнальные линии:

- TxD *(Transmit Data)*- выход,
- RxD *(Receive Data)* – вход,

При этом два устройства-приемопередатчика должны быть соединены между собой тремя линиями, или так называемым нуль-модемным кабелем (рис.13.2).

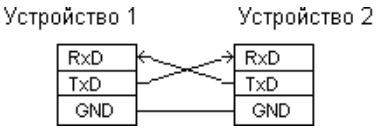


Рис. 13.2. Нуль-модемный кабель

Управление UART

Управление UART осуществляется через регистры ввода/вывода. В контроллере ATmega163 для управления используется 5 регистров (рис. 13.3):

- Регистр UDR *(UART Data Register)* – регистр данных UART
- Регистр UCSRA *(UART Control and Status Register A)* -регистр А управления и статуса UART

- Регистр UCSRB *(UART Control and Status Register B)* - регистр B управления и статуса UART
- Регистры UBRRH1 и UBRR *(UART Baud Rate registers)* – регистры скорости передачи.

		7	6	5	4	3	2	1	0
UDR	$0C ($2C)	MSB							LSB
UCSRA	$0B ($2B)	RXC	TXC	UDRE	FE	OR	-	U2X-	MPCR-
UCSRB	$0A ($2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
UBRRH1	$20 ($40)	-	-	-	-	MSB	.	.	LSB
UBRR	$09 ($29)	MSB	.	.	.	.	.	.	LSB

Рис.13.3. Регистры UART

Регистр данных UDR *(UART Data Register)* физически является двумя регистрами: регистром передачи данных и регистром приема данных, использующими одни и те же адреса $0C ($2C). При записи в регистр запись производится в регистр передачи данных UART, при чтении происходит чтение содержимого регистра приема данных UART.

Скорость обмена данными в UART задается с помощью бод-генератора (*Baud Rate Generator*). Он представляет собой делитель, генерирующий импульсы синхронизации с частотой, определяемой выражением: $BAUD = \frac{CK}{16(UBRR+1)}$, где

- BAUD = частота в бодах (бит/сек),
- CK = частота кварцевого генератора,
- UBRR = содержимое 12-битного регистра UBRR (*UART Baud Rate register*).

Физически 12-битный регистр UBRR размещается в двух 8-битных регистрах. Младшие 8 бит в регистре UBRR, старшие 4 бита – в регистре UBRRH1 (рис. 13.3).

При использовании стандартных кварцевых резонаторов, наиболее часто используемые скорости передачи в бодах могут быть получены установками UBRR, представленными в таблице 13.1. При установках UBRR, указанных в таблице, реальные скорости в бодах будут иметь отличия менее 2% от стандартных скоростей.

Таблица 13.1.

Установки UBRR при стандартных частотах синхронизации

Скорость (бод)	UBRR							
	1 MHz	Ошибка %	2 MHz	Ошибка %	4 MHz	Ошибка %	8 MHz	Ошибка %
2400	25	0,2	51	0,2	103	0,2	207	0,2
4800	12	0,2	25	0,2	51	0,2	103	0,2
9600	6	7,5	12	0,2	25	0,2	51	0,2
14400	3	7,8	8	3,7	16	2,1	34	0,8
19200	2	7,8	6	7,5	12	0,2	25	0,2
28800	1	7,8	3	7,8	8	3,7	16	2,1

Скорость (бод)	UBRR							
	1 MHz	Ошибка %	2 MHz	Ошибка %	4 MHz	Ошибка %	8 MHz	Ошибка %
38400	1	22,9	2	7,8	6	7,5	12	0,2

Фактически, для регулирования скорости передачи UART достаточно только одного регистра UDDR. В регистр UDDRH1 во всех рассмотренных случаях записывается константа $00.

Передатчик

Блок-схема передатчика UART показана на рис. 13.3.

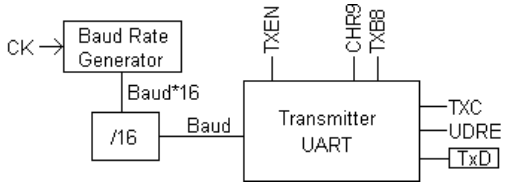


Рис. 13.3. Передатчик UART

Установленный в состояние 1 бит TXEN регистра UCSRB разрешает передачу данных UART. Передача инициируется записью передаваемых данных в регистр данных UDR. Данные пересылаются из UDR в сдвиговый регистр передачи в следующих случаях:

- Новый символ записан в UDR после того как был выведен из регистра стоповый бит предшествовавшего символа. Сдвиговый регистр загружается немедленно.
- Новый символ записан в UDR прежде, чем был выведен стоповый бит предшествовавшего символа. Сдвиговый регистр загружается после выхода стопового бита передаваемого символа, находившегося в сдвиговом регистре.

Если из 10(11)-разрядного сдвигового регистра передачи выведена вся информация (сдвиговый регистр передачи пуст) данные из UDR пересылаются в сдвиговый регистр. В это время устанавливается бит UDRE (*UART Data Register Empty*) регистра статуса USR (*UART Status Register*). При установленном в состояние 1 бите UDRE приемопередатчик готов принять следующий символ. Запись в UDR очищает бит UDRE. В то самое время, когда данные пересылаются из UDR в 10(11)-разрядный сдвиговый регистр, бит 0 сдвигового регистра сбрасывается в состояние 0 (состояние 0 - стартовый бит) а бит 9 или 10 устанавливается в состояние 1 (состояние 1 - стоповый бит). Если в регистре управления UCSRB установлен бит CHR9 (т.е. выбран режим 9-разрядного слова данных), то бит TXB8 регистра UCSRB пересылается в бит 9 сдвигового регистра передачи.

Сразу после пересылки данных в сдвиговый регистр тактом бод-генератора стартовый бит сдвигается на вывод TxD. За ним следует LSB данных. Когда будет выдан стоповый бит сдвиговый регистр загружается новой порцией данных, если она была записана в UDR во время передачи. В процессе загрузки бит UDRE находится в установленном состоянии. Если же новые данные не будут загружены в UDR до выдачи стопового бита, флаг UDRE остается установленным. В этом случае, после того как стоповый бит будет присутствовать на выводе TxD в течение одного такта, в регистре

управления и статуса UCSRA устанавливается флаг завершения передачи TxC (*TX Complete Flag*).

ПРИЕМНИК

Структурная схема приемника UART приведена на рис. 13.4.

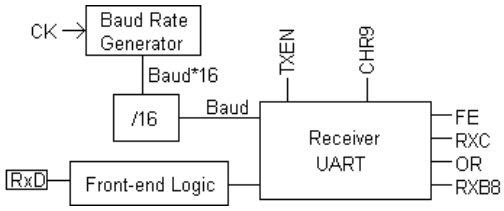


Рис. 13.4. Приемник UART

Логика восстановления данных (*Front-End Logic*) производит выборку состояний вывода RxD с частотой в 16 раз большей, чем частота передачи. При нахождении линии в пассивном состоянии одиночная выборка нулевого логического уровня будет интерпретироваться как падающий фронт стартового бита и будет запущена последовательность детектирования стартового бита. Считается, что первая выборка обнаружила первый нулевой логический уровень вероятного стартового бита. На выборках 8, 9 и 10 приемник вновь тестирует вывод RxD на изменение логических состояний. Если две или более из этих трех выборок обнаружат логические 1, то данный вероятный стартовый бит отвергается как шумовой всплеск и приемник начнет выявлять и анализировать следующие переходы из 1 в 0.

Если же был обнаружен действительный стартовый бит, то начинает производиться выборка следующих за стартовым битом информационных битов. Эти биты также тестируются на выборках 8, 9 и 10. Логическое состояние бита принимается по двум и более (из трех) одинаковым состояниям выборок. Все биты вводятся в сдвиговый регистр приемника с тем значением, которое было определено тестированием выборок.

Тестирование выборок битов принимаемых символов показано на рис. 13.5.

Рис. 13.5. Тестирование выборок принимаемых данных

При поступлении стопового бита необходимо, чтобы не менее двух выборок из трех подтвердили прием стопового бита (показали высокий уровень). Если же две или более выборок покажут состояния 0, то при пересылке принятого байта в UDR в регистре управления и статуса UCSRA устанавливается бит ошибки кадра FE (*Framing Error*). Для обнаружения ошибки кадра пользователь перед чтением регистра UDR должен проверять состояние бита FE. Флаг FE очищается при считывании содержимого регистра данных UART (UDR).

Вне зависимости от того принят правильный стоповый бит или нет, данные пересылаются в регистр UDR и устанавливается флаг RXC в регистре управления UCSRA. Регистр UDR фактически является двумя физически отдельными регистрами, один из которых служит для передачи данных и другой для приема. При считывании UDR обращение ведется к регистру приема данных, при записи обращение ведется к регистру передачи. Если выбран режим обмена 9-разрядными словами данных (установлен бит CHR9 регистра UCR), при пересылке данных в UDR бит RXB8 регистра UCR загружается из девятого бита сдвигового регистра передачи. Если после получения символа к регистру UDR не было обращения, начиная с последнего приема, в регистре UCSRA устанавливается флаг переполнения OR. Это означает, что новые данные, пересылаемые в сдвиговый регистр, не могут быть переданы в UDR и потеряны. Бит OR буферирован и доступен тогда, когда в UDR читается байт достоверных данных. Пользователю, для обнаружения переполнения, необходимо всегда проверять флаг OR после считывания содержимого регистра UDR.

При очищенном (сброшенном в логическое состояние 0) бите RXEN регистра UCR прием запрещен.

Мультипроцессорный режим обмена

Мультипроцессорный режим позволяет данные от одного ведущего контроллера передать нескольким подчиненным (ведомым) контроллерам. Это осуществляется декодированием первого адресного байта, позволяющего выяснить, который из подчиненных контроллеров адресован. Если подчиненный контроллер адресован, он примет следующие байты данных, в то время как другие подчиненные контроллеры игнорируют эти байты данных до получения другого адресного байта.

Для ведущего контроллера необходимо ввести 9-разрядный режим передачи (CHR9 в регистре UCSRB установлен). 9-ый бит должен быть равен 1, если передается адресный байт, и нулю, если передается байт данных.

Для ведомых контроллеров, механизм проявляется слегка по-другому для 8-разрядного и 9-разрядного режима приема. В 8-разрядном режиме приема (CHR9 в регистре UCSRB сброшен), стоповый бит равен 1 для адресного байта и нулю - для байта данных. В 9-разрядном режиме приема (CHR9 регистре UCSRB установлен), 9-ый бит равен 1 для адресного байта и нулю - для байта данных, принимая во внимание, что стоповый бит - всегда равен 1.

