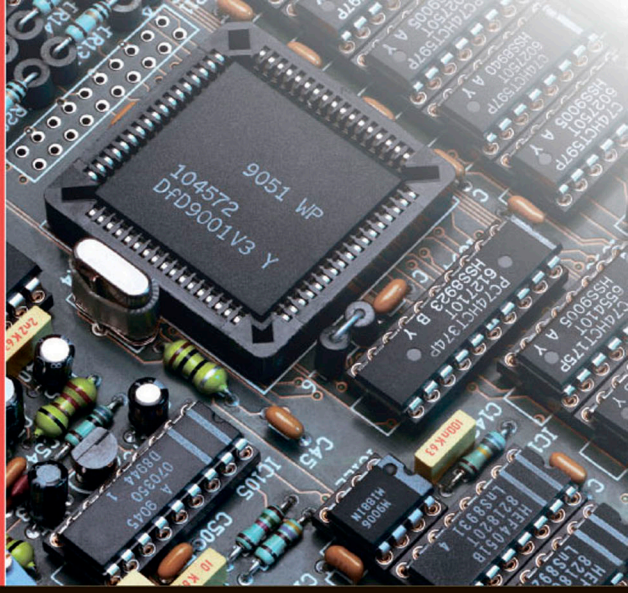
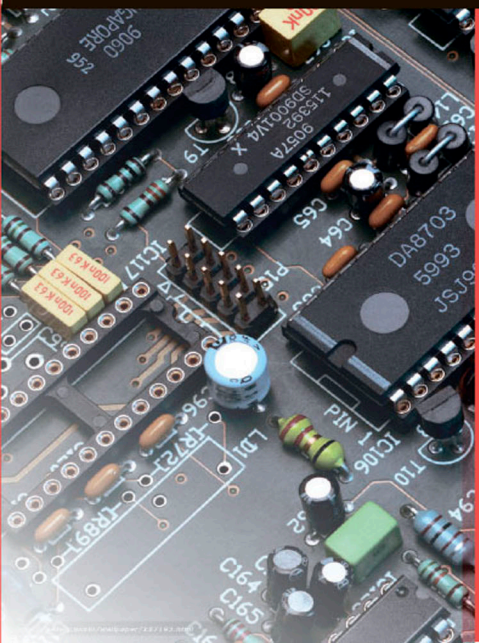
А.Водовозов



МИКРОКОНТРОЛЛЕРЫ ДЛЯ СИСТЕМ АВТОМАТИКИ



Учебное пособие

«Инфра-Инженерия»

А.М. Водовозов

МИКРОКОНТРОЛЛЕРЫ ДЛЯ СИСТЕМ АВТОМАТИКИ

Учебное пособие

Инфра-Инженерия
Москва-Вологда
2016

УДК 621.375(03)
ББК 32.85
В 62

Рецензенты:
кандидат технических наук, доцент, зам. главного энергетика
ООО МП «Межрайонные электротеплосети» *С.Б. Федотовский*;

кандидат технических наук, зав. кафедрой прикладной информатики
и естественно-математических наук Вологодского
института бизнеса *Н.А. Хромцова*

В 62 **Водовозов А.М.**
Микроконтроллеры для систем автоматики: учебное пособие / А.М. Водовозов. Изд. 3-е, доп. и перераб. – М.: Инфра-Инженерия, 2016.– 164 с. : ил. ; табл.

ISBN 978–5–9729–0138–8

Рассмотрены архитектурные принципы построения микроконтроллеров, процессорного ядра, подсистемы памяти, современного набора периферийных модулей и технологии их программирования.

Предназначено для студентов технических вузов, может быть использовано также студентами средних профессиональных учебных заведений при изучении современных микроконтроллеров. Представляет интерес для инженерно-технических работников, занимающихся проектированием микропроцессорной техники.

ISBN 978–5–9729–0138–8

ОГЛАВЛЕНИЕ

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 и вычислительной техники.

За 40 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, Texas Instruments и др.

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 микроконтроллеров, 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 микроконтроллеров всегда строится на конкретных примерах какой-либо одной архитектуры. В пособии все основные понятия микропроцессорной техники 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. Архитектура микроконтроллера включает в себя структуру, отражающую состав изделия, и его программно-математическое обеспечение.

1.1. Обобщенная структура микроконтроллера

Типовая структура микроконтроллера изображена на рисунке 1.1. Микроконтроллер состоит из трех, связанных системными шинами, элементов: процессорного ядра, памяти и набора устройств ввода/вывода различного назначения.

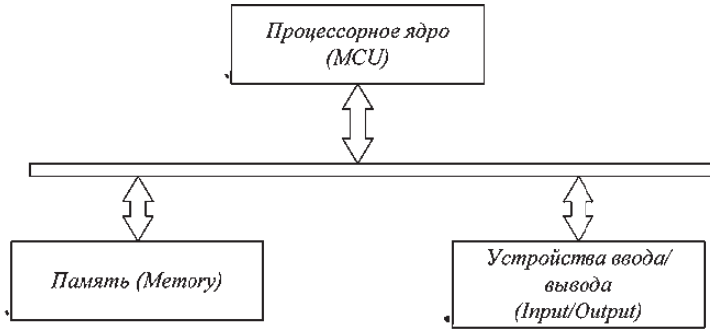


Рис. 1.1. Структура микроконтроллера

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 различными устройствами ввода/вывода, интегрированными на кристалл. 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*). В памяти данных хранятся результаты промежуточных вычислений, в памяти программ – программа, по которой работает микроконтроллер. Память программ микроконтроллера энергонезависима, при отключении питания в ней сохраняется записанная программа.

Устройства ввода/вывода 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 любое устройство ввода/вывода 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 ходе 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 микроконтроллера. Понятие «архитектура» включает в себя множество его 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.
- 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.

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.

1.2. Микроконтроллеры с AVR-архитектурой

Стандартные семейства

Концепция AVR-микроконтроллеров была разработана группой разработчиков исследовательского центра ATMEL в Норвегии в середине 90-х годов. Первые микроконтроллеры AVR появились в 1997 г.

AVR-архитектура объединила гарвардский RISC-процессор с раздельным доступом к памяти программ и данных и развитую систему команд фиксированной 16-битной длины. Большинство команд выполняются за один машинный такт с одновременным исполнением текущей и выборкой следующей команды, что обеспечивает производительность до 1 миллиона инструкций в секунду на каждый МГц тактовой частоты. Базовый набор команд AVR содержит 130 инструкций.

Все микроконтроллеры AVR имеют встроенную FLASH-память с возможностью внутрисхемного программирования через последовательный 4-проводной интерфейс.

Периферия микроконтроллеров AVR включает: таймеры-счётчики, широтно-импульсные модуляторы, систему прерываний, аналоговые компараторы, 8-10-разрядные многоканальные АЦП, параллельные порты, интерфейсы UART и SPI, сторожевой таймер и устройство сброса по включению питания.

В рамках единой базовой архитектуры AVR-микроконтроллеры подразделяются на три подсемейства (таблица 1.1), отличающиеся разрядностью, тактовой частотой, производительностью, объемом памяти и областями применения.

Таблица 1.1

Характеристики микроконтроллеров с AVR-архитектурой

Семейство	Разрядность	Флэш-память (кВ)	Корпуса (выводы)	Максимальная частота (МГц)	Производительность (MIPS/МГц)
tinyAVR	8	0,5–8	6–32	20	1
megaAVR	8	4–256	28–100	20	1
AVR XMEGA	8	16–384	44–100	32	1
AVR UC3	32	16–512	48–144	66	1,5

AVR-микроконтроллеры поддерживают спящий режим и режим микропотребления. В спящем режиме останавливается центральное процессорное ядро, в то время как регистры, таймеры-счётчики, сторожевой таймер и система прерываний продолжают функционировать. В режиме микропотребления сохраняется содержимое всех регистров, останавливается тактовый генератор, запрещаются все функции микроконтроллера, пока не поступит сигнал внешнего прерывания или аппаратного сброса.

Семейство tiny AVR

Микроконтроллеры tiny AVR оптимизированы для систем, в которых требуется высокая производительность, эффективное использование энергии, простота применения и компактность. В микроконтроллеры встроены АЦП, память EEPROM и детектор понижения напряжения питания, для проектирования систем не требуются дополнительные внешние компоненты. Они сочетают в себе миниатюрность, вычислительную мощность, высокую производительность аналоговой части и интеграцию на системном уровне. Это самые компактные полнофункциональные устройства в семействе AVR; и только они могут работать с напряжением питания всего 0,7 В, что является их главным преимуществом.

Семейство mega AVR

Микроконтроллеры семейства mega AVR оснащены памятью программ и данных большого объема при быстродействии до 20 млн операций в секунду, а потому идеально подходят для приложений со значительным объемом программы. Все устройства mega AVR поддерживают самопрограммирование, что позволяет выполнять внутрисхемное обновление быстро, безопасно и без лишних затрат. В семейство входит самый широкий перечень устройств с различными объемами памяти, количеством выводов и набором периферии. Это модели общего назначения и модели со специализированными периферийными устройствами и интерфейсами, например интерфейсом USB либо контроллерами ЖК-дисплея, сетей CAN и LIN или усилителей мощности.

Семейство AVR XMEGA

Микроконтроллеры AVR XMEGA предназначены для работы в реальном времени. В них минимизируется количество необходимых комплектующих для широкого круга приложений. В одном устройстве объединяются высокоскоростные аналоговые модули, гибко настраиваемые таймеры и счетчики, множество коммуникационных модулей и интерфейсов, а также меха-

низм эффективного управления электропотреблением. Контроллер прямого доступа к памяти совместно с системой обработки событий гарантируют прогнозируемость производительности в реальном времени даже при большой нагрузке.

Семейство AVR UC3

Микроконтроллеры AVR UC3 предназначены для использования в задачах с высокими требованиями к производительности. Благодаря интеллектуальной периферии и динамическому управлению питанием они целесообразны для применения в портативных системах и приборах с питанием от батареи. Некоторые микроконтроллеры семейства оснащены встроенным блоком вычислений с плавающей точкой, повышающим производительность арифметических операций с десятичными числами, увеличивающим точность вычислений и расширяющим их динамический диапазон.

Контрольные вопросы

- Какая интегральная схема называется микроконтроллером?
- Какая информация хранится в памяти данных микроконтроллера?
- Какая информация хранится в памяти программ микроконтроллера?
- Какая память микроконтроллера является энергонезависимой?
- Чем определяется разрядность микроконтроллера?
- Могут ли ячейки памяти иметь разрядность, отличную от разрядности микроконтроллера?
- Какая архитектура системы команд предусматривает исполнение команд преимущественно за 1 такт работы процессорного ядра?
- В какой архитектуре микроконтроллера память данных и память программ могут иметь разную разрядность?
- В какой архитектуре системы команд инструкции могут быть разной длины?
- В какой архитектуре системы команд большинство инструкций размещается в одной ячейке памяти?
- Какие особенности имеет AVR-архитектура?
- Отличительные особенности семейства микроконтроллеров tiny AVR?
- Отличительные особенности семейства микроконтроллеров mega AVR?

- Отличительные особенности семейства микроконтроллеров AVRXMEGA?
- Отличительные особенности семейства микроконтроллеров UC3 AVR?

2. ПАМЯТЬ

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

2.1. Память программ

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

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: 0x0, 0x1, 0x2 …0xa, 0xb…..Здесь символами 0x обозначаются числа в шестнадцатеричной системе счисления. Допускается также шестнадцатеричные числа обозначать символом $, например, $5, $a1.

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. В зависимости от способа программирования память типа ROM разделяется на несколько групп: OTPROM, EPROM, EEPROM и flash-memory.

Однократно программируемая память

Однократно программируемая память (*OTPROM – One Time Programmable ROM*) 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 - ErasablePROM)* 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 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…20 В, 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 раздел начальной загрузки, где располагается специальная программа.

Например, микроконтроллер ATmega16 имеет 8K 16-битных ячеек Flash-памяти программ. Общий объем памяти 16 Кбайт. При этом всё адресное пространство флэш-памяти программ (0x0000…0x1FFF) 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*).

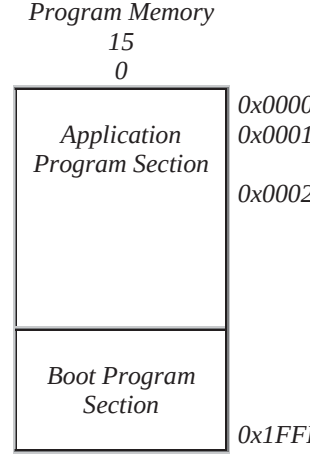


Рис. 2.1. Организация памяти программ микроконтроллера ATmega16

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 раздел Application Program Section. Эта команда может быть использована только в Boot Program Section .

В Boot Program Section записывается программа, позволяющая процессорному ядру микроконтроллера организовать перепрограммирование раздела прикладной программы непосредственно в изделии.

Весь массив 8К ячеек flash-memory микроконтроллера ATmega16 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 *(Data Memory)* 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.

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

Например, микроконтроллер ATmega16 имеет 1024 восьмибитные ячейки встроенной статической памяти типа SRAM. Объем статической памяти микроконтроллера равен 1кбайт. Сама память занимает адресное пространство 0x60…..0x45F (рисунок 2.2). Размещенные в том же адресном пространстве микроконтроллера 32 ячейки с адресами 0x0…0x1F используются микроконтроллером как регистры общего назначения (*General Purpose Registers*), а 64 ячейки с адресами 0x2F…0x5F – как регистры ввода/вывода (*Input/Output Registers*).

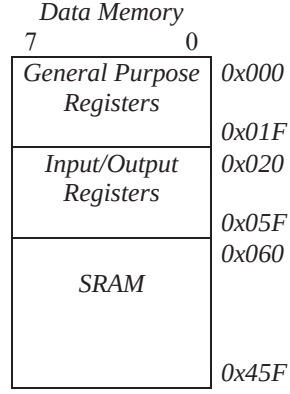


Рис. 2.2. Статическая память данных микроконтроллера ATmega16

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

Память с электрическим стиранием

Память данных типа EEPROM может быть включена в общее пространство памяти данных или организована в виде отдельного массива с возможностью считывания и записи каждого отдельного байта. В отличие от SRAM она допускает только ограниченное количество (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 инструкции. Защита EEPROM от разрушения при снижении напряжения питания в контроллерах решается как программными, так и аппаратными средствами.

Например, EEPROM микроконтроллера ATmega16 имеет объем 512 байт. Она обеспечивает 100000 циклов стирания/записи. Обращение к EEPROM ведется через специальные регистры ввода/вывода.

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.

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 ATmega16. В его структуре предусмотрены:

- Биты блокировки (*lock-bits*) LB1 и LB2 (таблица 2.2), 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.

Микроконтроллер ATmega16, поддерживающий режим последовательного программирования имеет fuse-биты:

- SPIEN *(SPI Enable),* запрещающий режим последовательного программирования кристалла,
- CKSEL[0..3] *(Clock Select),* задающий источник тактового сигнала микроконтроллера,
- SUT1:0 *(Start-up Times),* задающий время задержки старта микроконтроллера после сброса,
- BODEN и BODLEVEL, управляющие схемой контроля напряжения питания,
- BOOTSZ, задающие объем секции начальной загрузки в памяти программ,
- CKOPT *(Oscillator Option),* задающий вид внешнего кварцевого осциллятора.

Байты сигнатуры (*signature-byte*) служат для идентификации типа кристалла, программируются изготовителем и доступны только для чтения.

Контрольные вопросы

- Какая память программ используется в микроконтроллерах с архитектурой AVR?
- Какой объем памяти программ имеет микроконтроллер ATmega16?
- Для чего в памяти программ предусматривают раздел начальной загрузки?
- Как загружается программа пользователя в память программ?
- Память какого типа используется для хранения данных в микроконтроллерах с AVR-архитектурой?
- Для чего в структуре микроконтроллера предусматривается память типа EEPROM?
- Почему память типа EEPROM не может быть использована непосредственно при выполнении инструкций процессорного ядра?
- Какой объем статической памяти имеет микроконтроллер типа ATmega16?
- Какой адрес имеет старшая ячейка статической памяти микроконтроллера ATmega16?
- Какой адрес имеет младшая ячейка статической памяти микроконтроллера ATmega16?

- Какая информация хранится в ячейках статической памяти с адресами 0x0÷0x1f?
- Какая информация хранится в ячейках статической памяти с адресами 0x20÷0x5f?
- Для чего в структуре микроконтроллера предназначаются биты блокировки?
- Для чего в структуре микроконтроллера предназначаются биты-предохранители?
- Для чего в структуре микроконтроллера предназначаются байты сигнатуры?

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:

- программный счетчик,
- 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 инструкции содержимое программного счетчика изменяется. В простейшем случае оно увеличивается на единицу. Но некоторые инструкции 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 ATmega16 с ядром AVR имеет только 8К

ячеек памяти программ, поэтому в нем используется только 13 младших бит программного счетчика ядра AVR (2^{13} = 8K).

Регистры общего назначения

Регистры общего назначения (*GPR – General purpose registers*) 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.

Например, ядро AVR имеет 32 регистра общего назначения. 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.

General purpose registers	*SRAM*	
R0	*0x00*	
R1	*0x01*	
R26	*0x1A*	*X-register low byte*
R27	*0x1B*	*X-register high byte*
R28	*0x1C*	*Y - register low byte*
R27	*0x1D*	*Y - register high byte*
R30	*0x1E*	*Z - register low byte*
R31	*0x1F*	*Z - register high byte*

Рис. 3.1. Файл регистров общего назначения ядра AVR

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 реги-

страх хранятся младшие байты 16-битных переменных, а в нечетных – старшие (рис. 3.2).

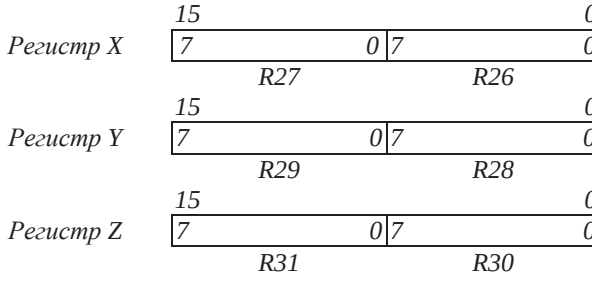


Рис. 3.2. Регистры косвенной адресации архитектуры AVR

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 ATmega16 предусмотрено 64 регистра ввода/вывода (*Input/Output registers*). Все они размещены в пространстве ввода/вывода процессорного ядра по адресам от 0x00 по 0x3F (приложение 1). Для обращения к регистрам можно использовать специальные инструкции (IN) и вывода (OUT). При этом в инструкции необходимо указывать адрес регистра. 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.

Зарезервированные (не указанные в списке) адреса при программировании микроконтроллера 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. Ре-

гистр обычно рассматривается как набор бит (флагов), каждый из которых несет в себе определенную информацию о каком-либо одном признаке результата.

Например, 8-битный регистр состояния ядра AVR (*SREG – Status register*) размещен в пространстве регистров ввода/вывода (приложение 1) по адресу 0x3F и его биты определяются в соответствии с рисунком 3.3:

- бит 7 – I *(Global Interrupt Enable)* - бит разрешения глобального прерывания;
- бит 6 – T *(Bit Copy Storage)* - бит сохранения копии;
- бит 5 – H (*Half Carry Flag) –флаг* полупереноса;
- бит 4 – S (*Sign Bit)* –бит знака. Бит S всегда находится в состоянии, определяемом операцией, исключающей 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.