В мультипроцессорном режиме обмена должна использоваться следующая процедура:

- Все подчиненные контроллеры переводятся в мультипроцессорный режим связи (устанавливается бит MPCM в регистре UCSRA).
- Ведущий контроллер посылает адресный байт, и все ведомые принимают и читают этот байт (флажки RXC в регистрах UCSRA устанавливаются).
- Каждый ведомый контроллер читает свой регистр UDR и определяет был ли он выбран. Если так, то он очищает MPCM бит в регистре UCSRA, и далее ожидает байт данных.
- Для каждого полученного байта данных, принимающий контроллер устанавливает флажок RXC в регистре UCSRA. В 8-разрядном режиме, принимающий контроллер также генерирует ошибку кадровой синхронизации (FE в регистре UCSRA), начиная с нулевого стопового бита. Другие контроллеры, у которых бит MPCM равен 1, игнорируют байт данных. В этом случае, регистр UDR и бит RXC или FE не работают.
- После передачи последнего байта повторяется процесс пересылки адреса.

Регистры управления UART

Регистр UCSRA

- Бит 7 – RXC (UART Receive Complete) прием завершен. Данный бит устанавливается в состояние 1 при пересылке принятого символа из сдвигового регистра приема в UDR. Бит устанавливается вне зависимости от отсутствия или наличия ошибок приема кадра. При установленном в регистре UCSRB бите RXCIE и установленном бите RXC выполняется прерывание по завершению приема UART. Бит RXC очищается при считывании UDR. При приеме данных инициированном прерыванием, подпрограмма обработки прерывания по завершению приема UART должна считать UDR, с тем, чтобы очистить RXC, иначе по окончании подпрограммы обработки прерывания произойдет новое прерывание.
- Бит 6 – TXC (*UART Transmit Complete*) - передача завершена. Данный бит устанавливается в состояние 1 когда весь символ (включая стоповый бит) выведен из сдвигового регистра передачи и в UDR не записаны новые

данные. Этот флаг используется при полудуплексном связном интерфейсе, когда оборудование передачи должно установить режим приема и освободить коммуникационную шину сразу после завершения передачи. При установленном бите TXCIE установка TXC приведет к выполнению прерывания по завершению передачи UART. Флаг TXC очищается аппаратно при выполнении обработки соответствующего вектора прерывания. Очистить бит TXC можно записью логической 1.

- Бит 5 – UDRE (*UART Data Register Empty*) - регистр данных пуст. Данный бит устанавливается в состояние 1 когда символ, записанный в UDR, пересылается в сдвиговый регистр передачи. Установка этого бита означает, что передатчик готов к получению нового символа для передачи. Когда бит UDRIE установлен, до тех пор пока установлен UDRE, выполняется прерывание по завершению передачи UART. Бит UDRE очищается при записи в UDR. При приеме данных инициированном прерыванием, подпрограмма обработки прерывания по пустому регистру данных UART должна считать UDR, с тем, чтобы очистить UDRE, иначе по окончании подпрограммы прерывания произойдет новое прерывание. Во время сброса бит UDRE устанавливается в состояние 1 с тем, чтобы индицировать готовность передатчика.
- Бит 4 – FE (*Framing Error*) ошибка кадра. Данный бит устанавливается в состояние 1 при обнаружении условий ошибочного приема кадра, т.е. когда стоповый бит входящего символа в состоянии 0. Бит FE очищается при приеме стопового бита с логическим уровнем 1.
- Бит 3 – OR (*Over Run*) - переполнение данных. Бит OR устанавливается в состояние 1 при обнаружении условий переполнения, т.е. когда символ, уже находящийся в регистре UDR, не считан перед пересылкой нового символа из сдвигового регистра приема. Бит OR буферирован, что означает, что он будет оставаться установленным пока не будут считаны правильные данные из UDR. Бит OR очищается (сбрасывается в 0) когда данные приняты и пересланы в UDR.
- Бит 1 - U2X (*Double UART Transmission Speed*) – удвоение скорости передачи. Установка этого бита приводит к изменению коэффициента деления с 16 на 8, что эквивалентно удвоению скорости передачи.
- Бит 0 – MPCM (*Multi-processor Communication Mode*) – режим мультипроцессорного обмена. При установке этого бита ведомый контроллер принимает адресный байт.

Регистр UCSRB

- Бит 7 – RXCIE *(RX Complete Interrupt Enable*) - разрешение прерывания по завершению приема. При установленном в состояние 1 бите RXCIE и установленном разрешении глобального прерывания установка бита RXC в регистре UCSRA приведет к выполнению прерывания по завершению приема.
- Бит 6 – TXCIE (*TX Complete Interrupt Enable*) - разрешение прерывания по завершению передачи. При установленном в состояние 1 бите TXCIE и установленном разрешении глобального прерывания установка бита TXC в регистре UCSRA приведет к выполнению прерывания по завершению передачи.

- Бит 5 – UDRIE (*UART Data Register Empty Interrupt Enable*) - Разрешение прерывания по пустому регистру данных. При установленном в состояние 1 бите UDRIE и установленном разрешении глобального прерывания установка бита UDRE в регистре USR приведет к выполнению прерывания по пустому регистру данных UART.
- Бит 4 – RXEN (*Receiver Enable*) - разрешение приемника. Установленный в состояние 1 бит RXEN разрешает приемник UART. Если приемник запрещен, то флаги статуса TXC, DOR и FE установить невозможно. Если эти флаги установлены, то очистка бита RXEN не приведет к очистке этих флагов.
- Бит 3 – TXEN (*Transmitter Enable*) - разрешение передатчика. Установленный в состояние 1 бит TXEN разрешает передатчик UART. При запрещении передатчика во время передачи символа, передатчик не будет заблокирован прежде, чем будут полностью переданы символ в сдвиговом регистре плюс любой находящийся в UDR следующий символ.
- Бит 2 - CHR9 (*9 Bit Characters*) - режим 9-разрядных символов. При установленном в состояние 1 бите CHR9 передаются и принимаются 9 - разрядные символы плюс стартовый и стоповый биты. Девятые биты читаются и записываются с использованием битов RXB8 и TXB8 (соответственно) регистра UCSRA. Девятый бит данных может использоваться как дополнительный стоповый бит или бит контроля четности.
- Бит 1 - RXB8 (*Receive Data Bit 8)* - прием 8-разрядных данных. При установленном в состояние 1 бите CHR9 бит RXB8 является девятым битом данных принятого символа.
- Бит 0 - TXB8 (*Transmit Data Bit 8)* - передача 8-разрядных данных. При установленном в состояние 1 бите CHR9 бит TXB8 является девятым битом данных передаваемого символа.

Программирование UART

При программировании UART в простейшем случае решаются три задачи: инициализация UART(задание режимов работы), организация приема данных и организация передачи данных.

Примером инициализирующей подпрограммы может быть следующая последовательность команд:

```
            .def tmp =r20
Init_uart:  ldi tmp,0b00011101  ;Инициализация UART
            out UCR,tmp         ;TXEN=1,RXEN=1,CPH9=1,TXB8=1
            ldi tmp,25          ;9600 бит/с при fclk=4МГц
            out UBRR,tmp
            ret
```

Если микропроцессор использует последовательный канал для коротких однобайтных сообщений, то передавать и принимать данные можно, просто опрашивая флаги готовности передатчика и приемника UART. При передаче байта в последовательный канал все, что должен сделать процессор – дождаться установки флага готовности передатчика UDRE и записать затем передаваемый символ в регистр данных передатчика:

```
trans: sbis USR, UDRE       ;Если бит UDRE в USR установлен, то пропустить
                            ;следующую ;команду
```

```
        rjmp trans              ;Вернуться на метку trans
        out UDR, r15            ;Вывести в регистр данных передатчика
                                UART ;содержимое r15
        ret
```

Во время приема данных, очевидно, регистр данных приемника UART необходимо считывать лишь тогда, когда установлен флаг готовности RXC. Например:

```
receive: sbis USR, RXC      ;Если бит UDRE в USR установлен, то пропустить
                            ;следующую  команду
        rjmp receive        ;Вернуться на метку receive
in r15, UDR                 ;Считать регистр данных приемника в r15
ret
```

Или, например, так:

```
receive: sbic USR, RXC      ; Если бит RXC в USR установлен, то пропустить
                            ;следующую команду
        in r15, UDR         ;Считать регистр данных приемника UART в r15
        ret
```

В первом случае бесконечный цикл опроса флага RXC приведет к “зависанию” процессора, если ожидаемые данные так и не поступят на вход UART. Поэтому во второй подпрограмме предлагается не ждать бесконечно долго готовности приемника, а лишь один раз опросить флаг RXC. В случае, если он установлен, считать байт из регистра данных приемника, если нет – не считывать, но затем в любом случае завершить подпрограмму.

Описанные выше программы очень просты, но чрезвычайно неэффективны с точки зрения использования процессорного времени. Для передачи многобайтных сообщений процессорное ядро сможет быстро (за 0,25 мкс при Fck = 4МГц) записать в передатчик только первые два байта, а затем в течение интервала времени передачи байта в последовательный канал (при скорости 9600 бод передача одного байта занимает ~1ms) необходимо, опрашивая флаг готовности передатчика, ждать освобождения UART.

В этом отношении более рациональны программы, в которых взаимодействие быстрого процессорного ядра и медленного UART организуется с использованием системы прерываний. Пример такой организации программ для передачи данных по последовательному каналу можно найти на сайте www.atmel.ru.

13.2. Интерфейс SPI

Последовательный периферийный интерфейс SPI (*Serial Peripheral Interface*) предложен фирмой Motorola. Он обеспечивает полный дуплексный обмен данными между двумя контроллерами. При этом один контроллер считается ведущим (*master*), второй - ведомым (*slave*). Ведущий контроллер является источником сигнал синхронизации (SCK) (рис. 13.6).

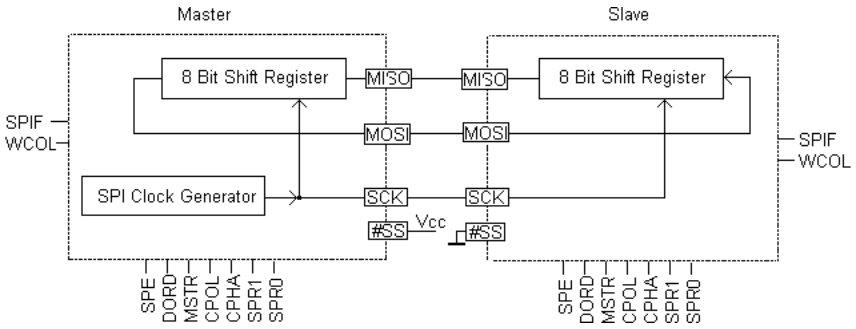


Рис. 13.6. Соединение устройств с интерфейсом SPI

Ведущий и ведомый контроллеры связаны тремя сигнальными линиями:

- MISO (*Master In Slave Out)* – вход ведущего – выход ведомого,
- MOSI *(Master Out Slave In*) – выход ведущего – вход ведомого,
- SCK (*Serial Clock*) – синхронизация,

Контакт #SS (*Slave Select*) – предназначен для выбора ведомого контроллера. Контроллер является ведомым при #SS = 0.