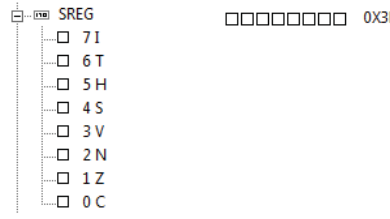


Рис. 3.3. Регистр состояния ядра AVR

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 в области SRAM. В этом 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. Микроконтроллер ATmega16 оснащен 16-разрядным указателем стека, размещенным в двух регистрах SPH (*Stack Pointer High*) и SPL (*Stack Pointer Low)* пространства ввода/вывода по адресам 0x3E и 0x3D (рис. 3.4). Поскольку рассматриваемый микроконтроллер имеет SRAM с адресами от 0x000 до 0x045F (рис. 2.2), то в нем используется только 11 бит указателя стека SP0….SP10.

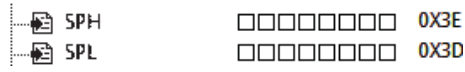


Рис. 3.4. Регистры указателя стека микроконтроллера ATmega16

Указатель стека (SP) указывает на область в статической памяти данных, в которой размещается 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

Система команд микроконтроллера – набор инструкций, которые может выполнять процессорное ядро. Каждая инструкция из системы команд представляется простым трех- или четырехбуквенным мнемоническим символом. Мнемонические символы ассоциируются с функциями инструкции и однозначно соответствуют действиям процессорного ядра при выполнении этой инструкции. Каждая архитектура имеет фиксированный набор символических команд, созданных и описанных его производителем.

Программа, записанная на языке символического кодирования, процессорным ядром не воспринимается. При вводе её в память программ осущест-

вляется 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 имеются , 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 и адрес регистра или ячейки памяти, задействованных при выполнении этой операции. Вместе с тем, ряд инструкций имеют размерность 32 бита. В этом случае 16-битный адрес одного из операндов хранится в 16 младших разрядах кода.

Непосредственная адресация

При непосредственной адресации одним из операндов инструкции является числовая константа (K). Она хранится непосредственно в коде инструкции. Вторым операндом инструкции должен быть 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 K записывается в указанный регистр Rd. В сокращенном виде эти действия можно представить в виде: Rd<- K.

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 <- 0x00
ser Rd	*set register*	Rd <- 0xFF

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.

Битовые операции с регистром общего назначения

Операндом инструкции 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.

Операции с регистром ввода/вывода

Операнды инструкции 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

Битовые операции с регистром ввода/вывода

Операндом инструкции является бит в ввода/вывода. Побитно адресуемыми регистрами в пространстве ввода вывода являются регистры с номерами 0x00….0x1F.

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

Операции с памятью данных

Операнды инструкции содержатся в одном из регистров общего назначения Rd или Rr и в одной из ячеек памяти данных (с номером 0xK).

Например,

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

Операции с памятью программ

В инструкции указывается адрес ячейки памяти программ. Обычно таким образом адресуются различные инструкции 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

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)

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.

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

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.

Например:

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

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.

Например,

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

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, инструкции, 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).

Инструкции переходов

ijmp	*Indirect jump to (z)*	PC <- Z

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.

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, родственные по применению инструкций, например, арифметические и логические инструкции, инструкции 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 ATmega16.

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

- инструкции пересылки данных (*data transfer instructions*),
- арифметические и логические инструкции (*arithmetic and logic instructions*),
- инструкции работы с битами *(bit and bit-test instructions*),
- инструкции 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 ≤d≤ 31),
- Rr - регистр-источник (0 ≤r≤ 31),
- Rdl: регистры R24, R26, R28, R30 (для инструкций ADIW и SBIW),
- P- адрес регистра ввода/вывода,
- P*- адрес побитно адресуемого регистра ввода/вывода (0x00-0x1F)
- 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

Инструкции пересылки

Инструкции 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 инструкции 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 инструкции 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

Инструкции пересылки AVR-микроконтроллеров

Мнемоника	Операнды	Описание инструкции	Операция	Такты
mov	Rd, Rr	*move between registers*	Rd <- Rr	1
movw	Rd, Rr	*copy register word*	Rd+1:Rd <- Rr+1:Rr	1
ldi	Rd*, K	*load immediate*	Rd <- K	1
ld	Rd, X	*load indirect*	Rd <- (X)	2
ld	Rd, X+	*load indirect and post-inc.*	Rd <- (X), X <- X + 1	2
ld	Rd, - X	*load indirect and pre-dec.*	X <- X - 1, Rd <- (X)	2
ld	Rd, Y	*load indirect*	Rd <- (Y)	2
ld	Rd, Y+	*load indirect and post-inc.*	Rd <- (Y), Y <- Y + 1	2
ld	Rd, - Y	*load indirect and pre-dec.*	Y <- Y - 1, Rd <- (Y)	2
ldd	Rd, Y+q	*load indirect with displacement*	Rd <- (Y + q)	2
ld	Rd, Z	*load indirect*	Rd <- (Z)	2
ld	Rd, Z+	*load indirect and post-inc.*	Rd <- (Z), Z <- Z+1	2
ld	Rd, -Z	*load indirect and pre-dec*	Z <- Z - 1, Rd <- (Z)	2
ldd	Rd, Z+q	*loadin direct with displacement*	Rd <- (Z + q)	2
lds	Rd, k	*load direct from SRAM*	Rd <- (k)	2
st	X, Rr	*store indirect*	(X) <- Rr	2
st	X+, Rr	*store indirect and post-inc.*	(X) <- Rr, X <- X + 1	2
st	- X, Rr	*store indirect and pre-dec.*	X <- X - 1, (X) <- Rr	2
st	Y, Rr	*store indirect*	(Y) <- Rr	2
st	Y+, Rr	*store indirect and post-inc.*	(Y) <- Rr, Y <- Y + 1	2
st	- Y, Rr	*store indirect and pre-dec.*	Y <- Y - 1, (Y) <- Rr	2
std	Y+q,Rr	*store indirect with displacement*	(Y + q) <- Rr	2
st	Z, Rr	*store indirect*	(Z) <- Rr	2
st	Z+, Rr	*store indirect and post-inc.*	(Z) <- Rr, Z <- Z + 1	2
st	-Z, Rr	*store indirect and pre-dec*	Z <- Z - 1, (Z) <- Rr	2
std	Z+q,Rr	*store indirect with displacement*	(Z + q) <- Rr	2
sts	k, Rr	*store direct to SRAM*	(k) <- Rr	2
lpm		*load program memory*	R0 <- (Z)	3
lpm	Rd, Z	*load program memory*	Rd <- Z)	3
lpm	Rd, Z+	*load program memory and post-inc.*	Rd <- Z), Z=Z+1	3
spm		*store program memory*	(Z) <- R1:R0	3
in	Rd, P	*in port*	Rd <- P	1
out	P, Rr	*out port*	P <- Rr	1
push	Rr	*push register on stack*	STACK <- Rr; SP<- SP-1	2
pop	Rd	*pop register from stack*	SP<- SP+1, Rd <- STACK	2

Арифметические и логические инструкции

В группу арифметических команд входят инструкции, 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 ATmega16 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 инструкции сложения и вычитания выполняют сложение и вычитание одно- и двухбайтных операндов. Инструкции 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 инструкции 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.

Инструкции 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 инструкции с непосредственной адресацией, работающие с двухбайтными словами: adiw (*add immediate to word*) и sbiw (*subtract immediate from word*), и инструкции умножения, выполняющиеся за два такта.

Таблица 3.2

Арифметические и логические инструкции

Мнемоника	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 <- 0xFF – Rd	Z,C,N,V	1
neg	Rd	*two's complement*	Rd <- 0x00 – 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 <- 0xFF	None	1
cbr	Rd*, K	*clear bit(s) in register*	Rd <- Rd (0xFF – K)	Z,N,V	1

Мнемо-ника	Операнды	Описание	Операция	Флаги	Такт
clr	Rd	*clear register*	Rd <- 0x00	Z,N,V	1
inc	Rd	*increment*	Rd <- Rd + 1	Z,N,V	1
dec	Rd	*decrement*	Rd <- Rd –1	Z,N,V	1
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 инструкции

Битовые инструкции*(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 инструкции могут влиять на отдельные флаги регистра признаков.

Битовые инструкции 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 инструкции предусмотрены для установки и сброса отдельных флагов регистра состояния. Команда swap меняет местами тетрады (полубайты) регистров общего назначения.

Таблица 3.3

Битовые инструкции микроконтроллера ATmega16

Мне-моника	Операн-ды	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
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 brim 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

Инструкции переходов микроконтроллера *ATmega16*

Мнемоника	Операнды	Описание	Операция	Такт
rjmp	k	*relative jump*	PC <- PC + k + 1	2
ijmp		*indirect jump to (z)*	PC <- Z	2
jmp	k	*jump*	PC <- k	3
rcall	k	*relative subroutine call*	PC <- PC + k + 1	3
call	k	*call subroutine*	PC <- k	4
icall		*indirect call to (z)*	PC <- Z	3
ret		*subroutine return*	PC <- STACK	4
reti		*interrupt return*	PC <- STACK	4
cpse	Rd,Rr	*compare, skip if equal*	if (Rd = Rr) PC <- PC + 2 or 3	1/ 2 / 3
sbrc	Rr, b	*skip if bit in register cleared*	if (Rr(b)=0) PC <- PC + 2 or 3	1 / 2
sbrs	Rr, b	*skip if bit in register is set*	if (Rr(b)=1) PC <- PC + 2 or 3	1 / 2
sbic	P*, b	*skip if bit in i/o register cleared*	if (P(b)=0) PC <- PC + 2 or 3	1 / 2
sbis	P*, b	*skip if bit in i/o register is set*	if (P(b)=1) PC <- PC + 2 or 3	1 / 2
brbs	s, k	*branch if status flag set*	if (SREG(s) = 1) then PC <- PC+k + 1	1 / 2
brbc	s, k	*branch if status flag cleared*	if(SREG(s) = 0) then PC <- PC+k + 1	1 / 2
breq	k	*branch if equal*	if (Z = 1) then PC <- PC + k + 1	1 / 2

Мнемоника	Операнды	Описание	Операция	Такт
brcs	k	*branch if carry set*	if (C = 1) then PC <- PC + k + 1	1 / 2
brne	k	*branch if not equal*	if (Z = 0) then PC <- PC + k + 1	1 / 2
brcc	k	*branch if carry cleared*	if (C = 0) then PC <- PC + k + 1	1 / 2
brsh	k	*branch if same or higher*	if (C = 0) then PC <- PC + k + 1	1 / 2
brlo	k	*branch if lower*	if (C = 1) then PC <- PC + k + 1	1 / 2
brmi	k	*branch if minus*	if (N = 1) then PC <- PC + k + 1	1 / 2
brpl	k	*branch if plus*	if (N = 0) then PC <- PC + k + 1	1 / 2
brge	k	*branch if greater or equal, signed*	if (N ⊕ V= 0) then PC <- PC + k + 1	1 / 2
brlt	k	*branch if less than zero, signed*	if (N ⊕ V= 1) then PC <- PC + k + 1	1 / 2
brhs	k	*branch if half carry flag set*	if (H = 1) then PC <- PC + k + 1	1 / 2
brhc	k	*branch if half carry flag cleared*	if (H = 0) then PC <- PC + k + 1	
brts	k	*branch if T flag set*	if (T = 1) then PC <- PC + k + 1	1 / 2
brtc	k	*branch if T flag cleared*	if (T = 0) then PC <- PC + k + 1	1 / 2
brvs	k	*branch if overflow flag is set*	if (V = 1) then PC <- PC + k + 1	1 / 2
brvc	k	*branch if overflow flag is cleared*	if (V = 0) then PC <- PC + k + 1	1 / 2
brie	k	*branch if interrupt enabled*	if (I = 1) then PC <- PC + k + 1	1 / 2
brid	k	*branch if interrupt disabled*	if (I = 0) then PC <- PC + k + 1	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 ATmega16

Мнемоника	Описание	Операция	Такт
nop	*no operation*	none	1
sleep	*sleep*	none	3
wdr	*watchdog reset*	none	1

3.3. 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 код может быть непосредственно запрограммирован в микроконтроллеры AVR.

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.

Инструкции 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 = 0x19      ; метка pina  с помощью директивы .set связывается с числом 0x19
```

- 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 = 0x19                ; метка pina связывается с числом 0x19
.set porta = pina + 2           ; метка porta связывается с числом 0x19+2=0x1B
……
out porta,r2          Пересылка данных из регистра r2 в porta (по адресу 0x1B)
```

- 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 = 0x19              ; метка pina связывается с числом 0x19
```

.equ porta = pina + 2 ; метка porta связывается с числом 0x19+2=0x1B

……

out porta,r2 ; пересылка данных из регистра r2 в porta (по ;адресу 0x1B)

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

Операторы

Все операторы ассемблера перечислены в таблице 3.6. 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 3.6

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*	Сдвигвправо	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,!0xf0  ; в регистр r16 пересылается 0x00
```

Побитное НЕ

Оператор ~ возвращает побитную инверсию выражения. Например:

```
ldi r16,~0xf0 ; в регистр r16 пересылается 0x0f
```

Смена знака

Оператор - возвращает отрицание арифметического выражения. Например:

```
ldi r16,-2 ; пересылается число –2 (0xfe) 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. Например:

```
ldi r30,label/2 ; пересылается в регистр r30 целая часть выражения label/2
```

Сложение

Оператор + возвращает сумму двух выражений. Например:

```
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. Например:

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) тоже, чтоhigh;
- 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.

Директивы

Директивы ассемблера не транслируются в коды операций, они используются для размещения программы в памяти, определяют макроинструкции, инициализируют память данных и выполняют ещё целый ряд различных вспомогательных операций, облегчающих процесс программирования. Полный список директив ассемблера AVR приведен в таблице 3.7.

Таблица 3.7

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 list file generation on*	Включение генерации листинга
listmac	*Turn Macro expansion in list file on*	Включение макроса в листинг
nolist	*Turn list file 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 инструкции ассемблера поддержаны и нет никаких ограничений на EEPROM и память программ. Например,

```
.device atmega16     ; выбираем atmega16
```

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 из 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. Директива cseg не имеет параметров. Например,

```
.cseg    ; начало сегмента кода
const: .dw 2          ; запись 0x0002 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. 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. 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 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. Например,

```
.dseg
var1:   .byte 1              ; резервируется 1 байт для var1
table:  .byte tab_size       ; резервируется tab_size байт
.cseg
ldir30,low(var1)             ; загрузка младшего байта регистра z
ldir31,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.

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

Если директива db расположена в сегменте программы cseg и список выражений содержит более чем одно выражение, то выражения упаковыва-

ются 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. Например,

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

dw

Директива dw (*Define constant word(s) in program memory and EEPROM*) резервирует 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. Каждое выражение должно находиться в пределах от -32768 до 65535. Отрицательные числа записываются в дополнительном коде. В списке выражений последовательность выражений разграничивается запятыми. 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. Например,

```
.deftemp=r16
.defior=r0
.cseg
ldi temp,0xf0          ; запись 0xf0 в регистр temp
in ior,0x3f            ;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, указы-

вающая на константу в соответствии с директивой set, не может быть впоследствии переопределена.

Директива .equ обычно используется для присвоения символических имен всем регистрам микроконтроллера, отдельным битам регистров и ячеек памяти.

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.Например,

.set pina = 0x19 ; имя pina присваивается числу 0x19

.set porta = pina + 2 ; имя porta присваивается выражению 0x19+2=0x1b

……

out porta,r2 ;Пересылка данных из регистра r2 в porta (по адресу 0x1B)

Управление файлами

include

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

.include “m16def.inc “ ;подключаем файл описания микроконтроллера *ATmega16*

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 «+».Например,

```
.macrosubi16            ; начало определения макроса
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. Например,

```
.macrosubi16 ; начало определения макроса
subir16,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. 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. Например,

```
.macromacx              ; определение макроса
add r0,@0
eor r1,@1
```

```
.endmacro               ; конец определения
.listmac                ; включение макроса в листинг программы
macxr2,r1               ; выполнение макроса
```

nolist

Директива nolist (*Turn list file 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.

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, то метке будет присвоено значение параметра. По умолчанию значения счетчиков адреса программы и EEPROM нулевые, а счетчика адреса данных - 32 (после регистров, занимающих адреса 0-31).

Например,

```
.dseg                   ; сегмент данных
.org 0x37               ; адрес 0x37
variable: .byte 1       ; резервирование байта
.cseg                   ; сегмент кода
.org 0x10               ; установка программного счетчика на адрес 0x10
movr0,r1        ;
```

Выход

exit

Директива exit (*Exit this file*) прекращает трансляцию файла. Обычно ассемблер выполняет трансляцию до конца файла. Если директива exit появля-

ется в подключенном файле, ассемблирование продолжается от строки, следующей за директивой include.Например,

```
.exit     ; конец трансляции
```

Контрольные вопросы

- Перечислить основные элементы процессорного ядра.
- Какая информация хранится в программном счетчике?
- Как изменяется содержимое программного счетчика после выполнения очередной инструкции?
- Какие инструкции позволяют принудительно изменить содержимое программного счетчика?
- Для чего предназначены регистры общего назначения?
- Какие операции можно выполнять с содержимым регистров общего назначения?
- Какая информация хранится в регистрах ввода/вывода?
- Какие операции можно производить в программе с содержимым регистров ввода/вывода?
- Какая информация хранится в регистре состояния (статуса)?
- В каких случаях в регистре состояния устанавливается флаг C (*Carry Flag*)?
- В каких случаях в регистре состояния устанавливается флаг Z (*Zero Flag*)?
- В каких случаях в регистре состояния устанавливается флаг V (*Overflow Flag*)?
- В каких случаях в регистре состояния устанавливается флаг H (*Half Carry Flag*)?
- В каких случаях в регистре состояния устанавливается флаг N (*Negative Flag*)?
- Где хранятся данные при их записи в стек?
- Как изменяется содержимое указателя стека при записи байта данных в стек?
- Как изменяется содержимое указателя стека при чтении байта данных из стека?
- Сохраняются ли данные в стеке при выключении питания?
- Где хранятся данные в инструкциях с непосредственной адресацией?
- Где могут храниться данные в инструкциях с прямой адресацией?

- Где хранится результат операции при выполнении инструкций с двумя регистрами?
- Где могут храниться данные в инструкциях с косвенной адресацией?
- Где могут храниться данные в инструкциях с относительной адресацией?
- Где могут храниться данные в инструкциях пересылки?
- Какие арифметические инструкции может выполнить микроконтроллер ATmega16?
- Какие логические инструкции может выполнить микроконтроллер ATmega16?
- Какие инструкции управления может выполнить микроконтроллер ATmega16?
- Для чего предназначены директивы ассемблера?
- Какие директивы позволяют присвоить символические имена регистрам?
- Какие директивы позволяют присвоить символические адреса ячейкам памяти?
- Какая директива позволяет задать начальный адрес программы в памяти программ?
- Какая директива позволяет подключать файлы библиотек в программу пользователя?

4. СИСТЕМНЫЙ ТАКТОВЫЙ ГЕНЕРАТОР

Работа микроконтроллера синхронизируется системным тактовым генератором *(System Clock)*, задающим время выполнения элементарных операций в ядре и различных устройствах ввода/вывода.

Схемы тактовых генераторов

Схемы тактовых генераторов могут быть различны. Они могут быть реализованы внутри микросхемы, либо обеспечиваться навесными элементами. Выбор той или другой схемы обеспечивается битами-предохранителями fuse-битами CKSEL3:0, программируемыми на стадии занесения программы. Кроме того, работу тактовых генераторов обеспечивают fuse-биты SUT1:0 *(Start-upTimes)*, задающие время задержки при включении питания, и CKOPT *(CK Options)*, задающие параметры кристаллического резонатора.

Внутренний тактовый генератор

Микросхема поставляется с fuse-битами, установленными в состояние CKSEL3:0 = 0001,SUT1:0 =10. Fuse-биты обеспечивают подключение внутреннего (реализованного на кристалле) тактового генератора, рассчитанного на тактовую частоту 1 МГц. Внутренний генератор не требует подключения внешних компонентов, но на частоту колебаний такого генератора влияет напряжение питания и температура.

Кристаллический осциллятор

К выводам XTAL1 и XTAL2 микросхемы может быть подключен кристаллический кварцевый или керамический осциллятор по схеме рис. 4.1.

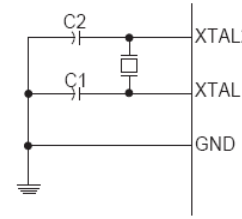


Рис. 4.1. Схема подключения кристаллического осциллятора

Подключенный осциллятор, вместе с конденсаторами C1 и C2 и встроенным в кристалл усилителем, образует схему тактового генератора, частота колебаний которого задается резонансной частотой кристалла. Рекомендуемые значения fuse-битов CKOPTи CKSEL3:1 и емкости конденсаторов приведены в таблице 4.1.

Таблица 4.1

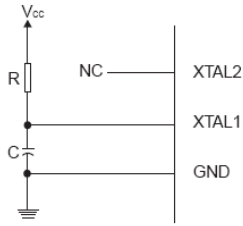
Рекомендуемые значения fuse-битов при подключении кристаллических осцилляторов

CKOPT	CKSEL3:1	Диапазон частот (МГ)	Рекомендуемая емкость (пФ)
1	101	0.4 - 0.9	-
1	110	0.9 - 3.0	12 – 22
1	111	3.0 - 8.0	12 – 22
0	101, 110, 111	1.0 ≤	12 - 22

RC-генератор

RC-генератор является простым решением, позволяющим изменять тактовую частоту. Внешняя времязадающая RC-цепь генератора подключается к выводам микроконтроллера по схеме рис. 4.2.

Сопротивление резистора R допускается варьировать в диапазоне от 3 до 100 кОм, а емкость C не должна быть менее 22 пФ. Частота сигнала приблизительно рассчитывается по формуле: $f \cong 1/(3RC)$. С помощью fuse-бита



CKOPT, пользователь может включить внутренний конденсатор емкостью C=36пФ.

Рис. 4.2. Схема подключения внешней RC-цепи

Генератор может работать в 4-х различных режимах, оптимизированных для определенного диапазона частот. Режимы переключаются fuse-битами CKSEL3:0 (таблица 4.2).

Таблица 4.2

Режимы работы генератора с внешней RC-цепью

CKSEL3:0	Диапазон частот (МГц)
0101	0.1 ≤ 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

Калиброванный внутренний RC-генератор

Калиброванный внутренний RC-генератор обеспечивает фиксированные тактовые частоты 1.0, 2.0, 4.0, или 8.0 МГц. Все частоты являются номинальными при напряжении питания 5В и температуре 25 °C. Эти частоты могут быть выбраны программированием CKSEL3:0 fuse-битов (табл. 4.3).

Таблица 4.3

Режимы работы внутреннего RC-генератора

CKSEL3:0	Номинальная частота (МГц)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

Во время аппаратного сброса в регистр OSCCAL (*Oscillator Calibration Register*) загружается калибровочный байт, позволяющий откалибровать частоты в пределах ± 3% от номинального значения.

Внешний генератор

Внешний генератор просто подключается к выводу XTAL1 микроконтроллера (рис. 4.3).

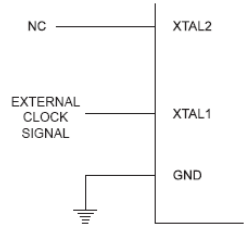


Рис. 4.3. Схема подключения внешнего генератора

В качестве внешнего генератора можно использовать любую схему генератора с ТТЛ-выходом. Внешние генераторы, как правило, обладают высокой стабильностью и предсказуемостью параметров.

Контрольные вопросы

- В чем заключаются преимущества генераторов с кварцевыми и керамическими резонаторами?
- Недостатки внутреннего тактового генератора?
- Недостатки внешнего тактового RC-генератора?
- Какие параметры тактовых генераторов задаются в процессе программирования?
- Можно ли в программе пользователя изменить частоту тактового генератора?
- От чего зависит максимальная частота тактового генератора микроконтроллера?
- Чему равна максимальная частота тактового генератора микроконтроллера ATmega16?
- - Чему равна максимальная частота тактового генератора AVR-микроконтроллеров?
- - Какие микроконтроллеры семейства AVR являются самыми быстрыми?

5. 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 нулевого начального адреса.

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 0x0 всегда приводит к сбросу устройства.

Источники сброса микроконтроллера ATmega16:

- Сброс при включении питания (*Power-on Reset*). Сброс происходит, если напряжение питания ядра ниже определенного порога (VPOT).
- Внешний сброс (*External Reset*). Сброс происходит при поступлении сигнала низкого уровня длительностью более 500 ns на внешний контакт RESET микросхемы.
- Сброс при кратковременном провале напряжения питания (*Brown-out Reset*). Сброс происходит, если напряжение питания контроллера в процессе работы опускается ниже определенного порога (V BOT).
- Сброс сторожевым таймером (*Watchdog Reset*). Сброс происходит по команде сторожевого таймера.
- Сброс через интерфейс периферийного сканирования JTAG (*Joint Test Action Group*).

По любой из этих причин микроконтроллер переходит к выполнению программы с адреса 0x0. В этой ячейке обычно размещают инструкцию jmp с адресом программы инициализации системы.

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 на контакте 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 5.1.

Таблица 5.1

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
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 0x34 (рис. 5.1).

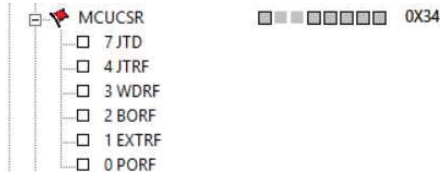


Рис. 5.1. Регистр MCUCSR

В системе сброса используются четыре младших бита регистра:

- 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 включении 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.

Сторожевой таймер

Сторожевой таймер *(Watchdog*) синхронизирован от отдельного внутреннего генератора на кристалле, работающего с частотой 1 МГц.

Задержка сброса устанавливается с помощью предделителя. 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. 5.2

Интервалы задержки сторожевого таймера

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. 5.2) 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.

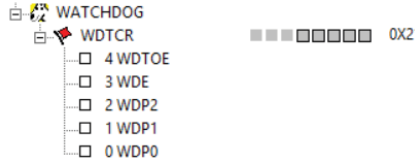


Рис. 5.2. Регистр управления сторожевым таймером WDTCR

- 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 (*Watchdog 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 5.2.

Контрольные вопросы

- С какого адреса начинается выполнение программы при сбросе микроконтроллера?
- Какие причины приводят к сбросу микроконтроллера ATmega16?
- Какой уровень сигнала на контакте RESET приводит к сбросу микроконтроллера ATmega16?
- При каком уровне напряжения питания происходит сброс микроконтроллера ATmega16?
- При каком кратковременном провале напряжения питания происходит сброс микроконтроллера?
- Содержимое какого регистра микроконтроллера ATmega16 позволяет определить источники сброса?
- От чего зависит время задержки от момента поступления одного из сигналов сброса до момента сброса?
- Чему равна максимальная задержка сторожевого таймера в контроллере ATmega16?
- Какая инструкция микроконтроллера предотвращает сброс контроллера от сторожевого таймера?
- Как включить сторожевой таймер?
- Как отключить сторожевой таймер?

6. СИСТЕМА ПРЕРЫВАНИЙ

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 поразному. 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.

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.

Вектора прерываний

Идентификация источника прерывания в системе может выполняться аппаратными средствами. При этом каждому источнику прерываний ставит-

ся в соответствие определенный адрес программы (вектор прерывания) и по этому адресу программист размещает команду перехода к соответствующей этому источнику подпрограмме обработки прерывания.

Например, контроллер ATmega16 имеет 20 различных источников прерывания: три внешних и 17 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 0x0 до 0x22. Самый младший вектор прерываний (с адресом 0x0) определен как сигнал сброса. Адрес каждого последующего вектора больше предыдущего на 2. Адреса 0x2 и 0x4 присвоены внешним входам запросов на прерывания (таблица 6.1).

Таблица 6.1

Вектора внешних прерываний контроллера ATmega16

Вектор	Адрес	Обозначение источника	Описание источника
2	0x02	INT0	Внешнее прерывание 0 (*External Interrupt Request0)*
3	0x04	INT1	Внешнее прерывание 1 (*External Interrupt Request1*)
19	0x24	INT2	Внешнее прерывание 2 (*External Interrupt Request 2*)

Полный список векторов приведен в приложении 3. Приоритет прерываний уменьшается с возрастанием адресов их векторов. Любое прерывание может быть запрещено (замаскировано) специальными битами в регистрах ввода/вывода микроконтроллера.

Типовая программа обработки векторов внешних прерываний выглядит следующим образом:

```
.cseg
.org0x0
jmpmain       ;       переход к программе пользователя
jmpext_int0 ; переход к подпрограмме обработки IRQ0
jmpext_int1 ; переход к подпрограмме обработки IRQ1
.org0x24
jmpext_int2;  переход к подпрограмме обработки IRQ2

main:         ; Начало программы пользователя
```

Для выхода из любой подпрограммы обработки прерывания в системе команд микроконтроллера ATmega16 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.

Регистры системы прерываний

Прерывания разрешаются только при единичном значении бита глобального разрешения прерываний I *(Interrupt)* в регистре состояния SREG (рис. 3.3). 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 прерывания.

Внешние прерывания вызываются с контактов INT0, INT11 и INT2. Для их обработки и программирования в контроллере задействованы отдельные биты четырех регистров (рис. 6.1):

- регистра управления прерываниями GICR (*General Interrupt Control Register*);
- регистра флагов прерываний GIFR (*General Interrupt Flags Register*);
- регистра управления процессорным ядром MCUCR (*MCU Control Register);*
- регистра управления и состояния процессорного ядра MCUCSR (*MCU Control and Status Register).*

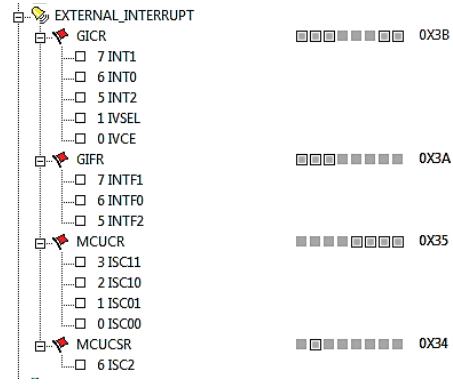


Рис. 6.1. Регистры системы прерываний микроконтроллера ATmega16

Биты 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 6.2. Импульсы короче одного такта не гарантируют генерирование прерывания.

Таблица 6.2

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.
- Бит ISC2 (*Interrupt Sense Control 2)* в регистре MCUCSR задает фронт сигнала на входе INT2, активизирующий прерывание. Если бит очищен, прерывание активизируется по падающему фронту сигнала, если бит установлен – по нарастающему.

В регистре управления GICR прерывания могут быть замаскированы. Прерывания разрешаются, только если соответствующие входам биты INT0<INT1, INT2 установлены.

В регистре флагов GIFR содержатся флаги внешних прерываний. При поступлении запросов на входы соответствующие биты устанавливаются, а когда счетчик команд микроконтроллера устанавливается на конкретный вектор прерывания, соответствующий бит в регистре GICR аппаратно сбрасывается. Флаги запросов на прерывания можно очистить, записав в соответствующий бит регистра логическую единицу.

- Бит 7 - INTF1: 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 - INTF0: 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;
- Бит 5 – INTF2: внешнее прерывание INT2 разрешено. Если бит INT2 установлен и бит I в регистре статуса SREG также равен 1, то внешний вход запроса на прерывание INT0 становится активным.

Контрольные вопросы

- Какие прерывания микроконтроллера считаются внешними?
- Что происходит в микроконтроллере при обработке прерывания?
- Какая инструкция позволяет работать системе прерываний?
- Какая инструкция запрещает все прерывания?
- Как разрешить отдельные прерывания?
- Когда в контроллере устанавливается флаг запроса на прерывание?
- Что такое вектор прерывания?
- Какая инструкция обеспечивает выход из прерывающей программы?
- Чем определяется приоритет запроса на прерывание?
- Что произойдет, если одновременно придет несколько запросов на прерывание?
- Можно ли прервать подпрограмму, выполняемую по запросу на прерывание?
- Сколько внешних входов запросов на прерывания имеет микроконтроллер ATmega16?
- Какие вектора прерываний соответствуют внешним входам запросов микроконтроллера ATmega16?
- Какие изменения сигналов на внешних входах приводят к запросам на прерывания?
- Какие регистры задействованы в системе внешних прерываний микроконтроллера ATmega16?
- В каком регистре хранятся флаги внешних запросов на прерывания микроконтроллера ATmega16?
- Какой регистр предназначен для маскирования внешних прерываний микроконтроллера ATmega16?

7. ПОРТЫ ВВОДА-ВЫВОДА

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 инструкции.

Микроконтроллер ATmega16 имеет 32 линии ввода-вывода с индивидуальной настройкой, сгруппированные 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. 7.1):

- 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.

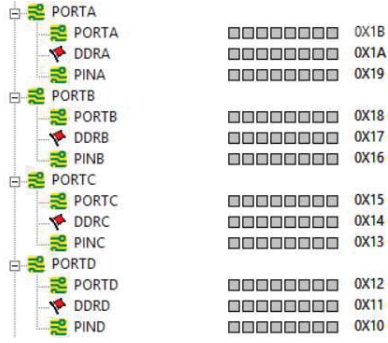


Рис. 7.1. Регистры портов ввода/вывода

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)* (рис. 7.2).

Рис. 7.2. Регистр специальных функций ввода/вывода

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

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.

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.

Программирование ввода/вывода

Ввод и вывод данных в микроконтроллере 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 ввода присутствуют на линиях порта, при выводе – данные будут прочитаны с линий порта до следующего вывода.

Например, микроконтроллер ATmega16 осуществляет вывод данных на контакты порта A при выполнении фрагмента программы:

```
ldir16,0xFF        ;запись 0xFF в r16
out ddra, r16      ;включение порта A на вывод
out porta, r0      ;вывод данных из регистра r0 в порт A.
```

Время выполнения инструкции вывода равно двум периодам тактового сигнала.

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