Регистры сдвига (*Shift Register*) ведущего и ведомого контроллеров по линиям MOSI и MOSO соединяются в кольцо. Запись в регистр данных ведущего контроллера запускает генератор синхронизации (*SPI clock generator*) и данные сдвигаются в регистрах сдвига соединенных в кольцо ведущего и ведомого контроллеров

При восьмибитных регистрах обмен длится 8 тактов. По окончании обмена генератор синхронизации останавливается и устанавливается флаг окончания передачи. Если в контроллере разрешены прерывания (бит SPIE в регистре SPCR установлен) прерывание регистрируется.

Управление SPI осуществляется через регистры ввода/вывода. В контроллере ATmega 163 такими регистрами являются:

- Регистр SPCR *(SPI Control Register*) – регистр управления SPI,
- Регистр SPSR (*SPI Status Register*) – регистр статуса SPI,
- Регистр SPDR (*SPI Data Register*) – регистр данных SPI

		7	6	5	4	3	2	1	0
SPCR	$0D ($2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
SPSR	$0E ($2E)	SPIF	WCOL	-	-	-	-	-	SPI2X
SPDR	$0F ($2F)	MSB							LSB

Рис. 13.7. Регистры SPI

Регистр данных SPDR доступен для записи и чтения. Запись в регистр инициализирует передачу данных. Чтение регистра позволяет получить результат обмена.

Регистр управления SPI содержит биты управления передачей данных.

- Бит 7 – SPIE (*SPI Interrupt Enable*) - бит маски прерывания SPI. Если бит SPIE установлен и установлен также бит I глобального разрешения прерывания в регистре статуса SREG, то прерывание разрешается.
- Бит 6 – SPE (*SPI Enable*) - бит разрешения работы SPI. Если бит SPE установлен, то SPI включен.
- Бит 5 – DORD (*Data Order*) - порядок передачи данных. Если бит DORD установлен, то младший бит данных (LSB) передается первым. Если бит DORD сброшен – первым передается старший бит данных (MSB).
- Бит 4 – MSTR (*Master/Slave Select*) - выбор ведущего/ведомого. При установке этого бита выбирается режим ведущий/ведомый. Бит устанавливается, если контроллер является ведущим и сбрасывается, если – ведомым. Если на контакт #SS подан 0, в то время как бит MSTR установлен, то бит MSTR будет очищен, и установится флаг прерывания (бит SPIF в SPSR). Пользователь в этом случае может вновь установить бит MSTR и сделать контроллер ведущим.
- Бит 3 – CPOL (*Clock Polarity*) - полярность сигнала синхронизации. Если бит CPOL установлен, то в SCK = 1 при отсутствии передачи.
- Бит 2 – CPHA *(Clock Phase)* - фаза синхронизации. Синхронизация осуществляется по переднему фронту сигнала, если бит CPHA установлен, и по заднему –если бит CPHA сброшен.
- Биты 1,0 - SPR1, SPR0 (*SPI Clock Rate Select 1 and 0*). Выбор частоты синхронизации. SPI синхронизируется от тактового генератора процессорного ядра ведущего контроллера СК. Частота передачи может быть изменена битами SPI2X регистра SPSR и SPR1 и SPR0 регистра SPCR в соответствии с таблицей 13.2. Для ведомого контроллера эти биты не существенны.

Таблица 13.2.

Частота синхронизации SPI

SPI2X	SPR1	SPR0	Частота	SPI2X	SPR1	SPR0	Частота
0	0	0	CK/ 4	1	0	0	CK / 2
0	0	1	CK/ 16	1	0	1	CK / 8
0	1	0	CK/ 64	1	1	0	CK/ 32
0	1	1	CK/ 128	1	1	1	CK/ 64

В регистре SPSR хранятся флаги прерываний и ошибок SPI.

- Бит 7 - SPIF (*SPI Interrupt Flag*) - флаг прерывания SPI. Бит устанавливается аппаратно при завершении передачи байта в SPI. Прерывание происходит, если установлен бит маски SPIE в регистре SPCR и прерывания глобально разрешены. Бит SPIF аппаратно сбрасывается при переходе по вектору прерывания.
- Бит 6 – WCOL *(Write Collision Flag*) - флаг ошибки. Бит WCOL устанавливается, если данные в регистр SPDR записываются во время процесса передачи. Бит WCOL, как и бит SPIF, сбрасывается при первом чтении регистра статуса SPSR после обращения к регистру данных SPDR.
- Бит 0 - SPI2X (*Double SPI Speed Bit)* - бит удвоения скорости передачи. Этот бит позволяет удвоить скорость передачи данных, как показано в таблице 13.2.

13.3. Интерфейс I^2C

2-проводной последовательный интерфейс (*2-Wire Serial Interface*) или I^2C (Inter-*Integrated Circuit*) предложен фирмой Philips. Он поддерживает двунаправленную последовательную связь нескольких устройств в полудуплексном режиме. В системе все устройства, участвующие в обмене, связываются двумя сигнальными линиями:

- *SDA (Serial Data)* -данные,
- *SCA (Serial Clock)* - синхронизация.

На рис. 13.8 показана типичная 2-проводная последовательная конфигурация шины.

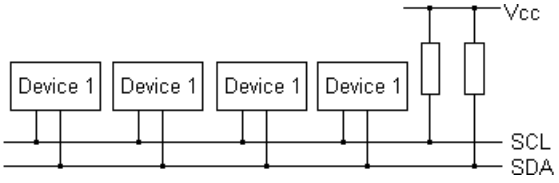


Рис.13.8. Соединение устройств с интерфейсом I^2C.

Каждое из устройств может выступать в роли передатчика или приемника. Синхронизацию обмена обеспечивает передатчик. Двунаправленную линию данных, выполненную по схеме «открытый коллектор» используют передатчик и приемник поочередно. Частота обмена ограничена сверху величиной 100 кГц для стандартного режима и 400 кГц - для скоростного. Протокол обмена иллюстрируется рис. 13.9.

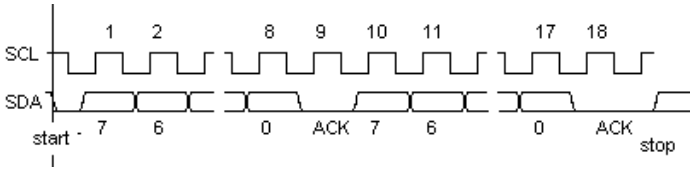


Рис. 13.9. Протокол передачи данных интерфейса I^2C

Начало любой операции (*Start*) инициализируется переводом сигнала SDA из высокого уровня в низкий при высоком уровне SCL. Завершается обмен переводом сигнала SDA из низкого уровня в высокий при высоком уровне SCL *(Stop*).

При передаче данных состояние линии SDA может изменяться только при низком уровне SCL. Биты данных стробируются положительным фронтом SCL.

Каждая посылка, формируемая передатчиком, состоит из байта данных. Посылка начинается со старшего бита. После чего передатчик на один такт освобождает линию, а приемник формирует нулевой сигнал подтверждения *Ack (Acknowledge*).

Каждое ведомое устройство имеет свой 7-битный адрес. Семь бит адреса передаются ведущим устройством в битах [7-1] первого байта, бит 0 содержит признак операции *R/W (Read/Write):*

- R/W =0 – запись,

- R/W =1 – чтение.

Старшие четыре бита адреса несут информацию о типе устройства (например, для EEPROM – 1010), а младшие три – номер устройства данного типа. К одной шине допускается подключение до восьми однотипных устройств.

Для управлениеTWSI используются регистры:

- Регистр задания скорости передачи TWBR *(Two wire Serial Interface Bit Rate Register)*;
- Регистр управления TWCR (*2-wire Serial Interface Control Regi*ster);
- Регистр статуса TWSR (*2-wire Serial Interface Status Register*);
- Регистр данных TWDR (*2-wire Serial Interface Data Register*);
- Регистр адреса (ведомого) TWAR (*2-wire Serial Interface (Slave) Address Register).*

		7	6	5	4	3	2	1	0
TWBR	$00 ($20)	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0
TWBR	$36 ($56)	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	TWIE	TWCR
TWSR	$01 ($21)	TWS7	TWS6	TWS5	TWS4	TWS3	-	-	-
TWDR	$03 ($23)	MSB	.	.	.	.	.	.	LSB
TWAR	$02 ($22)	MSB						LSB	TWGCE

Рис. 13.10. Регистры TWSI

Регистр данных TWDR (*2-Wire Serial Interface Data Reg)* хранит байт данных, для передачи или последний байт данных, полученный на 2-проводной последовательной шине.

Регистр – TWAR (*2-wire Serial Interface (Slave) Address Register*) хранит 7-битный адрес устройства на двухпроводной последовательной шине.

- Бит 0 – TWGCE (*2-wire Serial Interface General Call Recognition Enable bit*) - бит распознавания запроса. При установке бита выполняется операция чтения, при сбросе – операция записи.

Регистр TWBR (*2-wire Serial Interface Bit Rate Register)* предназначен для задания скорости передачи данных. Частота синхронизации связана с содержимым регистра соотношением:

$$BitRate = \frac{f_{ck}}{16 + 2(TWBR)}$$

- *Bit Rate* - частота SCL
- f_{ck} - тактовая частота ядра

Регистр TWCR предназначен для управления передачей данных.

- Бит 7 – TWINT (*2-wire Serial Interface Interrupt Flag*) - флаг прерывания 2-х проводного интерфейса. Бит устанавливается аппаратно, когда 2-проводной последовательный интерфейс закончил выполнение текущего задания и находится в состоянии ожидания. Если бит I в регистре статуса SREG установлен и установлен также бит TWIE в регистре TWCR, то происходит

переход на вектор прерывания TWSI с адресом $022. Флажок TWINT должен быть очищен программно.

- Бит 6 – TWEA (*2-wire Serial Interface Enable Acknowledge Fl*ag) - флаг разрешения подтверждения. Если флаг установлен, то сигнал подтверждения ACK генерируется при следующих условиях:
  - Устройством получен его адрес,
  - Общий запрос был получен, в то время как TWGCE бит в регистре TWAR установлен.
  - Байт данных был получен в главном приемнике или подчиненном режиме приемника.

  Если бит TWEA сброшен, устройство фактически разъединено с 2-проводной последовательной шиной. При повторной установке бита TWEA распознавание адреса возобновляется.
- Бит 5 – TWSTA (*2-wire Serial Bus START Condition Flag*) - флаг разрешения старта. Флаг TWSTA устанавливается, если устройство объявляется ведущим (мастером) на двухпроводной шине. Двухпроводной интерфейс аппаратно проверяет шину и разрешает установку бита, только если шина свободна. Если шина занята, интерфейс ждет конца передачи.
- Бит 4 – TWSTO (*2-wire Serial Bus STOP Condition Flag*) - флаг остановки. Если контроллер ведущий, бит TWSTO устанавливается для остановки передачи на линии. Когда условие остановки выполнено, бит TWSTO аппаратно очищается.
- Бит 3 – TWWC (*2-wire Serial Bus Write Collision Flag*) - флаг ошибки передачи. Бит TWWC устанавливается в случае попытки записи в регистр данных интерфейса при сброшенном бите TWINT. Этот флажок очищается при установке бита TWINT.
- Бит 2 – TWEN (*2-wire Serial Interface Enable Bit).* - флаг включения двухпроводного интерфейса. Бит TWEN включает двухпроводной интерфейс. Если бит сброшен, шины выводы SDA и SCL контроллера переводятся в высокоимпедансное состояние.
- Бит 0 – TWIE (*2-wire Serial Interface Interrupt Enable*) - бит маски прерывания. Если бит установлен и установлен бит I в регистре статуса SREG, то при установке флага TWINT происходит прерывание.