```
ldir16,0x0          ;запись 0 в r16
out ddra, r16       включение порта А на ввод данных
in r0, pina         ; ввод данных с контактовPINA в регистр r0.
```

Контрольные вопросы

- Для чего в портах ввода вывода предусмотрены регистры DDRx (x∈ A,B,C,D)?
- Можно ли выводить данные непосредственно в регистры PINx (x∈ A,B,C,D)?
- Сколько тактов выполняется инструкция вывода?
- При каком условии к линиям ввода/вывода подключаются подтягивающие резисторы?
- В каком направлении передачи данных находится порт по умолчанию?
- Надо ли в программе задавать направление передачи данных перед каждой операцией ввода/вывода?
- Могут ли отдельные линии порта иметь различные направления передачи данных?
- В каком направлении передачи данных подключается линия порта при записи 1 в соответствующий бит регистра направления?

8. ТАЙМЕРЫ-СЧЕТЧИКИ

Большинство задач управления решаются в реальном времени. При этом микроконтроллер должен в заданные 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*). Тай-

мер/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:

- захвата (*Capture);*
- сравнения (*Compare*);
- широтно-импульсной модуляции PWM *(Pulse-Width Modulation*);
- часов реального времени RTC *(Real Times Clock).*

В режиме захвата содержимое таймера/счетчика в момент времени, задаваемый каким-либо внешним событием на входе захвата, запоминается в специальном регистре захвата 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, хранящимся в специальном регистре сравнения микроконтроллера. В момент равенства данных формируется сигнал запроса на прерывание и сигнал на одном из контактов микросхемы.

В режиме широтно-импульсной модуляции таймер/счетчик формирует на одном из выходов микросхемы последовательность импульсов определенной частоты, в которой длительность импульсов может быть изменена программно.

В режиме реального времени таймер/счетчик производит отсчет времени в общепринятых физических единицах, секундах, минутах, часах.

Количество таймеров/счетчиков, интегрируемых на кристалл микроконтроллера, может быть различно. Например, микроконтроллер ATmega16 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. 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 также к отдельному тактовому генератору. Для этого к определенным выводам микроконтроллера (TOSC1 и TOSC2) подключается кристалл резонатора. Генератор оптими-

зирован на частоту кристалла 32768 кГц. Это позволяет использовать таймер/счетчик 2 в качестве часов реального времени RTC (*Real Time Clock).*

Таймер/счетчик 0

Схема

8-разрядный таймер/счетчик 0 (*Timer*/*Counter0*) тактируется сигналом синхронизации процессорного ядра СК (*Clock*) от встроенного предделителя, или от внешнего контакта Т0 (рис. 8.1), поддерживает 4 различных режима работы.

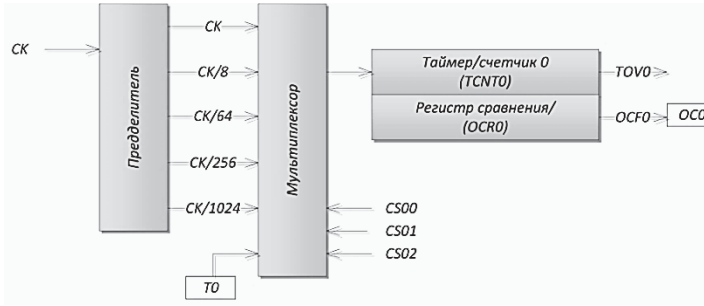


Рис. 8.1. Таймер/счетчик 0 микроконтроллера ATmega16

Вход таймера/счетчика может быть подключен к источнику тактовой частоты процессорного ядра СК *(Clock)* через предделитель, обеспечивающий изменение частоты на его входе в диапазоне от 1/8 до 1/1024 СК. Кроме того, вход счетчика можно подключить к внешнему контакту микросхемы Т0. Управление подключением входа осуществляют биты CS0_*(Clock Select 0)*: CS02, CS01,CS00 регистра TCCR0 в соответствии с таблицей 8.1.

Таблица 8.1

Коммутация входа таймера/счетчика 0

CS02	CS01	CS00	Источник сигнала
0	0	0	Остановка таймера
0	0	1	СК
0	1	0	СК / 8
0	1	1	СК / 64
1	0	0	СК / 256
1	0	1	СК / 1024
1	1	0	Внешний контакт Т0, задний фронт
1	1	1	Внешний контакт Т0, передний фронт

Регистры

Для обеспечения работы схемы используется 5 регистров (рис. 8.2):

- регистр управления TCCR0 *(Timer/Counter0 Control Register)*,
- регистр данных TCNT0 *(Timer/Counter0)*,
- регистр выхода сравнения OCR0 *(Output Compare Register 0)*,
- регистр маски прерывания TIMSK (*Timer/Counter Interrupt Mask Register*),
- регистр флагов прерывания TIFR (*Timer/Counter Interrupt Flag Register*).

Регистр TCNT0 хранит результаты счета. Счет идет от минимального значения BOTTOM = 0 до максимального значения TOP = 255. При переполнении счетчика он обнуляется и устанавливается флаг переполнения TOV0 *(Timer/Counter 0 Overflow)*в регистре TIFR и, если в регистре TIMSK установлен бит TOIE0 *(Timer/Counter 0 Interrupt Enable)*, генерируется запрос на прерывание. Соответствующий вектор прерывания равен 0x10.

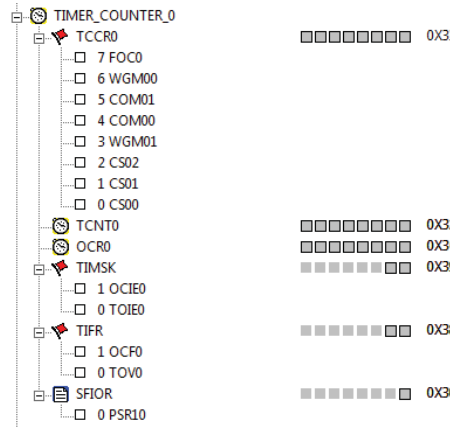


Рис. 8.2. Регистры, участвующие в управлении таймером/счетчиком 0

Регистр TCNT0 доступен для чтения и записи, поэтому программа пользователя может в любой момент считать содержимое регистра TCNT0 или записать в него константу. В последнем случае счет будет происходить от записанного значения. Флаг переполнения TOV0 может быть сброшен пользователем, либо он будет сброшен автоматически при переходе процессора на вектор прерывания.

Контакт микросхемы OC0 *(Output Compare0)* предназначен для использования в различных режимах с использованием регистра выхода сравнения OCR0. Содержимое регистра TCNT0 непрерывно сравнивается с содержимым регистра OCR0. При совпадении устанавливается флаг OCF0 *(Output Compare Flag 0)* в регистре TIFR. Если до этого момента был установлен бит OCIE0 *(Output Compare Interrupt Enable 0)* в регистре TIMSK и глобальный флаг прерывания Iв регистре SREG, то генерируется запрос на прерывание TIMER2 COMP с вектором 0x06. Флаг OCF0 автоматически очищается при переходе на вектор прерывания. Он также может быть сброшен программно путем записи логической единицы в соответствующий разряд регистра TIFR. Флаг OCF0 можно программно связать с выходом сравнения OC0 микросхемы. Результат зависит от состояния битов COM01 и COM00 *(Compare Match Output Mode 0)* регистра TCCR0 и от выбранного режима сравнения.

Бит FOC0 (*Force Output Compare 0*) в регистре TCCR0 позволяет изменить сигнал на выводе OC0 относительно значений установленных битами COM01 и COM00. Изменение происходит при установке бита. При этом никакое прерывание не генерируется и таймер не очищается.

Биты WGM01 иWGM00 *(Waveform Generation Mode)* обеспечивают 4 режима сравнения (таблица 8.2).

Таблица 8.2

Режимы таймера/счетчикаTCNT0

№	WGM01	WGM00	Режим
0	0	0	normal
1	0	1	PWM, PhaseCorrect
2	1	0	CTC
3	1	1	Fast PWM

***Нормальный режим* (*Normal Mode*)**

При WGM00:1 = 0 включается нормальный режим. В этом режиме счетчик TCNT0 является суммирующим и считает до переполнения. Переполнение происходит при TOP = 255. При этом устанавливается флаг переполнения OV0 в регистре флагов, и счетчик сбрасывается в ноль. Флаг может быть очищен при переходе к соответствующей прерывающей программе. При сравнении устанавливается флаг сравнения OC0 и возможен переход к подпрограмме обработки прерывания по сравнению.

Режим очистки таймера по совпадению (CTC Mode)

При WGM00:1 = 2 включается режим очистки таймера по совпадению CTC *(Clear Timer on Compare Match Mode)*. При этом счетчик TCNT0 обнуляется, когда его состояние совпадает с состоянием регистра сравнения OCR0.

Временная диаграмма для режима CTC показана на рисунке 8.3. Значение счетчика TCNT0 увеличивается до момента равенства TCNT0 и OCR0, а затем счетчик TCNT0 очищается.

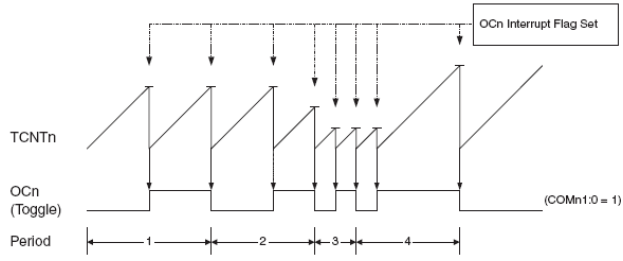


Рис. 8.3. Временные диаграммы работы таймера/счетчика 0 в режиме CTC

Каждый раз, когда значение счетчика TCNT0 достигает OCR0, устанавливается OCF0 флаг. Если прерывание разрешено, то обработчик прерывания можно использовать для обновления содержимого регистра сравнения OCR0.

Для генерации выходного сигнала в режиме CTC на выходе сравнения OC0 можно установить переключение его логического уровня в соответствии с таблицей 8.3 при каждой установке флага OCIE0.

Таблица 8.3

Управление выходом сравнения OC0 в режиме CTC

COM01	COM00	Результат
0	0	Выход отключен
0	1	Переключение выхода OC0 при установке флага
1	0	Очистка выхода OC0 при установке флага
1	1	Установка выхода OC0 при установке флага

Частота сигнала на выходе OC0 определяется следующим уравнением:

$$f_{OC0} = \frac{CK}{2N(1+OCR0)},$$

где N – коэффициент деления предделителя (1, 8, 64, 256 или 1024).

Быстрый режим ШИМ (Fast PWM Mode)

Быстрый режим широтно-импульсной модуляции (*Fast PWM Mode*) включается при WGM01:0 = 3. При этом счетчик TCNT0 считает до переполнения. Каждый раз при сравнении устанавливается флаг OCF0 и при переполнении - флаг TOV0 (рис. 8.4).

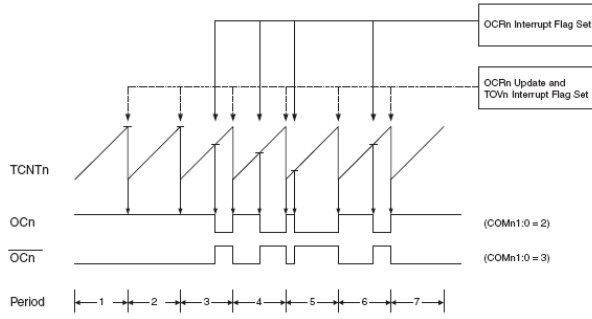


Рис. 8.4. Временные диаграммы работы таймера/счетчика 0 в режиме Fast PWM

Если прерывание разрешено, то обработчик прерываний может быть использован для обновления значения в регистре сравнения OCR0.

Режим Fast PWM позволяет сформировать ШИМ сигнал на контакте OC0. Установка битов COM01 и COM00 в регистре TCCR0 в соответствии с таблицей 8.4 позволяет получить неинвертированный или инвертированный ШИМ-сигнал, как показано на рисунке 8.4.

Таблица 8.4

Управление выходом OC0 в режиме FastPWM

COM01	COM00	Результат
0	0	Выход отключен
0	1	Резерв
1	0	Сброс OC0 при сравнении, установка OC0 при обнулении, (*non-inverting mode*)
1	1	Установка OC0 при сравнении, сброс OC0 при обнулении, (*inverting mode*)

Частота ШИМ для вывода может быть рассчитана по следующей формуле

$$f_{OC0} = \frac{CK}{256N},$$

где N – коэффициент деления предделителя (1, 8, 64, 256 или 1024).

Режим ШИМ с коррекцией фазы (*Phase Correct PWM Mode*)

Режим *Phase Correct PWM* включается при WGM01:0 = 1. В этом режиме счетчик считает от 0 до максимального значения MAX = 255, а далее в обратном порядке от MAX до 0 (рис. 8.5).

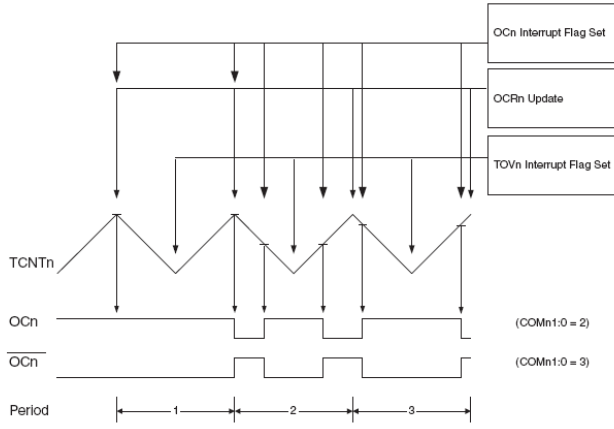


Рис. 8.5. Временные диаграммы работы таймера/счетчика 0 в режиме Phase Correct PWM

Настройка битов COM01 и COM00 в соответствии с таблицей 8.4 позволяет получить инвертированный или неинвертированный сигнал ШИМ на выходе OC0.

Частота ШИМ для вывода OC0 может быть рассчитана по формуле:

$$f_{OC0} = \frac{CK}{510N},$$

где N – коэффициент деления предделителя (1, 8, 64, 256 или 1024).

Таймер/счетчик 1

Схема

16-битный таймер/счетчик Timer/Counter1 микроконтроллера ATmega16 доступен процессорному ядру для чтения и записи, может считать импульсы синхронизации CK, импульсы с выхода предделителя или импульсы с контакта микросхемы T1, имеет внешний вход захвата ICP и два выхода сравнения OC1Aи OC1B, поддерживает 6 различных режимов работы.

Структурная схема таймера/счетчика 1 изображена на рисунке 8.6.

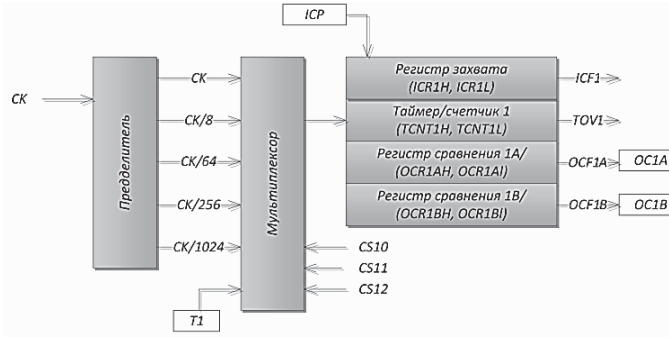


Рис. 8.6. Таймер/счетчик 1 микроконтроллера ATmega16

Таймер/счетчик TCNT1 считает импульсы со входа T1 или, через предделитель, тактовые импульсы СК. Переключением входа управляют биты CS10….CS12, размещенные в регистре управления. При переполнении счетчика формируется флаг запроса на прерывание TOV1.

При воздействии на вход захвата ICPсодержимое счетчика копируется в специальный 16-битный регистр захвата ICR1. В этом случае устанавливается флаг запроса на прерывание ICF1.

В процессе работы содержимое счетчика сравнивается с содержимым специальных регистров сравнения OCR1A и OCR1B. В моменты равенства формируются флаги запросов на прерывания OCF1A и OCF1B и могут изменяться состояния выходов OC1A и OC1B.

Регистры

Работу счетчика обеспечивают 8-битные регистры, доступные процессорному ядру в пространстве регистров ввода/вывода (рис. 8.7).

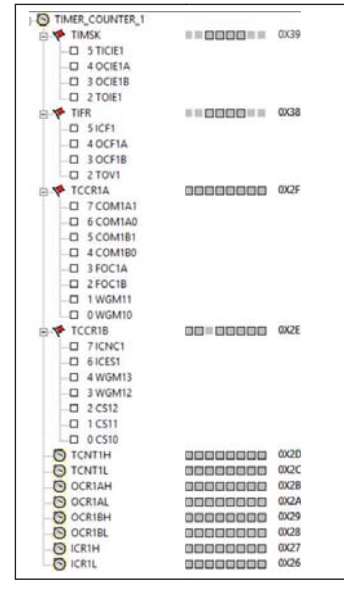


Рис. 8.7. Регистры таймера/счетчика 1

Регистр маски прерываний TIMSK

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

- бит5 - TICIE1 (*Timer/Counter1 Input Capture Interrupt Enable*) – бит разрешения прерывания при захвате;
- бит 4 - OCIE1A (*Timer/Counter1 Output Compare A Match Interrupt Enable)* – бит разрешения прерывания при равенстве содержимого счетчика и содержимого регистра сравнения OCR1A.
- бит 3 - OCIE1B*(Timer/Counter1 Output Compare B Match Interrupt Enable)*–бит разрешения прерывания при равенстве содержимого счетчика и содержимого регистра сравнения OCR1B.
- бит 2 - TOIE1 *(Timer/Counter1 Over flow Interrupt Enable)* – бит разрешения прерывания при переполнении таймера/счетчика 1.

Регистр флагов прерываний TIFR

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 захвате;
- 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 Over flow Flag)* – флаг прерывания при переполнении таймера/счетчика 1.

Регистр управления TCCR1A

Регистр управления TCCR1A *(Timer/Counter 1 Control Register A)* управляет работой таймера/счетчика 1:

- Биты 7,6,5,4 - COM1A1:0 (*Compare Output Mode1A, bits 1 and 0*) и– COM1B1:0 (*Compare Output Mode1B, bits 1 and 0*) биты управления для выходов OC1AиOC1B. Определяют характер выходного сигнала при сравнении содержимого таймера/счетчика 1 с регистрами OCR1A и OCR1B. Конфигурирование выходов представлено в таблице 8.5.

Таблица 8.5

Конфигурирование выходов OC1A и OC1B

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.

- Бит 3,2 - FOC1A (*Force Output Compare1A*) и FOC1B*(Force Output Compare 1B)* прямое воздействие на выходы OC1Aи OC1B. Запись логической единицы в эти биты приводит к изменениям на выводах OC1A и OC1B относительно значений, установленных битами COM1A1 и COM1A0; бит может использоваться, чтобы изменить состояние выходов, при этом никакое прерывание не генерируется и таймер не очищается.

- Биты 1, 0 – WGM 11:0 –в сочетании с битами WGM13:2, размещенными в регистре TCCR1B, задают режим работы таймера/счетчика и максимальное (TOP) значение счетчика.

Таблица 8.6

Режимы работы таймера/счетчика TCNT1

Режим	WGM13	WGM12	WGM11	WGM10	Описание
0	0	0	0	0	Normal
1	0	0	0	1	PWM, Phase Correct, 8-bit
2	0	0	1	0	PWM,Phase Correct, 9-bit
3	0	0	1	1	PWM, Phase Correct, 10-bit
4	0	1	0	0	CTC
5	0	1	0	1	Fast PWM, 8-bit
6	0	1	1	0	Fast PWM, 9-bit
7	0	1	1	1	Fast PWM, 10-bit
8	1	0	0	0	PWM, Phase and Frequency Correct
9	1	0	0	1	PWM, Phase and Frequency Correct
10	1	0	1	0	PWM, Phase Correct
11	1	0	1	1	PWM, Phase Correct
12	1	1	0	0	CTC
13	1	1	0	1	резерв
14	1	1	1	0	Fast PWM
15	1	1	1	1	Fast PWM

Регистр управления TCCR1B

В регистре управления TCCR1B *(Timer/Counter1 Control Register B)* находятся биты для переключения входа счетчика/таймера 1.

- 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.
- Бит 5 - резервный бит.
- Биты 4,3 – WGM13:2: *(Waveform Generation Mode)* задают режим работы счетчика совместно с битами WGM11:0 регистра TCCR1B.
- Биты 2..0 - CS12:0 (*Clock Select1, Bit 2,1, and 0*) управляют подключением входа таймера/счетчика 1 и внешнего контакта T1 в соответствии с таблицей 8.7.

Таблица 8.7

Управление входом таймера/счетчика T1

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 и TCNT1L

Регистры данных TCNT1HиTCNT1L (*Timer/Counter1 High and Low Register)* образуют 16-разрядный регистр, содержащий текущее значение 16-разрядного таймера/счетчика 1.

При 16-разрядных операциях записи обращение к старшему байту (TCNT1H) должно выполняться первым. При использовании таймера/ счетчика 1 в качестве 8-разрядного таймера достаточно производить запись только младшего байта.

При 16-разрядных операциях чтения первым должно выполняться обращение к младшему байту (TCNT1L). При использовании таймера/счетчика 1 в качестве 8-разрядного таймера достаточно производить запись только младшего байта.

Регистр входа захвата

Регистры ICR1H и ICR1L (*Timer/Counter1 Input Capture Register High and Low Register)* образуют 16-битный регистр входа захвата 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 захвата ICR1. Одновременно устанавливается в состояние 1 флаг захвата входа ICF1 *(Input Capture Flag 1)* в регистре флагов прерывания TIFR.

При чтении всего 16-разрядного регистра операцию чтения необходимо начинать с младшего байта ICR1L.

Регистры выходов сравнения

Регистры OCR1AH, OCR1AL *(Output Compare Register 1 A High and Low Register)*, OCR1BH и OCR1BL *(Output Compare Register B High and Low Register)* образуют два 16-разрядных регистра выходов сравнения 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/счетчика 1. Совпадение может произойти, только если таймер/счетчик 1 досчитает значения содержимого OCR1A или OCR1B.

Совпадения при сравнении устанавливают флаги прерывания по совпадению OCF1A и OCF1A. Старшие байты 16-битных регистров выходов сравнения OCR1A и OCR1B должны записываться первыми.

Нормальный режим (Normal Mode)

Нормальный режим включается при WGM13:0 = 0 (табл. 8.6). В этом режиме счетчик считает на сложение до максимального 16-битного значения (MAX = 0xFFFF) и сбрасывается в 0. При сбросе устанавливается флаг переполнения (TOV1). Установка флага переполнения приводит к формированию запроса на прерывание TIMER1 OVF *(Timer/Counter1 Overflow)*с вектором прерывания 0x10. Бит разрешения прерывания TOIE1 размещен в регистре TIMSK. При переходе на вектор прерывания флаг TOV1 автоматически сбрасывается.

Состояние счетчика можно изменить программно в любое время записью нового значения в регистр TCNT1Hи TCNT1L. При 16-разрядных операциях записи обращение к старшему байту (TCNT1H) должно выполняться первым. При использовании таймера/ счетчика 1 в качестве 8-разрядного таймера достаточно производить запись только младшего байта.

При появлении сигнала на входе захвата ICP содержимое таймера счетчика копируется в регистры входа захвата ICR1H и ICR1L. Сигналом может быть нарастающий или спадающий фронт импульса на входе ICP в зависимости от значения бита ICES1 *(Input Capture1 Edge Select)* в регистре управления TCCR1B. Одновременно в регистре флагов TIFR устанавливается флаг ICF1 и формируется запрос на прерывание TIMER1 CAPT *(Timer/Counter1 Capture Event)* с вектором прерывания 0x0A. Бит разрешения прерывания TICIE1 размещен в регистре TIMSK. При переходе на вектор прерывания флаг ICF1 автоматически сбрасывается.

Режим очистки таймера по совпадению (CTC Mode)

Режим очистки таймера по совпадению (*Clear Timer on Compare Match (CTC) Mode)*включается при WGM13: 0 = 4 или 12. В режиме CTC счетчик обнуляется, когда значение счетчика TCNT1 соответствует либо значению регистра OCR1A (WGM13: 0 = 4) или регистра ICR1 (WGM13: 0 = 12) (рис. 8.8).

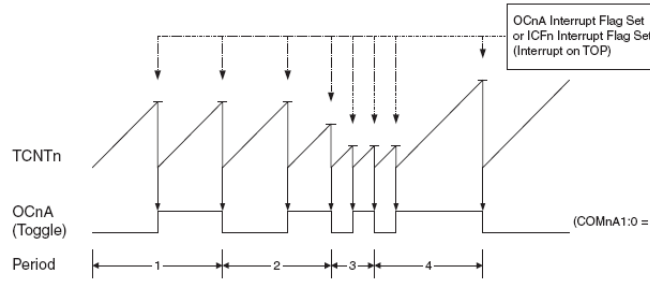


Рис. 8.8. Временные диаграммы таймера/счетчика 1 в режиме CTC

Регистры OCR1A или ICR1 определяют наибольшее значение в счетчике.

Запрос на прерывание может быть сформирован либо с помощью OCF1A или ICF1. Если соответствующее прерывание разрешено, то при переходе к подпрограмме произойдет сброс установленного флага запроса.

Для генерации выходного сигнала в режиме CTC, выход OC1A с помощью бит COM1A1: 0 = 1 (таблица 8.5) может быть установлен в режим переключения его логического уровня при каждом запросе.

Частота сигнала на выходе OC0 определяется следующим уравнением:

$$f_{OC0} = \frac{CK}{2N(1+OCR1A)},$$

где N – коэффициент деления предделителя (1, 8, 64, 256 или 1024).

Быстрый режим ШИМ (Fast PWM Mode)

Быстрый режим ШИМ включается при WGM13: 0 = 5,6,7,14 или 15. В этом режиме счетчик считает на сложение до максимального (TOP) значения, а затем сбрасывается в 0 (рис. 8.9).

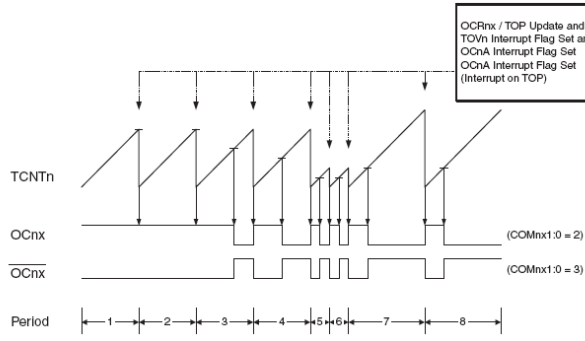


Рис. 8.9. Временные диаграммы таймера/счетчика 1 в режиме Fast PWM

Режим Fast PWM позволяет сформировать ШИМ сигнал на контактах OC1A и OC1A 0. Установка битов COM1A1:0 и COM1B1:0 в регистре TCCR1 в соответствии с таблицей 8.5 позволяет получить неинвертированный или инвертированный ШИМ-сигнал, как показано на рисунке 8.9.

Режим ШИМ с корректной фазой (Phase Correct PWM Mode)

В режиме ШИМ с корректной фазой (*Phase Correct PWM)* счетчик считает на сложение от 0 до максимального TOP значения и, далее, на вычитание от TOP до нуля (рис. 8.10).

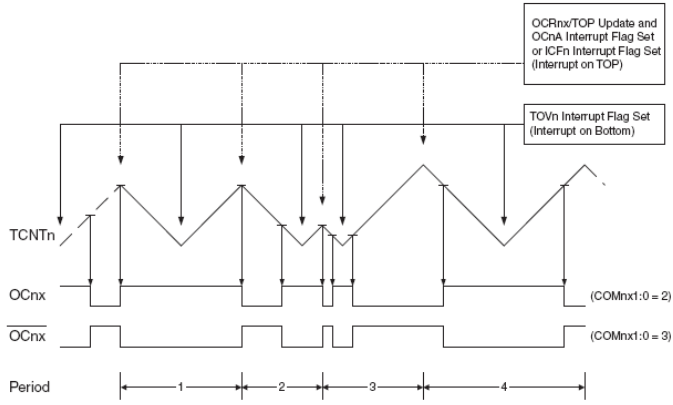


Рис. 8.10. Временные диаграммы таймера/счетчика 1 в режиме Phase Correct PWM

Значения TOP могут быть фиксированными 8-, 9-, или 10 битов, определяться содержимым регистра захвата ICR1 или регистра сравнения OCR1A (таблица 8.8).

Таблица 8.8

Значения TOP для режимов PWM, Phase Correct

Режим	Описание	TOP
1	PWM, Phase Correct, 8-bit	0x00FF
2	PWM, Phase Correct, 9-bit	0x01FF
3	PWM, Phase Correct, 10-bit	0x03FF
10	PWM, Phase Correct	ICR1
11	PWM, Phase Correct	OCR1A

Минимальное значение составляет 2 бита (ICR1 или OCR1A), максимальное - 16 бит (ICR1 или набор OCR1A).

Установка бит COM1x1:0 = 2 приводит к формированию неинвертированной ШИМ, а инвертированная ШИМ формируется при COM1x1:0 = 3 (табл. 8.5).

В неинвертированной ШИМ при счете на сложение сигнал на выходе OC1x сбрасывается при равенстве TCNT1 и OCR1x, а при счете на сложение - устанавливается. Если сравниваемое значение в регистре OCR1x превышает значение TOP, то выходной сигнал OCR1x изменяется на противоположный при каждом достижении TCNT1 TOP значения.

В этом режиме частота выходного ШИМ сигнала в 2 раза меньше, чем в режиме FASTPWM, однако из-за симметричного характера этот способ предпочтителен для применений в задачах управления электродвигателями.

Флаг сравнения OC1x всегда устанавливается при равенстве TCNT1 и OCR1x.

Флаг переполнения TOV1 устанавливается каждый раз, когда счетчик обнуляется. Когда регистры OCR1A или ICR1 используются для задания значения TOP, флаги OC1A или ICF1 устанавливаются в процессе выполнения операций сравнения или захвата.

Режим ШИМ с корректной фазой и частотой (Phase and Frequency Correct PWM Mode)

Режим *Phase and Frequency Correct PWM* включается при (WGM13: 0 = 8 или 9). В этом режиме, как и в предыдущем, счетчик считает на сложение от 0 до максимального TOP значения и, далее, на вычитание от TOP до нуля (рис. 8.11).

Значения TOP задаются содержимым регистра захвата ICR1 (WGM13: 0 = 8) или регистра сравнения OCR1A (WGM13: 0 = 9).

Флаги запросов на прерывания OC1A или ICF1 устанавливаются, когда TCNT1 достигает значения TOP. Они могут быть использованы, чтобы генерировать прерывание каждый раз при достижении счетчиком верхнего или нижнего значения.

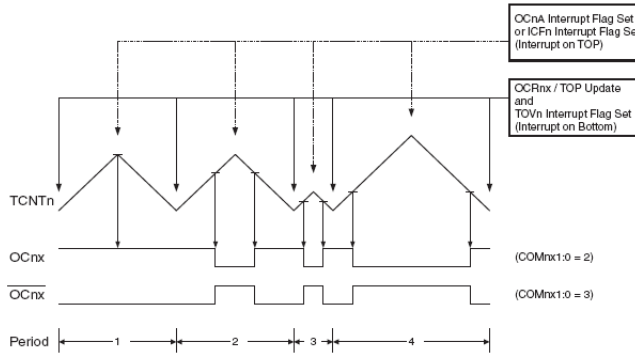


Рис. 8.11. Временные диаграммы таймера/счетчика 1 в режиме Phase and Frequency Correct PWM

Если TOP значение меньше, чем содержимое любого регистра сравнения, сравнение не происходит и сигнал на выходе не меняется. Поэтому выходной сигнал, в отличие от режима *Phase and Frequency Correct PWM Mode,* генерируется симметрично во все периоды работы счетчика.

Неинвертированный ШИМ сигнал на выходе OC1x формируется при COM1x1:0 = 2, инвертированный – при COM1x1:0 = 3 (таблица 8.5).

Таймер/счетчик 2

8-разрядный таймер/счетчик 2 (*Timer*/*Counter*2) тактируется сигналом синхронизации процессорного ядра СК (*Clock*) или от внутреннего генератора, формирующего сигнал TOSC1, поддерживает 4 различных режима работы.

Схема

8-разрядный таймер/счетчик 2 (*Timer/Counter 2*) микроконтроллера *ATmega16* может работать от генератора тактовых импульсов процессорного ядра СК. Программно его можно подключить и к отдельному тактовому генератору. Для этого к определенным выводам микроконтроллера (TOSC1 и TOSC2) подключается кристалл резонатора с резонансной частотой $2^{15} = 32768$ Гц (рис. 8.12).

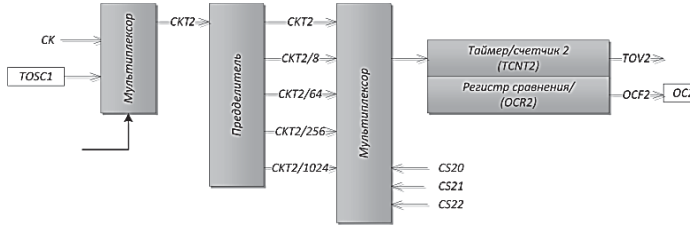


Рис. 8.12. Таймер/счетчик 2

Встроенный генератор с помощью внутреннего предделителя позволяет сформировать интервалы времени, равные или кратные 1 секунде, и реализовать в микроконтроллере режим часов реального времени *(RTC – Real Time Clock).*

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 моди-

фицируется состояние выхода OC2. Возможна его работа также и в режиме широтно-импульсной модуляции.

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

Регистры

В работе таймера/счетчика 2 задействованы следующие регистры (рис. 8.13):

- Регистр маски прерывания TIMSK *(Timer/Counter Interrupt Mask Register),*
- регистр флагов прерывания TIFR *(Timer/Counter Interrupt Flags Register);*
- регистр управления TCCR2 *(Timer/Counter Control Register 2);*
- регистр счетчика TCNT2 *(Timer/Counter 2);*
- регистр выхода сравнения OCR2 *(Output Compare Register 2);*
- асинхронный регистр состояния ASSR *(Asynchronous Status Register);*
- регистр специальных функций ввода/вывода SFIOR *(Special Function Input /Output Register).*

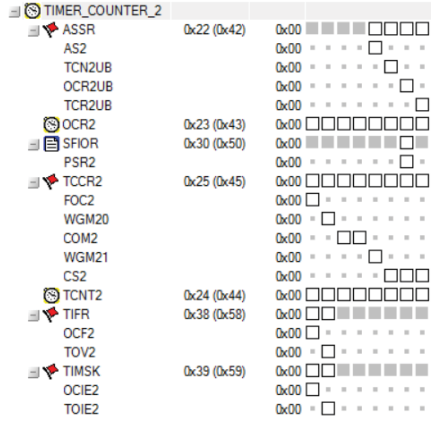


Рис. 8.13. Регистры таймера/счетчика 2

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

- бит 7 - OCIE2 *(Output Compare Match Interrupt Enable 2)* – 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, содержащий следующие биты:

- 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.

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.

Регистр выхода сравнения OCR2 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 6,3–WGM21:0*(Waveform Generation Mode)* – задание режима работы счетчика в соответствии с таблицей 8.9.

Таблица 8.9

Режимы работы таймера/счетчика 2

Mode	WGM21	WGM20	Timer/Counter Mode of Operation	TOP
0	0	0	Normal	0xFF
1	0	1	PWM, Phase Correct	0xFF
2	1	0	CTC	OCR2
3	1	1	Fast PWM	0xFF

- Бит 5,4 - COM21:0 (*Compare Output Mode, bits 1 and 0*) – биты конфигурирования выхода сравнения. Состояние этих битов определяет сиг-

нал на выводе OC2 микросхемы. Результат также зависит от режима работы таймер/счетчика.

- 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 8.10.

Таблица 8.10

Выбор тактового сигнала таймера/счетчика 2

CS22	CS21	CS20	Описание
0	0	0	Timer/Counter2 остановлен
0	0	1	CKT2
0	1	0	CKT2/8
0	1	1	CKT2/32
1	0	0	CKT2/64
1	0	1	CKT2/128
1	1	0	CKT2 / 256
1	1	1	CKT2 / 1024

Нормальный режим (Normal Mode)

При WGM21:0 = 0 (табл. 8.9) включается нормальный режим. В этом режиме счетчик TCNT2 является суммирующим и считает до переполнения. Переполнение происходит при TOP = 255. При этом устанавливается флаг переполнения TOV2 в регистре флагов, и счетчик сбрасывается в ноль. Флаг может быть очищен при переходе к соответствующей прерывающей программе. При сравнении устанавливается флаг сравнения OCF2и возможен переход к подпрограмме обработки прерывания по сравнению.

Вывод микросхемы OC2 может быть связан с флагом сравнения с помощью битов COM21:0 регистра TCCR2 (табл. 8.11).

Таблица 8.11

Конфигурирование выхода OC2 в режимах Normal и CTC

COM21	COM20	Описание
0	0	Таймер/счетчик 2 отключен от вывода OC2
0	1	Переключение выходной линии OC2
1	0	Очистка выходной линии OC2
1	1	Установка выходной линии OC2

Режим очистки таймера по совпадению (CTCMode)

При WGM211:0 = 2 включается режим очистки таймера по совпадению CTC *(Clear Timer on Compare Match Mode)*. При этом счетчик TCNT2 считает на сложение до момента равенства TCNT2 и OCR0, а затем счетчик TCNT2 обнуляется (рис. 8.14).

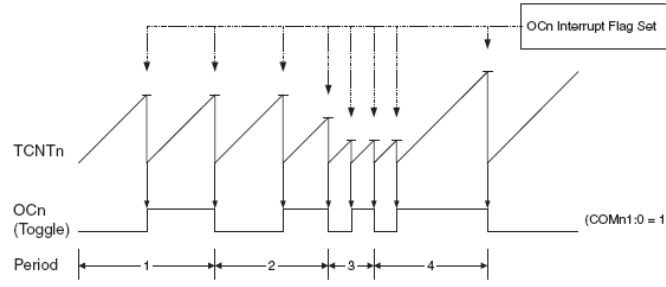


Рис. 8.14. Временные диаграммы таймера/счетчика 2 в режиме CTC

Флаг OCF2 устанавливается при обнулении счетчика и сбрасывается при переходе к подпрограмме обработки прерывания с вектором 0x06. Установка флага может быть связана с состоянием вывода OC2 с помощью битов COM21 и COM20 регистра TCCR2 (табл. 8.10).

Быстрый режим ШИМ (Fast PWM Mode)

Быстрый режим широтно-импульсной модуляции *(Fast PWM Mode)* включается при WGM21:0 = 3. При этом счетчик TCNT2 считает до переполнения. Каждый раз при сравнении устанавливается флаг OCF2 и при переполнении - флаг TOV2 (рис. 8.15).

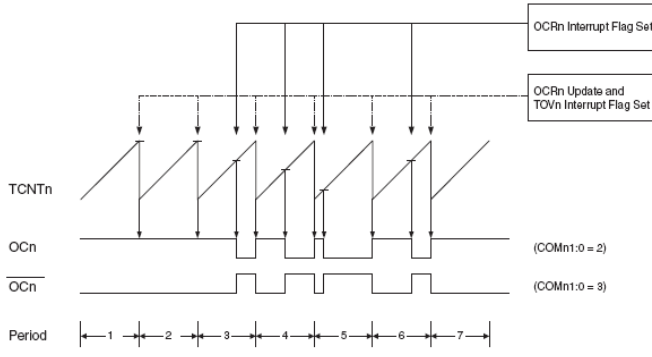


Рис. 8.15. Временные диаграммы таймера/счетчика 2 в режиме Fast PWM

В зависимости от состояния битов COM21:0 регистра TCCR2 (табл. 8.12) на выводе OC2 микросхемы формируется неинвертированный или инвертированный ШИМ-сигнал.

Таблица 8.12

Конфигурирование выхода OC2 в режиме Fast PWM

COM21	COM20	Описание
0	0	Таймер/счетчик1 отключен от вывода OC2
0	1	Резерв
1	0	Очистка OC2 при сравнении и установка при сбросе (неинвертирующий сигнал)
1	1	Установка OC2 при сравнении и очистка при сбросе (инвертирующий сигнал)

Режим ШИМ с коррекцией фазы (Phase Correct PWM Mode)

Режим *Phase Correct PWM* включается приWGM21:0 = 1. В этом режиме счетчик считает от 0 до максимального значения MAX = 255, а далее в обратном порядке от MAX до 0 (рис. 8.16).

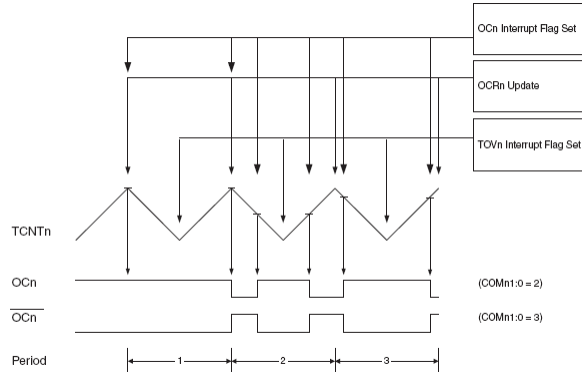


Рис. 8.16. Временные диаграммы таймера/счетчика 2 в режиме Phase Correct PWM

В зависимости от состояния битов COM21:0 регистра TCCR2 (табл. 8.13) на выводе OC2 микросхемы формируется неинвертированный или инвертированный ШИМ-сигнал.

Таблица 8.13

Конфигурирование выхода OC2 в режиме PWM, Phase Correct

COM21	COM20	Описание
0	0	Таймер/счетчик1 отключен от вывода OC2
0	1	Резерв
1	0	Очистка OC2 при счете на сложение и установка OC2 при счете на вычитание
1	1	Установка OC2 при счете на сложение и очистка OC2 при счете на вычитание

Контрольные вопросы

- Чем таймер отличается от счетчика?
- В каких случаях таймер/счетчик формирует запросы на прерывания?
- В каком порядке записываются в 16-битные таймеры/счетчики?
- В каком порядке считываются 16~битные таймеры/счетчики в 8-битные регистры?
- Для чего в структуре таймера/счетчика предусмотрен предделитель?
- Когда устанавливаются биты TOVx в регистре флагов таймера/счетчика?
- Когда сбрасываются биты TOVx в регистре флагов таймера/счетчика?
- К каким источникам сигналов можно подключать вход таймера/счетчика?
- В какой момент происходит захват содержимого таймера/счетчика?
- Что делает таймер/счетчик в режиме захвата?
- Что делает таймер/счетчик в нормальном режиме?
- Что делает таймер/счетчик в режиме Fast PWM?
- Чем режим CTC отличается от нормального режима?
- Чем режим Phase Correct PWM отличается от режима Fast PWM?
- Для чего предназначен регистр TIMSK?
- Какая информация хранится в регистре TIFR?
- Как связаны выводы OCx микросхемы с работой таймеров/счетчиков?
- От чего зависят сигналы на выводах OCx микроконтроллера?

9. АНАЛОГО-ЦИФРОВОЙ ПРЕОБРАЗОВАТЕЛЬ

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.

Схема

Микроконтроллер ATmega16 оснащен 10-разрядным ADC. Вход ADC через мультиплексоры и дифференциальный усилитель соединен с внешними выводами микросхемы ADC0…ADC7 и AGND*(Analog Ground)* и с источником эталонного напряжения 1,22 В (рис. 9.1).

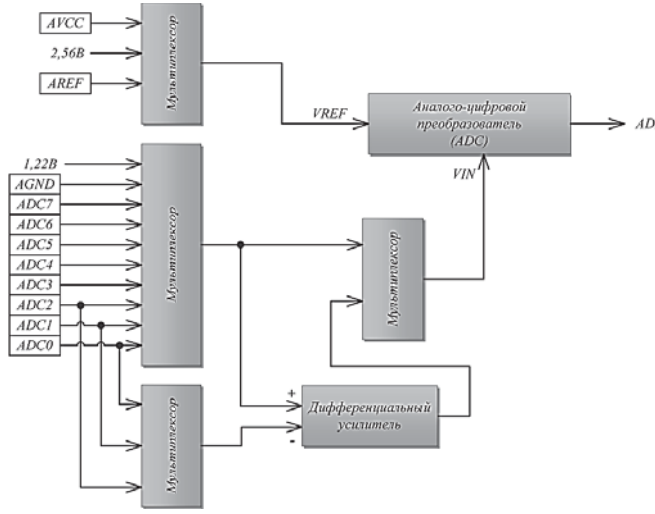


Рис. 9.1. Структура аналого-цифрового преобразователя

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.

Входной сигнал на аналого-цифровой преобразователь поступает через мультиплексоры и дифференциальный усилитель с контактов ADC0 – ADC7, AGND или от внутреннего источника опорного напряжения 1,22 В (рис. 9.1).