Регистр TWSR доступен только для чтения.

- Биты 7..3 – TWS (*2-wire Serial Interface Status*). Эти 5 битов отражают состояние логики последовательного интерфейса и 2-проводной последовательной шины. Код состояния доступен в регистре TWSR в течении одного тактового цикла после прерывания от интерфейса. Специальные таблицы позволяют определить состояние интерфейса при различных кодах состояния.

14. РЕЖИМЫ ЭНЕРГОСБЕРЕЖЕНИЯ

Одним из основных показателей микроконтроллера является энергопотребление. Величина энергопотребления характеризуется напряжением питания микроконтроллера и потребляемым током.

По напряжению питания все выпускаемые микроконтроллеры можно условно разделить на три группы:

- Микроконтроллеры с напряжением питания 5,0В±10%. Предназначены для работы в изделиях, питающихся от промышленной или бытовой сети. Как правило, это довольно мощные микроконтроллеры с развитым набором функциональных блоков.
- Микроконтроллеры с расширенным диапазоном напряжения питания. Диапазон может быть различен в пределах от 2-х до 7 В. в зависимости от изготовителя и конкретного типа. Например, микроконтроллер ATmega163 работоспособен при напряжении питания от 4 до 5,5 В, а его модификация ATmega163L – от 2,7 до 5,5В. Такие микроконтроллеры могут быть использованы как в изделиях с питанием от сети, так и с автономным питанием.
- Микроконтроллеры с пониженным напряжением питания. Напряжение питания таких микроконтроллеров обычно не превышает 3 В. Они специально разрабатываются для изделий с автономными источниками питания.

Ток, потребляемый микроконтроллером, зависит от его напряжения питания и тактовой частоты процессорного ядра. С ростом напряжения питания и с ростом тактовой частоты потребляемый ток всегда возрастает.

Большинство современных микроконтроллеров имеют несколько режимов работы, различающихся энергопотреблением. Обычно они разделяются на две группы:

- Активный режим *(Run Mode)*. Обычный рабочий режим, в котором микроконтроллер выполняет заложенную в него программу и включены все его функциональные блоки.
- Режимы энергосбережения (*Sleep Mode*). Режимы, в которых отключаются те или иные функциональные блоки микроконтроллера и его энергопотребление существенно снижается. Режимы энергосбережения, в свою очередь, могут быть различны, например:
  - Режим холостого хода (*Idle mode*). Останавливается процессорное ядро, но продолжают работу некоторые функциональные блоки, связывающие микроконтроллер с системой. Через эти блоки процессорное ядро может быть вновь переведено в активный режим.
  - Режимы ожидания (*Wait mode* или *Power Down Mode*). Останавливается процессорное ядро и отключается большинство функциональных блоков. Процессорное ядро может быть переведено в активный режим только через систему сброса или систему прерываний.

Для перевода микроконтроллера в тот или иной режим энергосбережения в системе команд предусматриваются специальные инструкции, а в его архитектуре – специальные регистры. Например, у AVR-микроконтроллеров фирмы Atmel для перехода в режимы энергосбережения используется инструкция Sleep, а задание режима осуществляется через регистр MCUCR *(MCU Control Register)* (рис. 14.1).

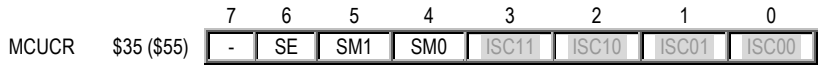


Рис. 14.1. Биты управления режимами энергосбережения микроконтроллера ATmega163

В активном режиме при напряжении питания 3В и тактовой частоте 4 МГц микроконтроллер потребляет ток 5 мА. Для снижения энергопотребления в нем предусмотрено 4 различных режима. Для перевода в любой из этих режимов энергосбережения бит SE (*Sleep Enable)* в регистре MCUCR должен быть установлен в состояние 1. Биты SM1 и SM0 (*Sleep Mode Select bits 1 and 0*) регистра MCUCR определяют, какой из четырех режимов:

- *Idle,*
- *ADC Noise Reduction,*
- *Power Down* или
- *Power Save*

будет запущен командой SLEEP. Выбор режима осуществляется в соответствии с таблицей 14.1.

Таблица 14.1.

Выбор режимов энергосбережения микроконтроллера *ATmega163*

SM1	SM0	Режим энергосбережения
0	0	Холостого хода (Idle)
0	1	Шумоподавления (ADC Noise Reduction)
1	0	Ожидания (Power-down)
1	1	Экономии (Power Save)

В любом из четырех режимов процессорное ядро контроллера останавливается. При возникновении разрешенного прерывания во время режима энергосбережения, ядро включается, выполняет подпрограмму обработки прерывания и продолжает работу в активном режиме. Если во время режима энергосбережения происходит сброс, ядро также активизируется и начинает работу по вектору сброса. Содержимое регистров общего назначения, памяти данных и регистров ввода/вывода в процессе активизации не изменяется.

14.1. Режим Idle

Если биты SM1/SM0 находятся в состоянии 00, то команда SLEEP переводит микроконтроллер в режим ожидания *Idle*. В этом режиме его ток потребления уменьшается примерно в 2,5 раза (при Vcc = 3В и Fск= 4 Мгц ток равен примерно 1,9 мА). При этом останавливается процессорное ядро, но остаются активными таймеры/счетчики, сторожевой таймер и система прерываний. Это обеспечивает последующую активизацию ядра внешними прерываниями и такими внутренними прерываниями, как переполнение таймера и завершение приема UART. При активизации микроконтроллера из *Idle* режима программа начинает выполняться незамедлительно.

14.2. Режим ADC Noise Reduction

Когда SM1/SM0 биты установлены в 01, команда SLEEP заставит микроконтроллер ввести режим шумоподавления *ADC Noise Reduction*. В этом режиме процессорное ядро останавливается, но разрешается работа аналого-цифрового преобразователя, внешних прерываний, 2-проводного последовательного интерфейса, таймера/счетчика и сторожевого таймера.

Режим ADC Noise Reduction улучшает шумовую среду для аналого-цифрового преобразователя, повышая его разрешающую способность. Аналого-цифровой преобразователь при введении этого режима запускается автоматически.

Активизация процессорного ядра происходит при прерываниях от аналого-цифрового преобразователя, от двухпроводного интерфейса и любом сбросе микроконтроллера.

14.3. Режим Power Down

При установке битов SM1/SM0 в состояние 10 команда SLEEP переводит микроконтроллер в режим останова *Power Down*. В этом режиме его ток потребления менее 1 мкА. При этом останавливается внешний генератор. Пользователь может разрешить работу сторожевого таймера. Если сторожевой таймер разрешен, то активизация процессорного ядра произойдет по завершении установленного в сторожевом таймере периода времени.

14.4. Режим Power Save

При установке битов SM1/SM0 в состояние 11 команда SLEEP переводит микроконтроллер в режим экономии *Power Save*. Этот режим аналогичен режиму *Power Down*, но если таймер/счетчик 0 тактируется асинхронно, т.е. бит AS0 в регистре ASSR установлен, то в этом режиме таймер/счетчик 0 будет работать и микроконтроллер будет активироваться прерываниями по переполнению или совпадению с выхода таймера/счетчика 0.

15. ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ

15.1. Способы программирования энергонезависимой памяти

В процессе программирования микроконтроллеров разработанная пользователем программа заносится в энергонезависимую память. При этом выполняются операции по стиранию, чтению и записи различных элементов памяти кристалла. Типовыми операциями являются:

- операция “*Chip erase*” (стирание кристалла);
- чтение/запись FLASH-памяти программ;
- чтение/запись EEPROM памяти данных;
- чтение/запись конфигурационных FUSE-битов;
- чтение/запись LOCK-битов защиты программной информации;
- чтение SYGNATURE-битов идентификации кристалла;

Микроконтроллеры обычно поставляются со стертыми встроенными FLASH и EEPROM блоками памяти (содержимое всех ячеек = $FF), готовыми к программированию. Программирование кристалла, как правило, может выполняться различными способами:

- Параллельное программирование – информация заносится в энергонезависимую память через параллельные порты ввода/вывода. Обычно эта операция выполняется на специальных приборах – программаторах.
- Последовательное программирование – информация заносится в энергонезависимую память через схему последовательного интерфейса. Операция, как правило, может выполняться непосредственно в микропроцессорной системе (*In System Programming*).
- Самопрограммирование может осуществляться самим микроконтроллером под управлением программы, записанной в *Boot Program Section* флэш-памяти программ.

При параллельном программировании информация поступает на микроконтроллер в параллельном коде через один из его портов. Ряд других контактов микроконтроллера используется для управления записью и чтением данных. На рис. 15.1. показана схема подключения микроконтроллера *ATmega163* при параллельном программировании.

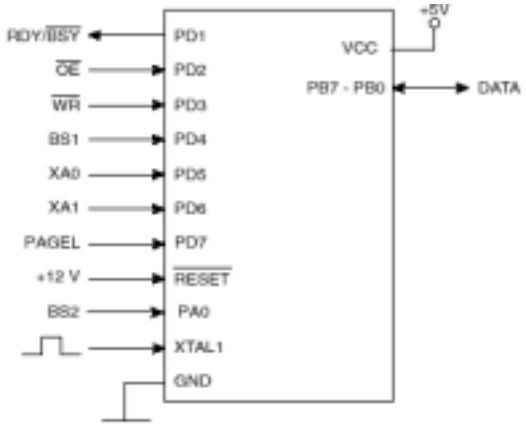


Рис. 15.1. Параллельное программирование микроконтроллера ATmega163

Для перевода устройства в режим параллельного программирования необходимо:

- Подать напряжение 4.5 - 5.5V на контакт Vcc.
- Установить на контактах #RESET, BS0 и BS1 сигнал низкого уровня на время не менее 100 нс..
- Подать напряжение 11.5 - 12.5V на контакт #RESET.

В каждом такте программирования импульс синхронизации длительностью не менее 500 нс подается на контакт XTAL1. Совокупность остальных сигналов (табл. 15.1 и табл. 15.2) описывает операцию, выполняемую в момент прихода синхроимпульса. В зависимости от сочетания сигналов информация на линиях порта B микроконтроллера, может быть восприняты как команда управления процессом записи, как данные или как один из байтов адреса ячейки памяти.

Таблица 15.1.