Через мультиплексоры и дифференциальный усилитель ко входу преобразователя можно подключить либо один из перечисленных контактов, либо дифференциальный сигнал, представляющий собой разность двух напряжений, поступающих на контакты.

Для одиночного входа результат в виде 10-битного двоичного числа D равен:

$$D = \frac{V_{IN}}{V_{REF}} \cdot 2^{10},$$

где $0 \le V_{IN} < V_{REF}$ – входное напряжение,

V_{REF} – опорное напряжение преобразователя.

Минимальный результат равен 0x000, при $V_{IN} = 0$, максимальный – $2^{10} - 1 = 0x3FF$.

В качестве источника опорного напряжения преобразователя можно использовать внешний сигнал с вывода AREF, внутренний источник 2,56 В, 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.

Для дифференциального входного сигнала

$$D = \frac{(V_{POS} - V_{NEG}) \cdot K}{V_{REF}} \cdot 2^{9},$$

где V_{POS} - напряжение на положительном входе,

V_{NEG} - напряжение на отрицательном входе,

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

$$D = 1,22 * 1024 / 2,56 = 488 = 0x1E8 = 0b111101000.$$

Регистры

Для управления аналого-цифровым преобразователем в микроконтроллере используются регистры (рис. 9.2):

- регистр управления аналого-цифровым преобразователем ADCSR (*ADC Control and Status Register);*
- регистр управления мультиплексором ADMUX (*ADC Multiplexer Selection Register);*
- регистры данных ADCL и ADCH (*ADC Low и ADC High);*

- регистр специальных функций ввода/вывода SFIOR *(Special Function IO Register).*

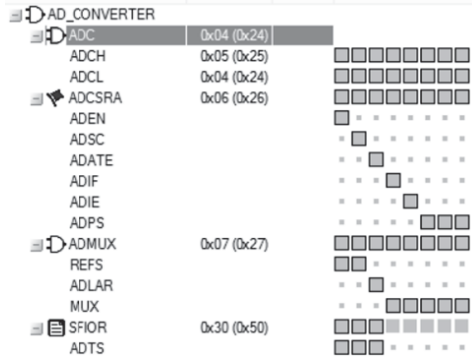


Рис. 9.2. Регистры аналого-цифрового преобразователя

Регистр управления мультиплексорами ADMUX (ADC Multiplexer Selection Register)

Регистр ADMUX (*ADC Multiplexer Selection Register*) предназначен для управления аналоговым и мультиплексорами в составе ADC.

- Биты 7 и 6 - REFS1..0 (*Reference Selection Bits*) – обеспечивают выбор эталонного напряжения на входе AREF аналого-цифрового преобразователя. Выбор производится в соответствии с табл. 9.1. Внутренние источники напряжения не могут быть использованы, если к контакту AREF приложено внешнее напряжение.

Таблица 9.1

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 формирования сигнала, коммутируемого на вход преобразователя. Выбор осуществляется в соответствии с таблицей 9.2.

Таблица 9.2

Выбор входного сигнала ADC

MUX4:0	Одиночный вход	Дифференциальный вход «POS»	Дифференциальный вход«NEG»	Коэффициент усиления
00000	ADC0			
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
111100		ADC4	ADC2	1x
111101		ADC5	ADC2	1x
11110	1,22В			
11111	0 (GND)			

Регистр управления и состояния ADCSRA (ADC Control and Status Register)

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

- Бит 7 – ADEN *(ADC Enable)* – разрешает работу аналого-цифрового преобразователя.
- Бит 6 – ADSC*(ADC Start Conversion)* - предназначен для запуска процесса преобразования. Преобразование начинается с установки бита в состояние 1. Если в процессе выполнения преобразования производится смена канала данных, то ADC вначале закончит текущее преобразование, а потом выполнит переход к другому каналу.
- Бит 5 – ADATE (*ADC Auto Trigger Enable*). Бит управления автозапуском ADC. При установке бита включается триггер запуска ADC. ADC начнет преобразование по положительному фронту сигнала выбранного триггера. Источник сигнала запуска выбирается путем установки ADTS (*ADC Trigger Select*) в регистре специальных функций SFIOR.
- Бит 4 – ADIF*(ADC Interrupt Flag).* Флаг запроса на прерывание. Бит устанавливается при завершении преобразования в ADC и обновлении регистров данных. Аналого-цифровой преобразователь имеет свое собственное прерывание ADC (вектор 0x1C). Переход к обработчику прерывания происходит, если установлены бит разрешения прерывания, ADIE бит и I-бит в регистре статуса SREG. Бит ADIF сбрасывается аппаратно при переходе на вектор прерывания.
- Бит 3 – ADIE *(ADC Interrupt Enable)* – бит разрешения прерывания.
- Биты 2:0 – ADPS2:0 *(ADC Prescaler Select Bits)* - выбор коэффициента предварительного деления. Данные биты определяют коэффициент деления тактовой частоты микроконтроллера для получения необходимой тактовой частоты ADC (табл. 9.3).

Таблица 9.3

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
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Регистры данных ADCL и ADCH (ADC Data Register)

Аналого-цифровой преобразователь формирует 10-разрядный результат и по завершении преобразования результирующие данные размещаются в двух регистрах данных ADCH и ADCL (*ADC Low и ADC High).* 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.

Бит ADLAR в регистре ADMUX воздействует на представление результата. Если бит ADLAR сброшен, результат сдвигается в сторону старших разрядов (рис. 9.3 а). При этом можно ограничиваться упрощенным 8-битным преобразованием, считывая только старший регистр ADCH. Если бит ADLAR установлен, результат сдвигается в сторону младших разрядов (рис. 9.3 б).

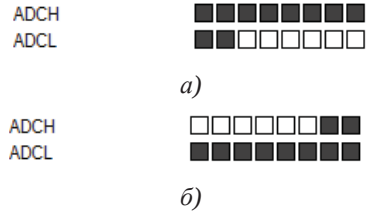


Рис. 9.3. Представление результата в регистре данных

Регистр специальных функций ввода/вывода SFIOR (Special Function IO Register)

Биты 7:5 – ADTS2:0 (*ADC Auto Trigger Source*) – определяют источник инициирующий запуск ADC при установленных битах ADEN и ADATE в регистре ADCSRA. Преобразование будет запущено по фронту выбранного флага прерывания (табл. 9.4).

Таблица 9.4

Выбор источника запуска ADC

ADTS2	ADTS1	ADTS0	Источник запуска
0	0	0	Свободный запуск
0	0	1	Аналоговый компаратор
0	1	0	Внешнее прерывание 0
0	1	1	Таймер/счетчик 0 сравнение
1	0	0	Таймер/счетчик 0 переполнение
1	0	1	Таймер/счетчик 1 сравнение B
1	1	0	Таймер/счетчик 1 переполнение
1	1	1	Таймер/счетчик 1 захват

Контрольные вопросы

- Для чего аналого-цифровой преобразователь снабжается мультиплексором?
- Когда аналого-цифровой преобразователь формирует запрос на прерывание?
- Когда сбрасывается флаг запроса на прерывание аналого-цифрового преобразователя?
- От чего зависит результат аналого-цифрового преобразования?
- Чему будет равен результат 8-битного аналого-цифрового преобразования, если напряжение на входе равно 1 В, а опорное напряжение 2,56 В?
- Чему будет равен результат 10-битного аналого-цифрового преобразования, если напряжение на входе равно 1,5 В, а опорное напряжение 2,56 В?
- Чему будет равен результат 8-битного преобразования, если вход АЦП подключен к источнику 1,22 В, а опорное напряжение выбрано равным 2,56 В?
- На какой контакт микроконтроллера подается опорное напряжение для АЦП?
- Какой бит устанавливается в регистрах аналого-цифрового преобразователя по окончании процесса преобразования?
- Какой бит используется в регистрах АЦП для запуска процесса преобразования?
- На какой контакт микроконтроллера необходимо подключить конденсатор при использовании АЦП с внутренним источником опорного напряжения 2,56 В?

- Какой бит необходимо установить в регистрах АЦП для разрешения процессорному ядру обрабатывать поступающие от АЦП запросы на прерывания?
- Какой бит необходимо установить в регистрах АЦП для разрешения его работы?

10. АНАЛОГОВЫЙ КОМПАРАТОР

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

Схема аналогового компаратора микроконтроллера *ATmega16* приведена на рисунке 10.1.

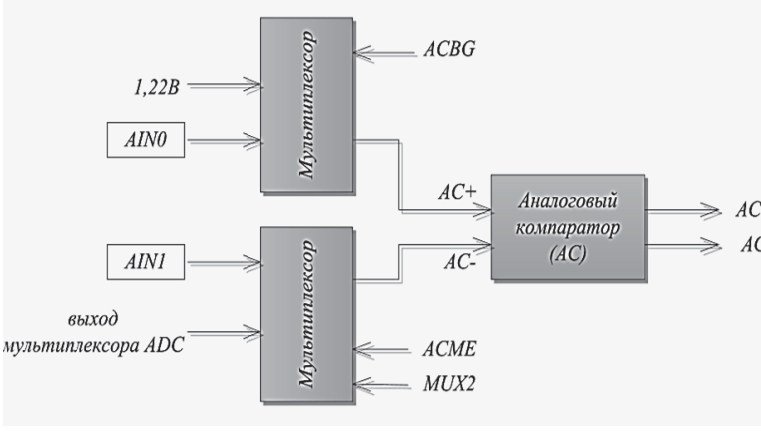


Рис.10.1. Структурная схема аналогового компаратора

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

ACI, при этом пользователь должен задать формирование запроса по нарастающему или падающему фронту или по переключению.

Регистры

В работе компаратора используются регистры (рис. 10.2):

- Регистр управления аналоговым компаратором ACSR (*Analog Comparator Control And Status Register*);
- Регистр специальных функций ввода/вывода SFIOR (*Special Function Input Output Register*.

Рис. 10.2. Регистры аналогового компаратора

Регистр управления ACSR
(Analog Comparator Control And Status Register)

Регистр ACSR 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 компара-

тору 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 0x20. 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/счетчика 1 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/счетчика 1. При сбросе бита ACIC соединения нет. Для запуска прерывания по захвату входа таймера/счетчика 1 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 10.1.

Таблица 10.1

Установки битов ACIS1/ACIS0

ACIS1	ACIS0	Режим прерывания
0	0	Прерывание по переключению выхода компаратора
0	1	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 коммутировать любой из входов порта PORTA (PA7 .. PA0). Для выбора входа используется мультиплексор аналого-цифрового преобразователя.

Регистр специальных функций ввода/вывода SFIOR
(Special Function Input Output Register)

- Бит ACME *(Analog Comparator Multiplexer Enable)* в регистре специальных функций ввода/вывода SFIOR предназначен для подключения ADCмультиплексора 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. 10.2). Если бит ACME равен 0 или бит ADEN равен 1, то к отрицательному входу компаратора подключается контакт AIN1.

Таблица 10.2

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

Контрольные вопросы

- В каком виде представляется результат преобразования сигналов в аналоговом компараторе?
- Для каких задач используется аналоговый компаратор?

- Какой бит устанавливается в регистрах аналогового компаратора при равенстве входных напряжений?
- В каких случаях аналоговый компаратор формирует запрос на прерывание?
- Когда сбрасывается флаг запроса на прерывание аналогового компаратора?
- Можно ли в контроллере одновременно использовать аналого-цифровой преобразователь и аналоговый компаратор?
- Какой бит необходимо установить в регистрах аналогового компаратора для разрешения процессорному ядру обрабатывать поступающие от него запросы на прерывания?

11. 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, которые 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.

Регистры

Для работы с EEPROM в структуре микроконтроллера всегда предусматривается несколько регистров, хранящих адрес ячейки памяти, данные, инструкции и флаги состояния памяти. В процессе работы энергонезависимой памяти микроконтроллера ATmega16 используются регистры (рис. 11.1):

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

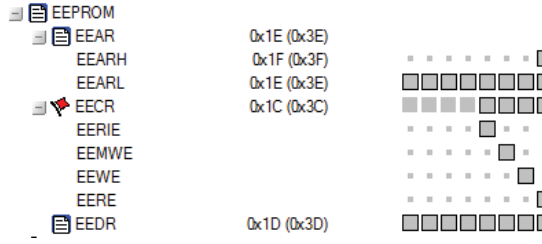


Рис. 11.1. РегистрыEEPROM

Регистры адреса EEPROM (EEPROM Address Registers)

Регистры адреса 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.

Регистр данных EEDR (EEPROM Data Register)

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.

Регистр управления EECR (EEPROM Control Register)

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 0x1E.
- 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:
  - 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 инструкции, останавливается на два тактовых цикла.

- Бит 0 – EERE (*EEPROM Read Enable*) –разрешение 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 процессорное ядро останавливается на четыре тактовых цикла, прежде чем начнет выполнение следующей инструкции. 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.

Контрольные вопросы

- Для чего в структуре микроконтроллера предусмотрена память типа EEPROM?
- Можно ли вместо EEPROM для хранения данных использовать flash-память программ?
- Какие регистры используются для управления EEPROM?

- Почему для работы с памятью типа EEPROM необходима система прерываний?
- Когда устанавливается флаг запроса прерывания EEPROM?
- Когда сбрасывается флаг запроса на прерывания EEPROM?
- Почему для управления записью в EEPROM в регистрах управления предусмотрено 2 бита?
- Какие биты используются для управления записью в EEPROM?

12. ПОСЛЕДОВАТЕЛЬНЫЙ ВВОД-ВЫВОД

12.1. Интерфейс USART *(Universal Synchronous and Asynchronous serial Receiver and Transmitter)*

Универсальный синхронно-асинхронный последовательный приемопередатчик USART *(Universal Synchronous and Asynchronous serial Receiver and Transmitter)* обеспечивает двунаправленный режим обмена информацией по двум или по трем линиям связи. В обмене всегда участвуют два устройства, одно из которых является передатчиком, второе – приемником.

В приемопередатчике предусмотрено два сигнальных контакта:

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

и один контакт XCK (*Transfer Clock*) для передачи сигнала синхронизации.

При этом два устройства-приемопередатчика должны быть соединены между собой двумя или тремя линиями (не считая общего провода GND) (рис. 12.1). Линия связи XCK используется только в синхронном режиме.

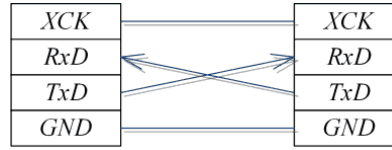


Рис. 12.1. Соединение устройств по интерфейсу USART

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. 12.2.

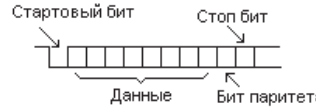


Рис. 12.2. Формат асинхронной передачи

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 единицы, 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. При синхронной передаче скорость задается сигналом синхронизации, формируемым на линии XCK.

Регистры USART

Управление USART осуществляется через регистры ввода/вывода. В контроллере ATmega16 для управления используется 5 регистров (рис. 12.3):

- регистры UBRRH1 и UBRR*(UART Baud Rate registers)* – регистры скорости передачи;
- регистры UCSRA *(UART Control and Status Register A)* –регистр A управления и статуса UART;
- регистр UCSRB *(UART Control and Status Register B)* – регистр B управления и статуса UART;
- регистр UCSRC *(UART Control and Status Register C)* – регистр C управления и статуса UART;
- регистр UDR *(UART Data Register)* – регистр данных UART.

Регистры данных UDR (USART I/O Data Register)

Буферные регистры приемника и передатчика располагаются по одному адресу пространства ввода/вывода и обозначаются как регистр данных UDR (*Universal Data Register*). В этом регистре хранятся младшие 8 разрядов принимаемых и передаваемых данных. При чтении выполняется обращение к буферному регистру приемника, при записи — к буферному регистру передатчика. В модулях USART буфер приемника является двухуровневым (FIFO буфер), изменение состояния которого происходит при любом обращении к регистру UDR.

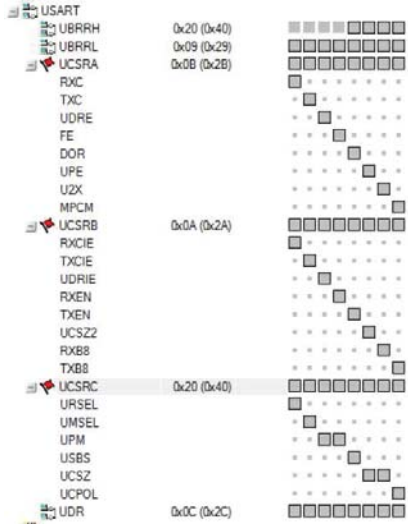


Рис.12.3. РегистрыUSART

Регистр управления и статуса UCSRA
(USART Control and Status Register A)

Разряды регистра предназначены для управления UART:

- Бит 7 RXC (*USART Receive Complete*) – флаг завершения приема; устанавливается при наличии непрочитанных данных в буфере приемника (регистр данных UDR). Сбрасывается флаг аппаратно после опустошения буфера (в режиме асинхронной передачи UART — после прочтения регистра данных). Если бит RXCIE регистра UCSRB установлен, то при установке флага генерируется запрос на прерывание USART, RXC (*USART, Rx Complete*) с вектором 0x15.