Назначение сигналов при параллельном программировании

Наименование сигнала в режиме программирования	Контакт	Вход/ выход (I/O)	Функция сигнала
RDY/BSY	PD1	O	0: Программирование, 1: Чтение команды
OE	PD2	I	Включение выхода (активный низкий уровень)
WR	PD3	I	Импульс записи (активный низкий

			уровень)
BS1	PD4	I	Выбор байта 1 (‘0’ – младший байт, ‘1’ – старший байт)
XA0	PD5	I	Операция (бит 0)
XA1	PD6	I	Операция (бит 1)
PAGEL	PD7	I	Загрузка страницы памяти программ
BS2	PA0	I	Выбор байта 2 (‘0’ – младший байт, ‘1’ – старший байт)
DATA	PB7	0	Двунаправленный ввод/вывод (вывод при OE=0)

Таблица 15.2.

Выбор операции при параллельном программировании

XA1	XA0	Операция при импульсе на входе XTAL1
0	0	Загрузка адреса Flash или EEPROM памяти (байт адреса задается сигналом BS1)
0	1	Загрузка данных (байт адреса задается сигналом BS1
1	0	Загрузка команды
1	1	Ожидание

В таблице 15.3 приведены команды параллельного программирования микроконтроллера *ATmega163*, различаемые микроконтроллером при выполнении операции «Загрузка команды» (XA1=1, XA0=0).

Таблица 15.3.

Команды параллельного программирования микроконтроллера *ATmega163*

Байт	Команда
1000 0000	Очистка кристалла
0100 0000	Запись Fuse битов
0010 0000	Запись Lock битов

Окончание табл. 15.3.

Байт	Команда
0001 0000	Запись Flash памяти
0001 0001	Запись EEPROM
0000 1000	Чтение байта Signature
0000 0100	Чтение Fuse и Lock битов
0000 0010	Чтение Flash памяти
0000 0011	Чтение EEPROM

FLASH и EEPROM блоки памяти программируются байт за байтом.

Во время операции очистки кристалла (*Chip erase)* стираются все ячейки FLASH и EEPROM памяти, а также LOCK-биты. Причем LOCK-биты стираются только после того, как будет очищена вся память программ. На состояние FUSE-бит операция *Chip erase* не оказывает воздействия.

У контроллера *ATmega163* имеется FUSE бит EESAVE, программирование которого приводит к тому, что при выполнении операции *Chip erase* содержимое EEPROM сохраняется.

Последовательное программирование

При последовательном программировании значительно меньше контактов микроконтроллера и не требуется источника высокого напряжения. На рис. 15.2 показана схема включения микроконтроллера в режиме последовательного программирования.

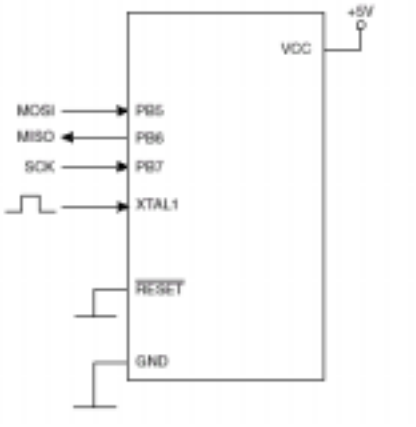


Рис. 15.2. Последовательное программирование микроконтроллера *ATmega163*

Здесь для программирования используется последовательный SPI интерфейс при подключении контакта Vcc к источнику питания 5В, а контактов GND и #RESET – к земле. Последовательный интерфейс использует контакты SCK (синхронизация), MOSI (ввод) и MISO (вывод).

При записи последовательно поступающих данных в микроконтроллер данные синхронизируются по переднему фронту сигнала SCK. При чтении данных с микроконтроллера данные синхронизируются по заднему фронту SCK.

Для программирования и проверки микроконтроллера ATMEGA163 в последовательном режиме программирования используются четырехбайтные команды (табл. 15.4).

Таблица 15.4.

Инструкции последовательного программирования микроконтроллера *ATmega163*

Команда	Формат команды				Операция
	Байт 1	Байт 2	Байт 3	Байт 4	

Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Включение последовательного программирования после сброса импульсом #RESET
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Очистка кристалла EEPROM и Flash.
Read Program Memory	0010 H000	xxxa aaaa	bbbb bbbb	oooo oooo	Чтение байта H (старший или младший) команды из памяти программ с заданным адресом a:b.
Load Program Memory Page	0100 H000	xxxx xxxx	xxbb bbbb	iiii iiii	Загрузка байта H (старший или младший) данных i в страницу памяти программ по адресу b.
Write Program Memory Page	0100 1100	xxxa aaaa	bbxx xxxx	iiii iiii	Запись страницы памяти программ с адресом a:b.
Read EEPROM Memory	1010 0000	xxxx xxxa	bbbb bbbb	oooo oooo	Чтение данных из EEPROM по адресу a:b.
Write EEPROM Memory	1100 0000	xxxx xxxa	bbbb bbbb	iiii iiii	Запись i в EEPROM по адресу a:b.
Read Lock Bits	0101 1000	xxxx xxxx	xxxx xxxx	xx65 4321	Чтение Lock bits. ‘0’ = programmed, ‘1’ = unprogrammed.
Write Lock Bits	1010 1100	111x xxxx	xxxx xxxx	1165 4321	Запись Lock bits. Установка бит 6 - 1 = ’0’ приводит к программированию Lock bits

15.2. Программно-аппаратные средства поддержки программирования

Подготовка программ для микроконтроллера выполняется на персональном компьютере и состоит из следующих этапов:

- создание текста программы;
- трансляция текста в машинные коды и исправление синтаксических ошибок;
- отладка программы, то есть устранение логических ошибок;
- окончательное программирование микроконтроллера.

Каждый из этапов требует использования специальных программных и аппаратных средств. Программные и аппаратные средства разрабатываются как производителями микроконтроллеров, так и независимыми фирмами. Они всегда ориентированы на конкретную архитектуру микроконтроллера. Программные средства обычно оформляются в

виде интегрированной отладочной среды и могут распространяться бесплатно, условно-бесплатно или на платной основе.

Текст программы, создаваемый на первом этапе проектирования оформляется в виде файла на языке ассемблера (с расширением .asm). Этот файл является входным для программ-трансляторов, которые, в свою очередь, создают новые файлы, ориентированные на использование с конкретными отладочными средствами. Обычно это:

- файл-листинг (с расширением .lst),
- объектный файл (с расширением .obj),
- файл-прошивка FLASH-памяти (с расширением .hex),
- файл-прошивка EEPROM-памяти (с расширением .eep).

Файл-листинг - это отчет транслятора о своей работе. В нем приводится транслируемая программа в виде исходного текста, каждой строке которого сопоставлены соответствующие двоичные коды. Кроме того, листинг содержит сообщения о выявленных ошибках.

Объектный файл, создаваемый программой ассемблером, используется в дальнейшем как входной для программы-отладчика и имеет специальный формат.

Файлы прошивки FLASH и EEPROM блоков памяти предназначены для работы с последовательными и параллельными программаторами и имеют стандартные форматы.

Кроме языка ассемблера, для программирования встраиваемых микропроцессоров широкое распространение получили языки программирования высокого уровня. Они предоставляют программисту такой же легкий доступ ко всем ресурсам микроконтроллера, как и ассемблер, но вместе с тем дают возможность создавать хорошо структурированные программы, снимают с программиста заботу о распределении памяти данных и содержат большой набор библиотечных функций для выполнения стандартных операций.

Отладка программ микроконтроллеров может выполняться двумя основными способами: на персональном компьютере при помощи программы-симулятора или в реальной микропроцессорной системе.

Два эти способа взаимно дополняют друг друга.

Программы-симуляторы отображают на экране компьютера программу пользователя и состояние внутренних регистров микроконтроллера. В результате, появляется возможность для наблюдения за изменениями в регистрах, памяти и процессорном ядре микроконтроллера при выполнении тех или иных команд программы. В реальной системе состояние внутренних регистров микроконтроллера посмотреть при помощи измерительных приборов невозможно. Использование симуляторов эффективно при отладке подпрограмм, которые выполняют численную обработку внутренних данных.

Внутрисхемные эмуляторы являются специальными схемами, которые с одной стороны связываются с проектируемой микропроцессорной системой через панель, предназначенную для установки микроконтроллера, а с другой - с персональным компьютером и работают под управлением установленного на компьютере программного обесечения. Внутрисхемные эмуляторы позволяют выполнять программу в системе в пошаговом режиме и неограниченное число раз вносить изменения в программу. При работе с внутрисхемным эмулятором на экране компьютера можно наблюдать состояние внутренних ресурсов процессора, а на микропроцессорной плате - реакцию системы на те или иные команды программы.

Окончательная отладка программного обеспечения производится в рабочей системе. Обычно производители микроконтроллеров предлагают пользователям различные аппаратные средства для создания такой системы. Например, для разработки

микропроцессорных систем на основе AVR-микроконтроллеров фирма Atmel выпускает стартовый набор AVR STARTER KIT типа STK500.

Набор STK500 содержат небольшую печатную плату *Development Board*, кабель для последовательного программирования через COM-порт компьютера, CD-ROM с полной документацией на все типы микроконтроллеров и многочисленными примерами прикладных программ (рис. 15.1).

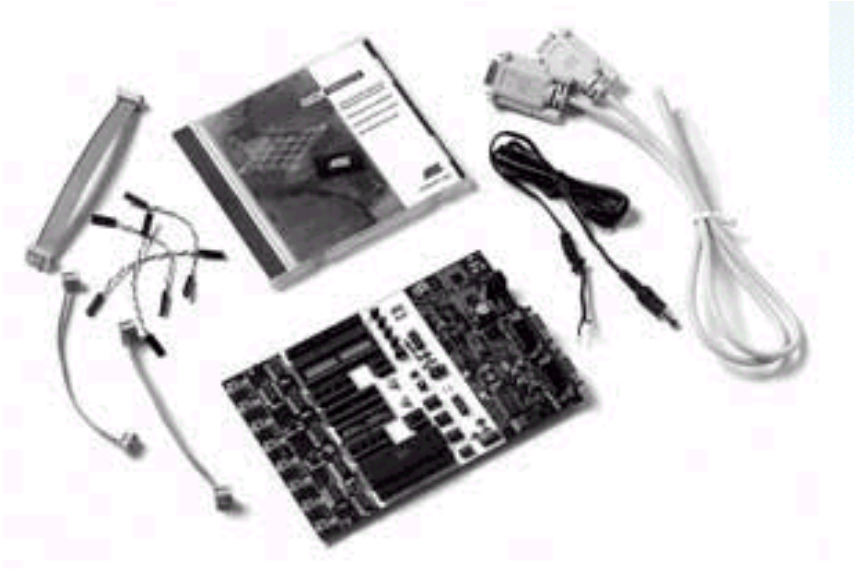


Рис. 15.1. Набор STK500 фирмы Atmel

Плата *Development Board* из набора STK500 (рис. 15.2) имеет следующие узлы:

- встроенный стабилизатор напряжения питания;
- сокеты для установки различных типов микроконтроллеров в DIP-корпусах;
- разъем для последовательного программирования*;*
- узел интерфейса RS-232;
- набор из 8 светодиодов, которые можно подключать к выводам портов микроконтроллера;
- набор из 8 кнопочных переключателей, которые можно подключать к выводам портов микроконтроллера;
- разъемы, через которые при помощи гибких кабелей можно наращивать микропроцессорную систему. Например, можно подключить собственный макет аналоговой части какого-либо устройства.

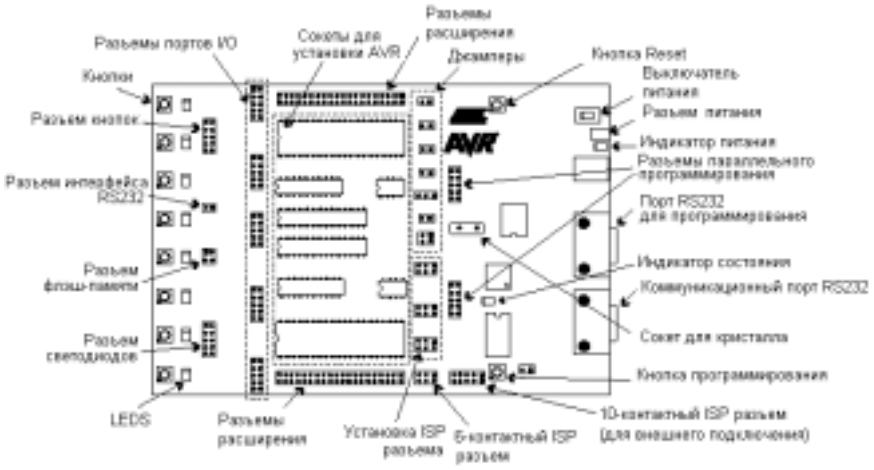


Рис. 15.2. Плата Development Board из набора STK500

Вместе с тем, для многих конкретных проектов платы *Development Board* будут избыточными. В таком случае выполняют специализированную разработку, удовлетворяющую требованиям конкретной задачи. В этом случае кроме собственного макета необходимо иметь еще какое-либо программирующее устройство.

Завершающим этапом программирования микроконтроллера является занесение в память уже отлаженной программы. Оно может быть выполнено так же, как и при отладке программы, то есть через SPI-интерфейс. Если в системе SPI-интерфейс не предусмотрен, то необходимо использовать программаторы, которые выполняют параллельное программирование. Параллельные программаторы обычно являются универсальными устройствами и позволяют работать и с различными микроконтроллерами.

15.3. Интегрированная отладочная среда

Интегрированная отладочная среда обычно включает в себя компилятор с языка ассемблера и позволяет пользователю полностью контролировать выполнение программ с использованием симулятора, который поддерживает различные типы микроконтроллеров.

Например, отладочная среда AVR Studio фирмы Atmel поддерживает выполнение программ в виде ассемблерного текста формата AVR Assembler, IAR Systems' Assembler и в формате языка C компилятора фирмы IAR Systems' ICCA90 C Compiler.

Пользователь может выполнять программу полностью в пошаговом режиме, трассируя блоки функций, или выполняя программу до места, где стоит курсор. В дополнение можно определять неограниченное число точек останова, каждая из которых может быть включена или выключена.

Интегрированная отладочная среда AVR Studio может использоваться с встроенным имитатором AVR (*AVR Simulator*) или с отдельным внутрисхемным эмулятором

AVR (*In-Circuit Emulator*). Когда пользователь запускает *AVR Studio*, программа проверяет наличие эмулятора на одном из последовательных портов компьютера. Если эмулятор найден, он выбирается как базовая система отладки. Если эмулятор не найден, то отладка будет выполняться на встроенном имитаторе AVR.

Независимо от того какая базовая система отладки функционирует, среда AVR Studio не меняется. При переключении между базовыми системами отладки, все параметры среды сохраняются.

Вместе с тем, в режиме реального времени эмулятор работает существенно быстрее, чем имитатор. Он также позволяет производить отладку, в то время как система соединена с реальной аппаратной средой.

Использование той или иной базовой системы отображается на строке состояния. Строка появляется при любых переключениях между базовыми системами.

Создание проекта в AVR-Studio

Для работы над новым проектом создается папка проекта с произвольным именем на диске C:. В папку необходимо скопировать файл инициализации выбранного для реализации проекта контроллера, например, "m163def.inc". Этот файл включен в прикладное программное обеспечение AVR studio версии 3.2 или выше и может быть скопирован из папки *AVR Studio\Appnotes.*

Программа AVR Studio версии 3.2 и выше функционирует в среде Windows 95/98/2000/NT. Значок запуска программы изображен на рис. 15.3.

Рис. 15.3. Значок программы AVR Studio

В результате запуска AVR Studio на экране появляется окно программы.

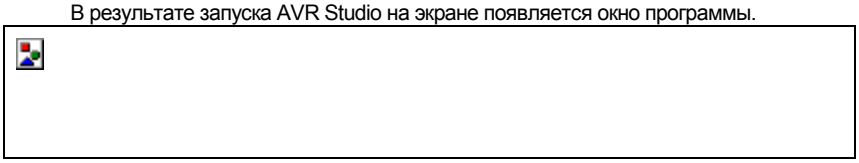


Рис. 15.4. Стартовое окно программы AVR Studio

Для создания нового проекта необходимо в меню **Project** выбрать команду **New** .В результате на экране появляется диалоговое окно (рис. 15.5), в окне которого необходимо ввести название проекта (*Project name*) и его расположение (*Location*).

Рис. 15.6. Диалоговое окно нового проекта

Далее выбирается тип проекта:

- **AVR Assembler** использует язык ассемблера. Никакой дальнейшей конфигурации при этом не требуется.
- **Generic 3D party C compiler** использует язык С: опция This позволяет пользователю вручную конфигурировать AVR Studio, чтобы использовать внешний компилятор при создании проекта.

При выборе **AVR Assembler** и нажатии кнопки OK на экране появляется окно организатора проекта (рис. 15.7), показывающее все связанные с ним файлы.

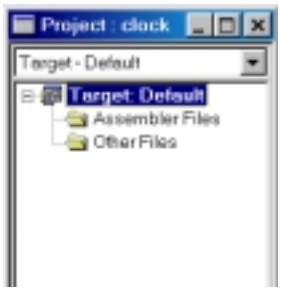


Рис. 15.7. Окно организатора проекта

Далее к проекту должен быть добавлен файл программы на языке ассемблера. Это можно сделать разными способами: или в проект добавляется уже существующий файл с расширением .asm или создается новый.

Для создания нового файла необходимо в организаторе проекта выбрать **Assembler Files** и в меню **Project** выбрать пункт **Add File**..В результате откроется диалоговое окно Open (рис. 15.8).

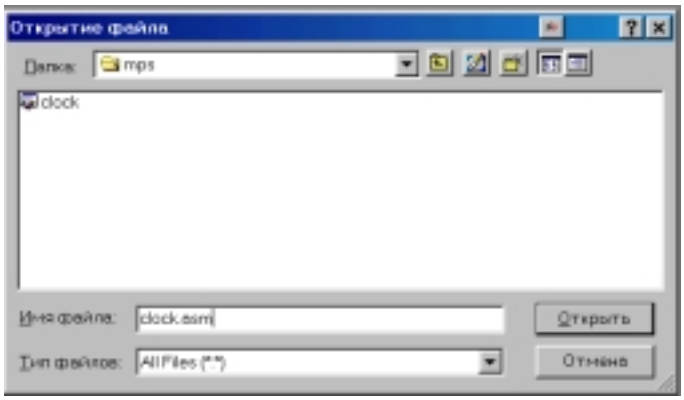


Рис. 15.8. Диалоговое окно открытия файла

В открывшемся окне необходимо ввести название файла с расширением .asm. Файл будет создан и помещен в выбранной ранее папке.

Если файл был создан ранее, то его необходимо найти на диске и двойным щелчком мыши занести в строку «Имя файла».

Созданный или найденный таким образом файл будет помещен в папку **Assembler Files** в окне организатора проекта (рис. 15.9). С этого файла в дальнейшем начнется трансляция проекта. Папка **Assembler Files** может содержать только один файл. Значок этого файла в окне организатора проекта отмечен красной стрелкой вправо, все другие файлы в дальнейшем будут отмечаться синими стрелками, направленными вниз.

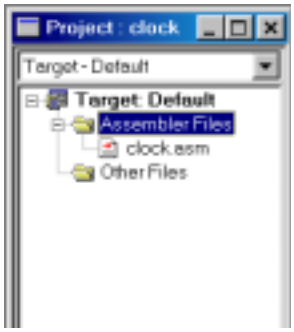


Рис. 15.9. Файл программы на языке ассемблера в окне организатора проекта

Для редактирования программы необходимо в папке **Assembler Files** в окне организатора проекта открыть созданный файл с расширением .asm. В открывшееся окно для редактирования файла можно с клавиатуры или через буфер компьютера ввести текст программы на языке ассемблера (рис. 15.10).

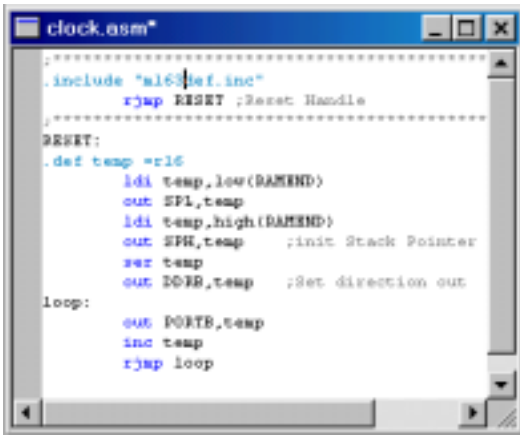


Рис. 15.10. Окно редактирования программы на языке ассемблера

Далее осуществляется трансляции программы и проверка правильности её написания. Выбирается пункт **Assemble** в меню **Project**. Открывшееся окно **Project Output** содержит сообщения ассемблера. В окне (рис. 15.11) сообщается о количестве слов программы и данных, о наличии ошибок.

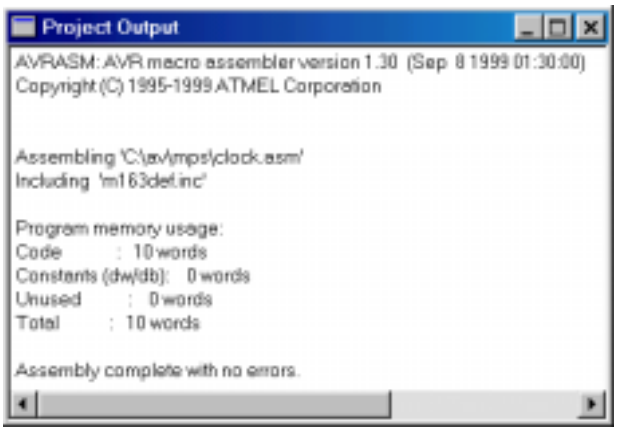


Рис. 15.11. Окно результатов трансляции программы

Для запуска созданной и откомпилированной программы используется команда **Trace into** в меню **Debug**. На экране появляется диалоговое окно выбора опций симулятора (*Simulator Options*) (рис. 15.12).

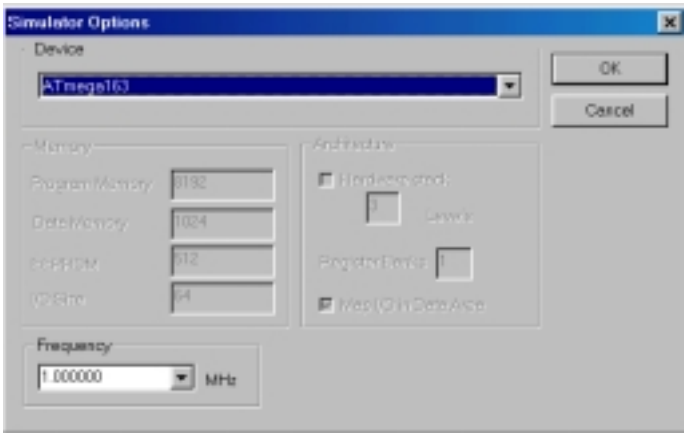


Рис. 15.12. Диалоговое окно выбора опций симулятора

В диалоговом окне из списка выбирается нужный микроконтроллер (*Device*), например *ATmega163*, и тактовая частота процессорного ядра (*Frequency*). Опции *Memory* и *Architecture* при выборе стандартного устройства в окне не используются.

После выбора опций симулятора в левом поле окна ассемблерной программы появляется желтая стрелка, указывающая на позицию программного счетчика микроконтроллера (рис. 15.13).

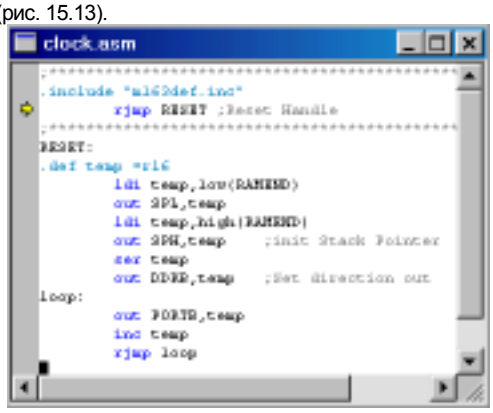


Рис. 15.13. Установка программного счетчика в тексте программы

В AVR Studio для отладки программы предусмотрены две команды шагового режима: **Step Over** и **Trace into**. Разница между ними в том, что команда **Step Over** не работает в подпрограммах. С помощью команд шагового режима в окне IO Window можно проследить изменения в регистрах устройств ввода/вывода. При этом регистры, в которых происходят изменения, отмечаются красным цветом.

Помимо шагового режима возможна отладка программы в режиме непрерывном. В этом случае для просмотра содержимого регистров и ячеек памяти в AVR Studio предусматриваются контрольные точки (*Breakpoints*). Командой **Go** или кнопкой запускается исполнение программы. Программа будет выполняться до остановки пользователем или до обнаружения в ней контрольной точки. Для установки контрольной точки в программе предусмотрен пункт **Toggle Breakpoint** в меню **Breakpoints.** Контрольная точка ставится в строке, отмеченной курсором (рис. 15.14).

Рис. 15.14. Установка контрольной точки в программе

Красная площадка в левом поле исходного окна программы показывает установленную контрольную точку.

Команда **Break** останавливает выполнение программы. Когда выполнение остановлено, вся информация во всех окнах изменяется. Команда доступна только при выполнении программы.

При отладке программы также можно воспользоваться командой **Run To Cursor** меню Debug. Эта же команда может быть запущена кнопкой . При таком запуске программы она выполняется до достижения команды, обозначенной курсором. Если программа сталкивается с контрольной точкой пользователя, выполнение не приостанавливается. Если команда, обозначенная курсором, не достигается, программа выполняется до остановки пользователем. После выполнения команды вся информация во всех окнах модифицируется. Поскольку команда Run To Cursor зависит от позиции курсора, она доступна только в активном окне Source.

Команда **Reset** в меню **Debug** и кнопка выполняют сброс микроконтроллера. Если программа при этом выполняется, то исполнение будет остановлено. После сброса вся информация во всех окнах модифицирована.

Для наблюдения за работой программы можно открыть диалоговое окно устройств ввода/вывода. Окно открывается нажатием кнопки (IO Window) на панели инструментов.

В диалоговом окне отражаются все функциональные блоки микроконтроллера. При этом любой блок может быть раскрыт нажатием на его значок. При раскрытии блока в окне отражаются все его регистры и отдельные, доступные программе, биты. Каждый бит в регистрах представлен переключателем. Его нулевое состояние представлено значком с номером бита: , а единичное - отмеченным переключателем: . (рис. 15.16).

В окне также отражаются адреса всех регистров и отдельных битов, участвующих в работе раскрытого устройства ввода/вывода.

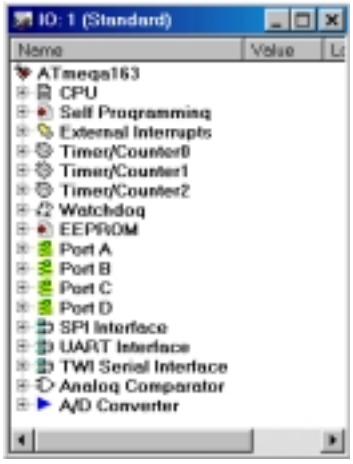


Рис. 15.15. Диалоговое окно устройств ввода вывода

Рис. 15.16. Развернутый порт PORTB в окне устройств ввода/вывода

Переключатели, отражающие состояние отдельных битов могут изменяться в процессе работы программы. Они отражают текущее состояние каждого бита. Пользователь, в процессе отладки программы, может также устанавливать и очищать эти биты, нажимая на соответствующий переключатель в любое время во время выполнения программы.

Для наблюдения за изменениями переменных можно использовать окно Watch. Окно открывается нажатием на значок  . Переменные, определенные в программе директивой .def, могут быть выделены в тексте программы и перетащены в открывшееся окно. В процессе выполнения программы эта временная переменная будет изменяться, и все её изменения можно наблюдать в открытом окне Watch (рис. 15.17).

Для просмотра программного счетчика, указателя стека, содержимого регистра статуса SREG и индексных регистров X,E и Z в процессе отладки программы можно воспользоваться окном Processor (рис. 15.18). Для открытия окна предназначена кнопка *Processor View*  на панели инструментов AVR Studio.

В этом же окне отображается текущее время выполнения программы, тактовая частота ядра.

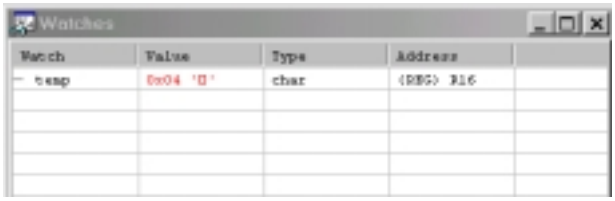


Рис. 15.17. Окно временных переменных

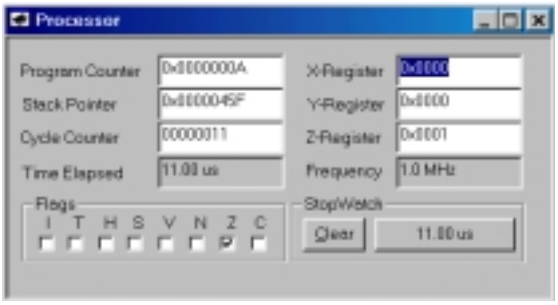


Рис. 15.18. Диалоговое окно для просмотра состояния процессорного ядра

Просмотр ячеек памяти программ, памяти данных, EEPROM и регистров портов ввода/вывода в ходе исполнения программы возможно также с помощью диалогового окна Memory. Окно открывается при нажатии кнопки *Memory Window*  на панели инструментов AVR Studio. Спадающее меню диалогового окна позволяет выбрать один из четырех массивов ячеек памяти: Data, IO, Eeprom, Program Memory. Для одновременного просмотра нескольких областей окно *Memory Window* может быть открыто несколько раз. Информация в диалоговом окне представляется побайтно в шестнадцатеричной системе счисления (рис. 15.19).

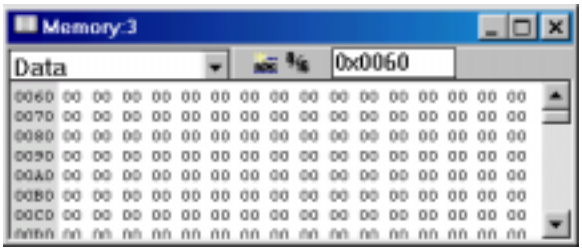


Рис. 15.19. Диалоговое окно Memory Window

Для внесения изменений в программу в процессе отладки необходимо редактировать её исходный текст. При запуске программы после редактирования командой **Go** на экране появляется окно, сообщающее об изменении программы и необходимости её повторного компилирования (рис. 15.20).

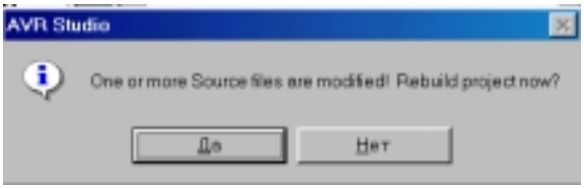


Рис. 15.20. Диалоговое окно изменения проекта

При нажатии кнопки **Да** проект будет перекомпилирован и программный счетчик установится на начало программы. Контрольные точки в программе сохраняются.

Для сохранения проекта необходимо воспользоваться пунктом **Close** меню **Project**. При закрытии проекта сохраняются все его настройки. Во время следующей загрузки настройки будут автоматически восстановлены.

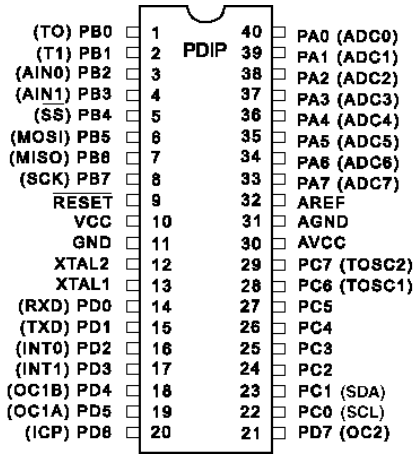
ЛИТЕРАТУРА

1. Бродин В.Б., Шагурин М.И. Микроконтроллеры. Архитектура, программирование, интерфейс.- М.: Издательство ЭКОМ, 1999.- 400 с.
2. Шагурин И.И. Микропроцессоры и микроконтроллеры фирмы Motorola: Справ. пособие.- М.: Радио и связь, 1998.- 560 с.
3. Однокристальные микроконтроллеры PIC12C5x, PIC12C6x, PIC16x8x, PIC14000, M16C/61/62. Под ред. Б.Я.Прокопенко.- М.: ДОДЕКА, 2000.- 336 с.
4. Микроконтроллеры семейства Z86 фирмы ZILIG. Руководство программиста. М.: 1999, - 96 с.
5. Ремизевич Т.В. Микроконтроллеры для встраиваемых приложений: от общих подходов – к семействам HC05 и HC08 фирмы Motorola./ под ред Кирюхина И.С.- М.: ДОДЕКА, 2000.- 272 с.
6. Современные микроконтроллеры: Архитектура, средства проектирования, примеры применения, ресурсы сети Интернет. Под ред. Коршуна И.В.- М.: Издательство «Аким», 1998 – 272 с.
7.

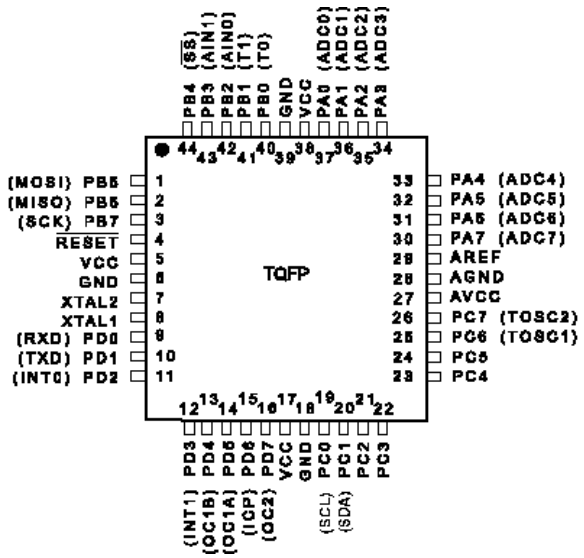
ПРИЛОЖЕНИЯ

Приложение 1. Корпуса микроконтроллера ATmega163

Корпус DIP (Dual In Line pin Package)



Корпус TQFP (Thin Plastic Quad Flat Package)



Приложение 2. Вектора прерываний контроллера ATmega 163

Вектор	Адрес	Обозначение источника	Описание источника	
1	$000	RESET	Reset	Сброс
2	$002	INT0	Interrupt 0	Внешнее прерывание 0
3	$004	INT1	Interrupt 1	Внешнее прерывание 1
4	$006	OC2	Output Compare2	Таймер/счетчик 2, сравнение
5	$008	OV2	Overflow 2	Таймер/счетчик 2, переполнение
6	$00A	ICP1	Input Capture1	Таймер/счетчик 1, захват
7	$00C	OC1A	Output Compare1A	Таймер/счетчик 1, сравнение A
8	$00E	OC1B	Output Compare 1B	Таймер/счетчик 1, сравнение B
9	$010	OV1	Overflow 1	Таймер/счетчик 1, переполнение
10	$012	OV2	Overflow 0	Таймер/счетчик 0, переполнение
11	$014	SPI	SPI	SPI, передача завершена
12	$016	URXC	UART Receive Complete	UART, прием завершен
13	$018	UDRE	UART Data Register Empty	UART, регистр данных пуст
14	$01A	UTXC	UART Transmit Complete	UART, передача завершена
15	$01C	ADCC	ADC	Аналого-цифровой преобразователь
16	$01E	ERDY	EEPROM	Память EEPROM
17	$020	ACI	Analog Comparator	Аналоговый компаратор
18	$022	TWI	TWI	Двухпроводной интерфейс Interface)

ACSR	*Analog Comparator Control and Status Register*	Регистр управления и статуса аналогового компаратора
ADCH	*ADC Data Register High*	Старший регистр данных аналого-цифрового преобразователя
ADCSR	*ADC Control and Status Register*	Регистр управления и контроля аналого-цифрового преобразователя
ADD	*Addition*	сложение
ADMUX	*ADC Multiplexer Select Register*	Регистр выбора мультиплексора аналого-цифрового преобразователя
ALU	*Arithmetic Logic Unit*	Арифметико-логическое устройство
AND	*AND*	Операция И
APS	*Application Program Section*	Раздел прикладной программы
ASSR	*Asynchronous Mode Status Register*	Регистр асинхронного режима таймера 2
BOR	`Brown-out Reset`	Сброс при кратковременном провале напряжения питания
BPS	*Boot Program Section*	Раздел программы начальной загрузки
CISC	*Complicated Instruction Set Computer*	Архитектура с развитой системой команд
CLR	*Clear*	Очистка
DDRA	*Data Direction Register, Port A*	Регистр направления порта A
DDRB	*Data Direction Register, Port B*	Регистр направления порта B
DDRC	*Data Direction Register, Port C*	Регистр направления порта C
DDRD	*Data Direction Register, Port D*	Регистр направления порта D
DEC	*Decrement*	Вычитание 1
EEARH	*EEPROM Address Register High-byte*	Регистр адреса EEPROM, старший байт
EEARL	*EEPROM Address Register Low-byte*	Регистр адреса EEPROM, младший байт
EECR	*EEPROM Control Register*	Регистр управления EEPROM
EEDR	*EEPROM Data Register*	Регистр данных EEPROM
EEPROM	*Electrically EPROM*	Память с электрическим стиранием
EPROM	*Erasable Programmable ROM*	Репрограммируемая память
FB	*Fuse bit*	Бит-предохранитель
Flash	*Flash memory*	Флэш-память
GIFR	*General Interrupt Flag Register*	Регистр флагов прерываний
GIMSK	*General Interrupt Mask Register*	Регистр маски прерываний
GPR	*General Purpose Registers*	Регистры общего назначения
I^2C	*2-Wire Serial Interface*	2-проводной последовательный интерфейс
IC	*Input Capture*	вход захвата
ICR1H T/C 1	*Input Capture Register High-byte*	Регистр входа захвата, старший байт
ICR1L T/C 1	*Input Capture Register Low-byte*	Регистр входа захвата, младший байт
IN	*Input*	Ввод
INC	*Increment*	Прибавить 1
IOR	*Input Output Registers*	Регистры ввода/вывода
IR	*Instruction register*	Регистр инструкций процессорного ядра
JMP	*Jump*	Переход

LB	*Lock bit*	Бит блокировки
MCU	*Microprocessor Core Unit*	Процессорное ядро
MCUCR	*MCU general Control Register*	Общий регистр управления процессорного ядра
MCUSR	*MCU general Status Register*	Общий регистр статуса процессорного ядра
MISO	*Master In Slave Out*	Вход ведущего – выход ведомого интерфейса SPI
MOSI	*Master Out Slave In*	Выход ведущего - вход ведомого интерфейса SPI
MUL	*Multiply*	Умножение
OC	*Output Compare*	Выходное сравнение
OCR1A	*Timer/Counter1 Output Compare Register A High-byte*	Регистр А выхода сравнения таймера/счетчика 1, старший байт
OCR1B	*Timer/Counter1 Output Compare Register B High-byte*	Регистр В выхода сравнения таймера/счетчика 1, старший байт
OCR2	*Timer/Counter2 Output Compare Register*	Регистр выхода сравнения таймера/счетчика 2
OR	*OR*	Операция ИЛИ
OSCCAL	*Oscillator Calibration Register*	Регистр калибровки резонатора
OTPROM	*One Time Programmable ROM*	Однократно программируемая память
PC	*Program counter*	Программный счетчик
PINA	*Input Pins, Port A*	Регистр входных контактов порта А
PINB	*Input Pins, Port B*	Регистр входных контактов порта В
PINC	*Input Pins, Port C*	Регистр входных контактов порта С
PIND	*Input Pins, Port D*	Регистр входных контактов порта D
POR	*Power-on Reset*	Сброс при включении питания
PORTA	*Data Register, Port A*	Регистр данных порта А
PORTB	*Data Register, Port B*	Регистр данных порта В
PORTC	*Data Register, Port C*	Регистр данных порта С
PORTD	*Data Register, Port D*	Регистр данных порта D
PUR	*Pull-up resistors*	Подтягивающие резисторы
PWM	*Pulse-Width Modulation*	Широтно-импульсная модуляция
RET	*Return*	Возврат
RISC	*Reduced Instruction Set Computer*	Архитектура с сокращенным набором команд
ROM	*Read Only Memory*	Постоянная память
RTC	*Real Time Clock*	Часы реального времени
SCA	*Serial Clock*	Синхронизация интерфейса I2C
SDA	*Serial Data*	Данные интерфейса I2C
SFIOR	*Special Function I/O Register*	Регистр специальных функций ввода/вывода
SP	*Stack Pointer*	Указатель стека
SPCR	*SPI Control Register*	Регистр управления интерфейса SPI
SPDR	*SPI I/O Data Register*	Регистр данных интерфейса SPI
SPH	*Stack Pointer High*	Указатель стека, старший байт
SPI	*Serial Peripheral Interface*	Последовательный периферийный интерфейс
SPL	*Stack Pointer Low*	Указатель стека, младший байт

SPSR	*SPI Status Register*	Регистр статуса интерфейса SPI
SRAM	*Static Random Access Memory*	статическая оперативная память
SREG	*Status register*	регистр состояния
SREG	*Status Register*	Регистр состояния
SUB	*Subtract*	Вычитание
TCCR0	*Timer/Counter0 Control Register*	Регистр управления таймера/счетчика 0
TCCR1A	*Timer/Counter1 Control Register A*	Регистр управления A таймера/счетчика 1
TCCR1B	*Timer/Counter1 Control Register B*	Регистр управления B таймера/счетчика 1
TCCR2	*Timer/Counter2 Control Register*	Регистр управления таймера/счетчика 2
TCNT0	*Timer/Counter0*	Таймер/счетчик 0
TCNT1H	*Timer/Counter1 High-byte*	Старший байт таймера/счетчика 1
TCNT1L	*Timer/Counter1 Low-byte*	Младший байт таймера/счетчика 1
TCNT2	*Timer/Counter2*	Таймер/счетчик 2
TIFR	*Timer/Counter Interrupt Flag Register*	Регистр флагов прерываний таймеров/счетчиков
TIMSK	*Timer/Counter Interrupt Mask Register*	Регистр масок прерываний таймеров/счетчиков
TWCR	*2-wire Serial Interface Control Register*	Регистр управления двухпроводного последовательного интерфейса
UART	*Universal Asynchronous Receiver Transmitter*	универсальный асинхронный приемопередатчик
UBRR	*UART Baud Rate Register*	Генератор скорости передачи
UBRRHI	*UART Baud Rate Register High-byte*	Старший байт генератора скорости передачи
UCSRA	*UART Control and Status Register A*	Регистр A управления и контроля UART
UCSRB	*UART Control and Status Register B*	Регистр B управления и контроля UART
UDR	*UART I/O Data Register*	Регистр данных UART
WDT	*Watchdog Timer*	Сторожевой таймер
WDTCR	*Watchdog Timer Control Register*	Регистр управления сторожевым таймером

ОГЛАВЛЕНИЕ