- Бит 6 TXC (*USART Transmit Complete*) – флаг завершения передачи; устанавливается после передачи всех разрядов посылки из сдвигового регистра передатчика, при условии, что в регистр данных UDR не было загружено новое значение. Если бит TXCIE регистра UCSRB установлен, то при установке флага генерируется прерывание USART, TXC (*USART, Tx Complete*) с вектором 0x1A. Флаг сбрасывается аппаратно при выполнении подпрограммы обработки прерывания или программно, записью в него логической единицы.

- Бит 5 UDRE *(USART Data Register Empty)* - флаг опустошения регистра данных; устанавливается при пустом буфере передатчика (после пересылки байта из регистра данных UDR в сдвиговый регистр передатчика). Установленный флаг обозначает, что в регистр данных можно загружать новое значение. Если бит UDRIE регистра UCSRB установлен, генерируется запрос на прерывание USART, UDRE (*USART Data Register Empty*) с вектором 0x18. Флаг сбрасывается аппаратно, при записи в регистр данных.

- Бит 4 FE (*Frame Error*) - флаг ошибки кадрирования; устанавливается, если первый стоп бит принятой посылки равен 0. Флаг сбрасывается при приеме стоп бита, равного 1.

- Бит 3 DOR (*Data Over Run*) - флаг переполнения; устанавливается, если буфер приемника полон и в момент обнаружения нового стартового бита в сдвиговом регистре приемника находится последнее принятое слово. В режиме асинхронной передачи UART флаг устанавливается, если новый кадр будет помещен в сдвиговый регистр приемника до того, как из регистра данных будет считано предыдущее слово. Флаг сбрасывается при пересылке принятых данных из сдвигового регистра приемника в буфер.

- Бит 2 PE*(Parity Error*) - флаг ошибки контроля четности; устанавливается, если в буфере приемника выявлена ошибка контроля четности. При отключенном контроле четности (бит UPM1 в регистре UCSRC установлен) этот бит постоянно сброшен.

- Бит 1 U2X (*Double the USART Transmission Speed*) – удвоение скорости обмена. Если этот бит установлен, коэффициент деления предделителя контроллера скорости передачи уменьшается с 16 до 8, удваивая тем самым скорость асинхронного обмена по последовательному каналу.

Используется только при асинхронном режиме работы, в синхронном режиме бит должен быть сброшен.

- Бит 0 MPCM (*Multi-processor Communication Mode*) – режим мультипроцессорного обмена. Если бит установлен, ведомый микроконтроллер ожидает приема кадра, содержащего адрес. Кадры, не содержащие адреса устройства, игнорируются.

Регистр управления и статуса UCSRB
(USART Control and Status Register B)

Биты регистра:

- 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 USART, RXC 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 (*USART Data Register Empty Interrupt Enable*) - разрешение прерывания при опустошении регистра данных. При установленном 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 приема. Установленный бит RXEN разрешает прием данных в USART. Если прием 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 передачи. Установленный бит TXEN разрешает передачу данных USART. Если разряд сбрасывается во время передачи, выключение передатчика произойдет только после завершения передачи данных, находящихся в сдвиговом регистре и буфере передатчика.

- Бит 2 UCSZ2 (*USART Character Size*) – формат посылок; используется совместно с разрядами UCSZ1:0 регистра UCSRC для задания размера слов данных, передаваемых по последовательному каналу.

- Бит 1 RXB8 (*Receive Data Bit 8)* - прием 8-разрядных данных. При использовании 9-разрядных слов данных этот разряд содержит значение старшего разряда принятого слова. Содержимое этого разряда должно быть считано до прочтения регистра данных UDR.

- Бит 0 TXB8 (*Transmit Data Bit 8)* - передача 8-разрядных данных. При использовании 9-разрядных слов данных содержимое этого разряда является старшим разрядом передаваемого слова. Требуемое значение должно быть занесено в этот разряд до загрузки байта данных в регистр UDR.

Регистр управления и статуса UCSRC
(USART Control and Status Register C)

Регистр UCSRC имеет в пространстве регистров ввода/вывода такой же адрес (0x20), как и регистр UBRRH. Биты регистра:

- Бит 7 URSEL (*USART Register Select*) – выбор регистра. Этот бит определяет, в какой из регистров модуля производится запись. Если бит установлен, обращение производится к регистру UCSRC; если бит сброшен, обращение производится к регистру UBRRH.

- Бит 6 UMSEL (*USART Mode Select*) – режим работы USART. Если бит сброшен, модуль USART работает в асинхронном режиме (режим UART); если бит установлен, то модуль USART работает в синхронном режиме.

- Биты 5:4 UPM1:0 (*Parity Mode*) - режим работы схемы контроля и формирования четности. Режим работы задается в соответствии с таблицей 12.1.

Таблица 12.1

Режимы работы схемы контроля четности

UPM1	UPM0	Режим работы
0	0	Выключено
0	1	Резерв
1	0	Проверка на четность
1	1	Проверка на нечетность

- Бит 3 USBS (*USART Stop Bit Select*) – количество стоп-битов, посылаемых передатчиком. Если разряд сброшен, передатчик посылает 1 стоп-бит, если установлен, то 2 стоп бита. Для приемника содержимое этого разряда безразлично.

- Биты 2:1 UCSZ1:0 (*USART Character Size*) – формат посылок. Совместно с разрядом UCSZ2 регистра UCSRB эти разряды определяют количество разрядов данных в посылках (табл. 12.2):

Таблица 12.2

Формат посылки

UCSZ2	UCSZ1	UCSZ0	Количество бит
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	0	0	резерв
1	0	1	резерв
1	1	0	резерв
1	1	1	9

- Бит 0 UCPOL (*USART Clock Polarity*) - полярность тактового сигнала. Значение этого разряда определяет момент выдачи и считывания данных на выводах модуля. Разряд используется только при работе в синхронном режиме. При работе в асинхронном режиме он должен быть сброшен.

Таблица 12.3

Полярность тактового сигнала

UCPOL	Вывод на TxD	Ввод с RxD
0	Нарастающий фронт XCK	Спадающий фронт XCK
1	Спадающий фронт XCK	Нарастающий фронт XCK

Регистры скорости передачи данных UBRRL and UBRRH (USART Baud Rate Registers)

В асинхронном режиме, а также в синхронном режиме при работе в качестве ведущего, скорость приема и передачи данных задается контроллером скорости передачи, функционирующим как делитель системного тактового сигнала с программируемым коэффициентом деления. Коэффициент определяется содержимым регистра контроллера UBRR. В блок приемника сформированный сигнал поступает сразу, а в блок передатчика — через дополнительный делитель, коэффициент деления которого (2, 8 или 16) зависит от

режима работы модуля USART. Регистр UBRR является 12-разрядным и физически размещается в двух регистрах ввода/вывода UBRRH и UBRRL.

При работе в асинхронном режиме скорость обмена определяется не только содержимым регистра UBRR, но и состоянием разряда U2X регистра UCSRA . Если этот разряд установлен, коэффициент деления предделителя уменьшается в два раза, а скорость обмена соответственно удваивается. При работе в синхронном режиме этот разряд должен быть сброшен.

Скорость обмена BAUD в бодах определяется тактовой частотой микроконтроллера f_{CK} и содержимым регистра UBRR по следующим формулам:

- асинхронный режим (обычный, U2X = 0)

$$BAUD = \frac{f_{CK}}{16(UBRR+1)};$$

- асинхронный режим (ускоренный, U2X = 1)

$$BAUD = \frac{f_{CK}}{8(UBRR+1)};$$

- синхронный режим ведущего

$$BAUD = \frac{f_{CK}}{2(UBRR+1)}.$$

12.2. Интерфейс SPI

Последовательный периферийный интерфейс SPI (*Serial Peripheral Interface*) обеспечивает полный дуплексный обмен данными между двумя контроллерами. При этом один контроллер считается ведущим (*master*), второй - ведомым (*slave*). Ведущий контроллер является источником сигнала синхронизации (SCK) (рис. 12.4).

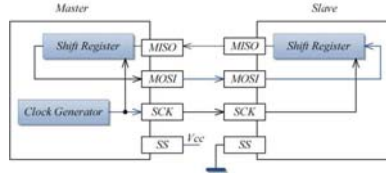


Рис. 12.4. Соединение устройств с интерфейсом SPI

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 (*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.

Регистры SPI

Управление SPI осуществляется через регистры ввода/вывода. В контроллере ATmega16 такими регистрами являются (рис. 12.5):

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

Регистр данных SPDR (SPI Data Register)

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

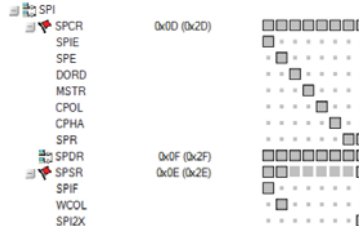


Рис. 12.5. Регистры SPI

Регистр управления SPCR(SPI Control Register)

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. В противном случае, при отсутствии передачи SCK = 0.

- 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 12.4. Для ведомого контроллера эти биты не существенны.

Таблица 12.4

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

Регистр SPSR статуса (SPI Status Register)

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 табл. 12.4.

12.3. Интерфейс I2C

2-проводной последовательный интерфейс (*2-Wire Serial Interface*) или I^2C (*Inter-Integrated Circuit*) 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.

Типовая 2-проводная последовательная конфигурация шины I2C показана на рисунке 12.6.

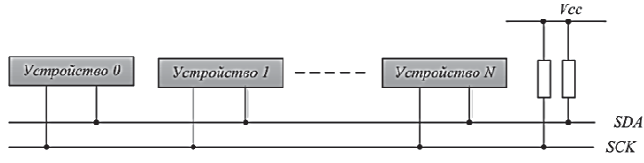


Рис. 12.6. Соединение устройств с интерфейсом I^2C

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 на рисунке 12.7.

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*).

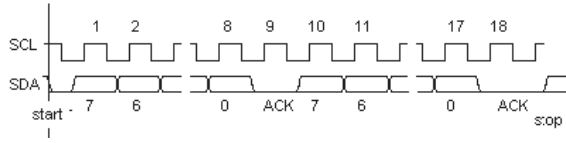


Рис. 12.7. Протокол передачи данных интерфейса I^2C

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.

Регистры TWI

Для управление TWSI используются регистры (рисунок 12.8):

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

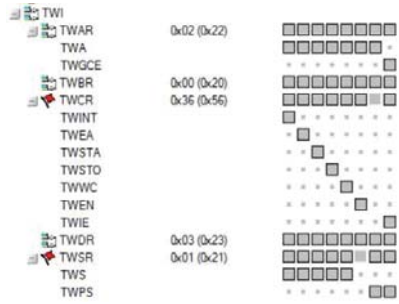


Рис. 12.8. Регистры TWI

Регистр адреса TWAR (2-wire Serial Interface (Slave) Address Register)

Регистр – TWAR 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;
- Биты 1:7 – TWA (*2-wireSerialInterfaceAddress)* – биты задания адреса.

Регистр задания скорости TWBR (*2-wire Serial Interface Bit Rate Register)*

Регистр TWBR предназначен для задания скорости передачи данных. Скорость передачи *Bit Rate* (бит/с) связана с содержимым регистра TWBR соотношением:

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

где f_{ck} - тактовая частота ядра микроконтроллера (с учетом коэффициента деления предделителя, задаваемого в регистре TWSR).

Регистр TWCR (2-wire Serial Interface Control Register)

Регистр TWCR предназначен для управления передачей данных.

- Бит 7 – TWINT (*2-wire Serial Interface Interrupt Flag*) - флаг прерывания 2-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 0x022. 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 выводов 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 данных TWDR (2-Wire Serial Interface Data Register)

Регистр данных TWDR хранит байт данных для передачи или последний байт данных, полученный на 2-проводной последовательной шине.

Регистр TWSR (2-wire Serial Interface Status Register)

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 течение одного тактового цикла после прерывания от интерфейса. Специальные таблицы позволяют определить состояние интерфейса при различных кодах состояния.
- Бит 2 – резервный бит.
- Биты 1:0 – TWPS (*TWI Prescaler Bits*). Биты предделителя для задания скорости передачи данных (таблица 12.5).

Таблица 12.5

Предделитель TWI

TWPS1	TWPS0	Коэффициент предделителя
0	0	1
0	0	4
0	1	16
0	1	64

Контрольные вопросы

- Сколько линий связи использует интерфейс USART/SPI/I2C?
- С какой скоростью передаются данные в интерфейсе USART/SPI/I2C?
- Как регулируется скорость передачи данных в интерфейсе USART/SPI/I2C?
- Какие регистры используются в интерфейсе USART/SPI/I2C?
- Какие прерывания используются в интерфейсе USART/SPI/I2C?
- Сколько устройств можно объединить по интерфейсу USART/SPI/I2C?

13. УПРАВЛЕНИЕ ПИТАНИЕМ И 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 ATmega16 работоспособен при напряжении питания от 4 до 5,5 В, а его модификация ATmega16L – от 2,7 до 5,5 В. 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 сна (*Sleep Mode*). Режимы, в которых отключаются те или иные функциональные блоки микроконтроллера и его энергопотребление существенно снижается.

Микроконтроллеры Atmel AVR обеспечивают различные режимы сна, позволяющие пользователю снизить потребление энергии.

Потребление энергии в микроконтроллере зависит от включенных внутренних тактовых генераторов и функциональных блоков.

13.1. Режим Idle

В режиме Idle останавливается центральный процессор, но остаются в работе интерфейсы SPI, USART, TWI, аналоговый компаратор, ADC, таймеры/счетчики, сторожевой таймер и система прерываний.

Пробуждение в режиме IDLE возможно от внешних прерываний и таких внутренних прерываний, как переполнение таймера и завершение приема UART. При активизации микроконтроллера из *Idle* режима программа начинает выполняться незамедлительно.

13.2. Режим ADC Noise Reduction

Режим ADC Noise Reduction останавливает центральный процессор, но позволяет работать аналого-цифровому преобразователю, внешним прерываниям, интерфейсу TWI, таймеру/счетчику 2 и сторожевому таймеру.

Если в режим в аналого-цифровом преобразователе включен режим ADC Noise Reduction ADC, то преобразование в ADC начинается автоматически, а пробуждение микроконтроллера возможно при сбросе (External Reset, Watchdog Reset, Brown-out Reset), по прерываниям от интерфейса TWI, таймера/счетчика 2,EEPROM, по внешним прерываниям от INT2 или INT0, INT1.

13.3. Режим Power-down

В этом режиме 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.

13.4. Режим Power-save

Этот режим аналогичен режиму *Power Down*, но если таймер/счетчик 2 тактируется асинхронно, т.е. бит AS0 в регистре ASSR установлен, то в этом режиме таймер/счетчик 2 будет работать, и микроконтроллер будет активироваться прерываниями по переполнению или совпадению с выхода таймера/счетчика 0.

13.5. Режим Standby

Этот режим идентичен режиму Power-down за исключением того, что генератор продолжает работать. Из режима Standby устройство просыпается за шесть тактов.

13.6. Extended Standby Mode

Этот режим также идентичен режиму Power-save за исключением того, что генератор остается в работе. Из режима Extended Standby устройство просыпается за шесть тактов.

13.7. Регистр управления MCUCR (*MCU Control Register*)

Для перевода микроконтроллера в тот или иной режим сна 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. 13.1).

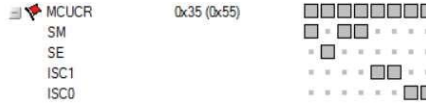


Рис. 13.1. Биты управления режимами сна микроконтроллера ATmega16

- Биты 7, 5, 4 – SM2:0 (*Sleep Mode Select Bits 2, 1, and 0*) позволяют задать один из шести режимов сна в соответствии с таблицей 13.1.

Таблица 13.1

Режимы сна

SM2	SM1	SM0	Режим
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	1	0	Standby
1	1	1	Extended Standby

- Бит 6 – SE (*Sleep Enable*). При установленном бите режим сна запускается при выполнении инструкции SLEEP.

Контрольные вопросы

- Какие напряжения используются для питания микроконтроллеров?
- Какие напряжения питания микроконтроллеров используются в автономных изделиях?
- От чего зависит ток, потребляемый микроконтроллером?
- Чем отличаются друг от друга различные режимы сна?
- Как задается режим сна микроконтроллера?
- Какая команда используется для перевода микроконтроллера в режим сна?
- Какие устройства могут выводить микроконтроллер из режима сна?
- Какие регистры используются для управления режимами сна микроконтроллера?

14. ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ

14.1. 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 = 0xFF), готовыми к программированию. Программирование кристалла 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*).

- Само программирование может осуществляться самим микроконтроллером под управлением программы, записанной в Boot Program Section флэш-памяти программ.

Параллельное программирование

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 рисунке 14.1 показана схема подключения микроконтроллера ATmega16 при параллельном программировании.

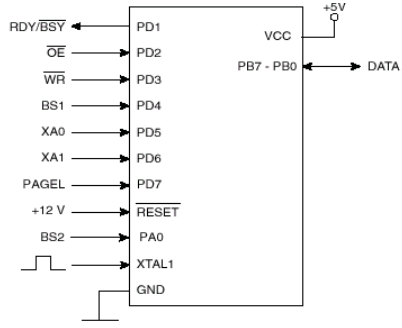


Рис. 14.1. Параллельное программирование микроконтроллера ATmega16

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 описывает операцию, выполняемую в момент прихода синхроимпульса. В зависимости от сочетания сигналов информация на линиях порта B микроконтроллера может быть воспринята как команда управления процессом записи, как данные или как один из байтов адреса ячейки памяти.

В таблице 14.1 приведены инструкции параллельного программирования микроконтроллера ATmega16, различаемые микроконтроллером.

Таблица 14.1

Инструкции параллельного программирования микроконтроллера

Код инструкции	Команда
0x80	Очистка кристалла(*Chip Erase*)
0x40	Запись *Fuse* битов
0x20	Запись *Lock* битов
0x10	Запись *Flash* памяти
0x11	Запись EEPROM
0x08	Чтение байта *Signature*
0x04	Чтение *Fuse* и *Lock* битов
0x02	Чтение *Flash* памяти
0x03	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 ATmega16 имеется FUSE бит EESAVE, программирование которого приводит к тому, что при выполнении операции Chip Erase содержимое EEPROM сохраняется.

Последовательное программирование

При последовательном программировании (ISP – *In System Programing*) значительно меньше контактов микроконтроллера и не требуется источника высокого напряжения. На рисунке 14.2 показана схема включения микроконтроллера в режиме последовательного программирования.

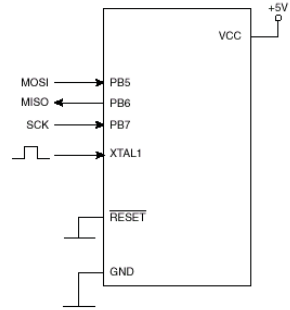


Рис. 14.2. Последовательное программирование микроконтроллера ATmega16

Здесь для программирования используется последовательный SPI интерфейс при подключении контакта Vcc к источнику питания 5 В, 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 ATMEGA16 в последовательном режиме программирования используются четырехбайтные инструкции [14].

14.2. 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 обеспечения. 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 внут-

ренних ресурсов процессора, а на микропроцессорной плате - реакцию системы на те или иные инструкции 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 STK500.

14.3. 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 С 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.

Запуск программы

Для работы над новым проектом создается папка проекта с произвольным именем на диске С.Программа AVRStudio версии 4 и выше функционирует в среде WindowsXP/7/8. Значок запуска программы изображен на рисунок 14.3.

Рис. 14.3. Значок программы AVR Studio

В результате запуска AVRStudio на экране появляется окно программы (рис. 14.4).

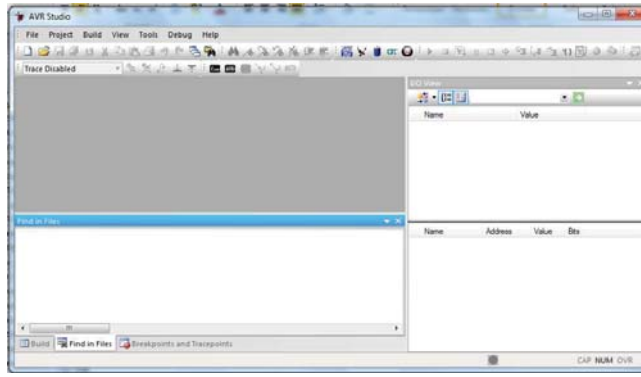


Рис. 14.4. Стартовое окно программы AVR Studio

Для создания нового проекта необходимо в выпадающем меню *Project* выбрать команду *New Project*. В результате на экране появляется диалоговое окно (рис. 14.5), в окне которого необходимо ввести название проекта (*Project name*) и его расположение (*Location*).

Рис. 14.5. Диалоговое окно нового проекта

Далее выбирается тип проекта:

- *Atmel AVR Assembler*–проект на языке ассемблера.
- *AVR GCC*–проект на языке С.

При выборе *AVR Assembler* и нажатии кнопки *Next* на экране появляется окно выбора отладочной платформы (рис. 14.6), позволяющее выбрать отладочную среду и конкретный микроконтроллер для реализации проекта.

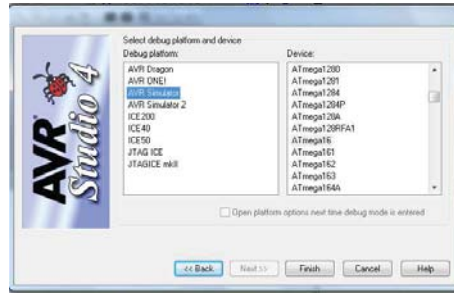


Рис. 14.6. Окно выбора платформы проекта

В открывшееся окно для редактирования программы можно с клавиатуры или через буфер компьютера ввести текст программы на языке ассемблера (рис. 14.7).

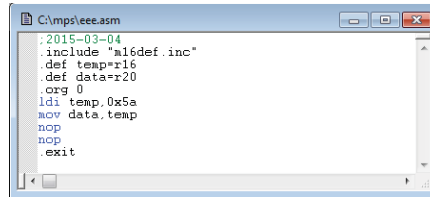


Рис. 14.7. Окно редактирования программы

Далее осуществляется трансляции программы и проверка правильности её написания. Выбирается пункт *Build* в меню *Build***.** Открывшееся окно *Build* содержит сообщения ассемблера. В окне (рисунок 14.8) сообщается о количестве слов программы и данных, о наличии ошибок.

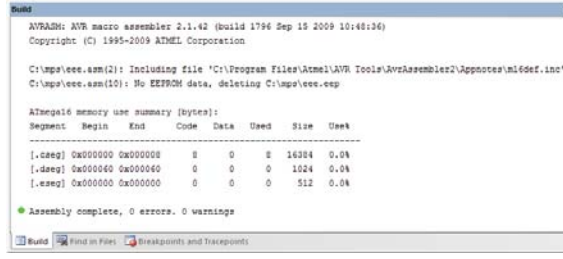


Рис. 14.8. Окно результатов трансляции программы

Для запуска созданной и откомпилированной программы используется команда *Build and Run* в меню *Build*.

Отладка программы

Для отладки программы предназначено меню *Debug*. Пункт меню *AVR Simulator Options* позволяет установить опции симулятора: выбрать контроллер, посмотреть его параметры, задать тактовую частоту ядра, границу *Boot*-секции flash памяти (рис. 14.9).

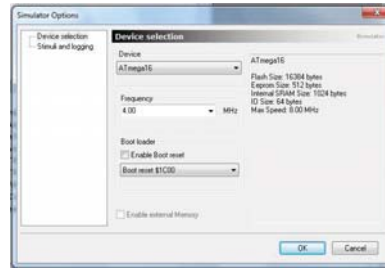


Рис.14.9. Диалоговое окно выбора опций симулятора

После выбора опций симулятора в левом поле окна ассемблерной программы появляется желтая стрелка, указывающая на позицию программного счетчика микроконтроллера (рис.14.10).

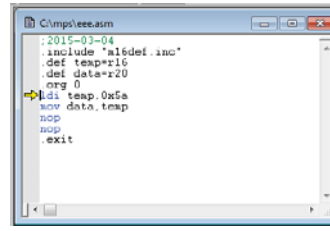


Рис.14.10. Установка программного счетчика в тексте программы

В *AVR Studio* для отладки программы предусмотрен ряд инструкций, отображенных во вкладке меню *Debug* (рисунок 14.11).

Команда *Reset (Shift+F5)* выполняет сброс микроконтроллера. Если программа при этом выполняется, то исполнение будет остановлено. После сброса вся информация во всех окнах модифицируется.

С помощью команд шагового режима в окне *I/O View* можно проследить изменения в регистрах устройств ввода/вывода.

Команда *Step Into (F11)*запускает пошаговое исполнение программы. После исполнения инструкции, вся информация во всех окнах обновляется. Команда *Step Into* не работает в подпрограммах.

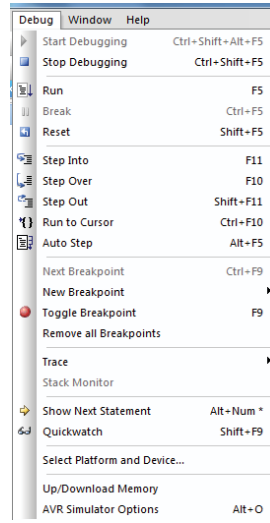


Рис. 14.11. Меню отладки программы

Команда *Step Over (F10)*также выполняет одну инструкцию. Если инструкция содержит вызов подпрограммы, то также выполняется подпрограмма. Если во время шага встречается точка останова, то выполнение программы прерывается. После завершения шага вся информация во всех окнах обновляется.

Команда *Run (F5)* запускает исполнение программы. Программа будет выполняться до остановки пользователем, либо до точки останова (*Breakpoints*). Для установки контрольной точки в программе предусмотрен пункт *Toggle Breakpoint (F9).*Контрольная точка ставится в строке, отмеченной курсором (рисунок 14.12).

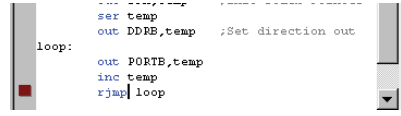


Рис. 14.12. Установка контрольной точки в программе

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

Команда *Break (CTRL-F5)*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 (Ctrl+F10)*. При таком запуске программы она выполняется до достижения инструкции, 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 инструкции вся информация во всех окнах модифицируется. Поскольку команда *Run To Cursor* зависит от позиции курсора, она доступна только в активном окне Source.

Для наблюдения за работой программы можно открыть диалоговое окно устройств ввода/вывода*(I/O View)*(рисунок 14.13).

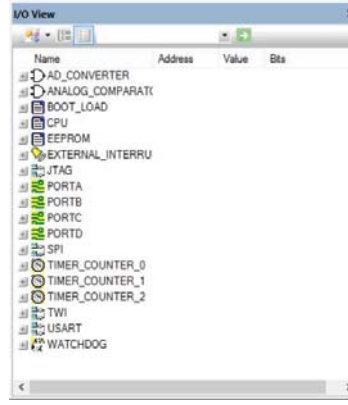


Рис. 14.13. Диалоговое окно устройств ввода/вывода

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

Для наблюдения за изменениями переменных можно использовать окно *Toggle Watch Windows*. 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. 14.14).

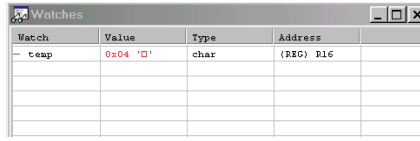


Рис. 14.14. Окно временных переменных

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. 14.15).

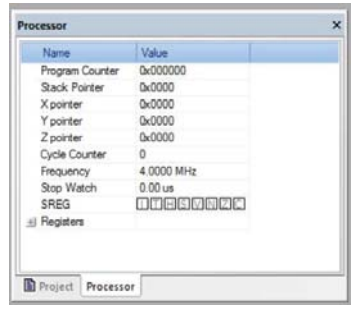


Рис. 14.15. Диалоговое окно для просмотра состояния процессорного ядра

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

Просмотр ячеек памяти программ, памяти данных, EEPROM и регистров портов ввода/вывода в ходе исполнения программы возможно также с помощью диалогового окна Memory. Спадающее меню диалогового окна позволяет выбрать один из четырех массивов ячеек памяти: *Data, IO, EEPROM, ProgramMemory*. Для одновременного просмотра нескольких областей окно *Memory* может быть открыто несколько раз. Информация в диалоговом окне представляется побайтно в шестнадцатеричной системе счисления (рис. 14.16).

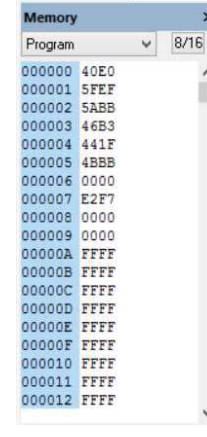


Рис. 14.16. Диалоговое окно Memory

14.4. Стартовый набор STK 500

STK500 – завершенный стартовый набор и система проектирования для AVR флэш-микроконтроллеров корпорации *Atmel*. Набор предназначен для проектировщиков, желающих быстро приступить к разработке программного кода и выполнить тестирование новой разработки, и используется в лабораторном практикуме.

Состав набора

Набор STK500 содержит печатную плату *Development Board*, кабель для последовательного программирования через COM-порт компьютера.

Плата *Development Board* из набора STK500 (рис. 14.17) 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.

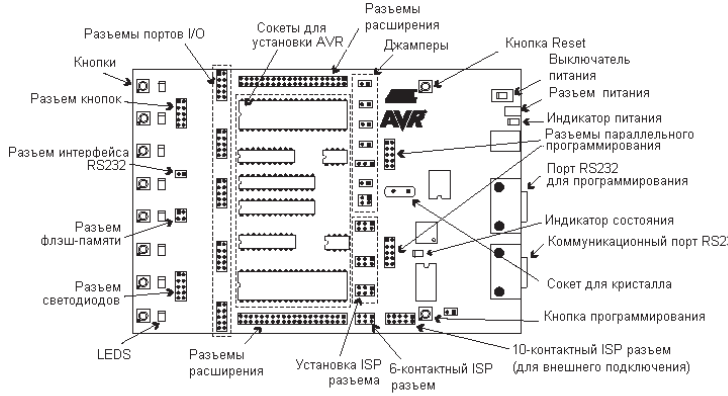


Рис. 14.17. Плата Development Board из набора STK500

Программирование

В качестве программного приложения, используемого для связи с платой STK500, используется интегрированная среда *AVR Studio* версии не ниже 3.2.

Нажатие на кнопку Program AVR в меню Tools на панели инструментов *AVR Studio* при подключенном устройстве STK 500 приводит к запуску пользовательского интерфейса STK500 , как показано на рис. 14.18.

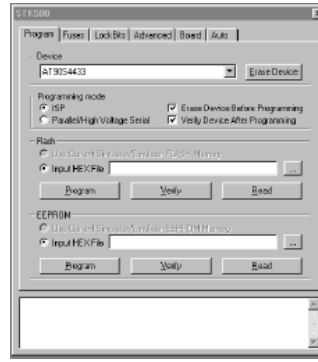


Рис. 14.18. Пользовательский интерфейс STK500

Пользовательский интерфейс STK500 выполняет функции управления платой STK500. Доступные настройки разделены на 6 окон, каждое из которых вызывается щелчком на соответствующей закладке. В зависимости от выбранного типа микроконтроллера определяется доступный набор функций по выбору и установкам. Недоступные функции отделяются серым цветом.

Установки программирования разделены на 5 областей.

В поле программируемого устройства *Device* из раскрывающегося списка необходимо выбрать тип микроконтроллера. Сюда же входит кнопка стирания памяти микроконтроллера *Erase Device.*

В поле *Programming Mode* (рис. 14.18) задается режим программирования. Для устройств, поддерживающих возможность только параллельного программирования, опция ISP-программирования будет отделена серым цветом. Установка флажка *Erase Device Before Programming* активизирует полезную функцию стирания памяти программ перед программированием, а при установке флажка *Verify Device After Programming* STK500 будет выполнять проверку правильности записанной информации.

Если при запуске пользовательского интерфейса STK500 в *AVR Studio*не был загружен какой-либо проект, то опция *Use Current Simulator/Emulator FLASH Memory* в поле программирования флэш-памяти *Flash* будет отделена серым цветом. Если проект был открыт, то данная опция позволит записать во флэш-память содержимое окна *Flash Memory view* из *AVR Studio.*Если запущенного проекта нет или необходимый *hex*-файл хранится отдельно, то после нажатия кнопки *Input HEX File* необходимо указать путь к файлу и его имя.

Если пользовательский интерфейс STK500 открыт без предварительного открытия проекта *AVR Studio*, то опция *Use Current Simulator/Emulator EEPROM Memory* в поле программирования *EEPROM* будет недоступна. Если проект был открыт, то данная опция позволит записать в *EEPROM* содержимое окна *EEPROM Memory view*. Если не было запущено проекта или необходимый hex-файл хранится отдельно, то после нажатия кнопки *Input HEX File* необходимо указать путь к файлу и его имя.

В окне *Fuses* (рис. 14.19) представлены доступные для выбранного типа микроконтроллера конфигурационные биты. Некоторые из них становятся доступными только в режиме параллельного программирования.

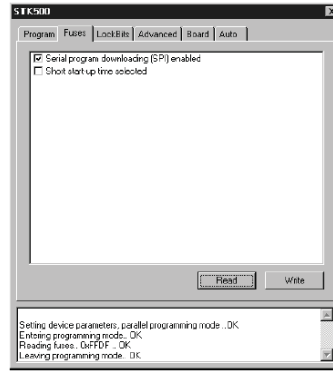


Рис. 14.19. Окно конфигурационных бит

В режиме ISP-программирования данные биты отображаются, но воздействовать на них невозможно. При нажатии кнопки *Read* считываются текущие установки доступных бит из микроконтроллера, а при нажатии *Write* происходит запись текущих установок окна в микроконтроллер. Наличие флажка возле бита конфигурации означает, что данная конфигурация разрешена, а в памяти значение этого бита запрограммировано как 0.

Окно Lock Bits (рис. 14.20) показывает, какие режимы защиты программы доступны при заданном типе микроконтроллера. Все биты защиты доступны как в режиме ISP-программирования, так и в режиме программирования повышенным напряжением. Режим защиты задается комбинацией нескольких бит защиты. Если указан режим защиты, то возможности снижения уровня защиты нет. При возникновении такой потребности необходимо стереть все биты защиты путем стирания памяти *Erase Device* в окне *Program*, что дополнительно вызовет стирание флэш-памяти программ и EEPROM. Если у микроконтроллера запрограммирован конфигурационный бит *EESAVE*, то содержимое EEPROM будет сохраняться даже при выполнении команды стирания памяти.

Рис. 14.20. Окно бит защиты программы

Окно *Advanced* (рис. 14.21) представляет собой два поля для идентификации параметров микроконтроллера, не вошедших в предыдущие окна.

Нажатие на кнопку считывания сигнатуры *Read Signature* приводит к считыванию из микроконтроллера и отображению сигнатурных байт. Сигнатурные байты используются для идентификации микросхемы и ее производителя. После считывания сигнатуры программа проверяет ее на соответствие выбранному типу микроконтроллера. Детальная расшифровка сигнатурных байт приводится в документации на микроконтроллер.

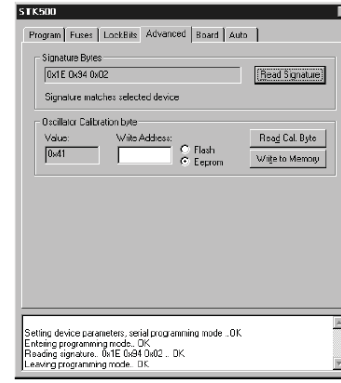


Рис. 14.21. Окно прочих параметров

Калибровочный байт *Oscillator Calibration Byte* записывается в микроконтроллер на стадии производства и, поэтому, доступен только для чтения. Данное значение используется в программе для записи в регистр OSCCAL для подстройки на номинальную частоту встроенного RC-генератора.

Нажатие на кнопку *Read Cal. Byte* приводит к отображению на экране значения калибровочного байта в текстовом поле *Value*.

Окно *Board* (рис. 14.22) позволяет изменить рабочие условия на плате STK500. Для изменения доступны следующие параметры: *VTARGET*, *AREF* и частота генератора.

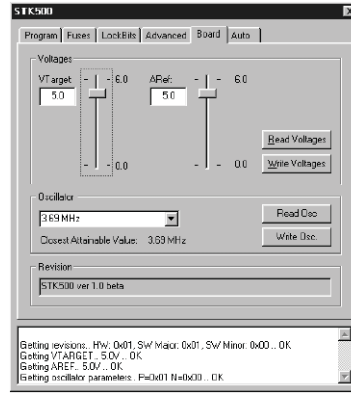


Рис. 14.22. Окно управления платой

VTARGET управляет напряжением питания целевой платы (микроконтроллера). Задать напряжение питания можно в диапазоне 0…6 В с шагом 0,1 В путем перемещения ползунка либо заданием числового значения в текстовом окне. Для считывания установленного на плате напряжения необходимо нажать кнопку *Read Voltages*, а для установки нового значения нажать кнопку *Write Voltages*.

AREF управляет аналоговым опорным напряжением, которое используется встроенным в микроконтроллер аналогово-цифровым преобразователем. Данный параметр необходимо настраивать только для микроконтроллеров, имеющих встроенный ADC. Данное напряжение задается в диапазоне 0..6 В с шагом 0,1 В путем перемещения ползунка или путем введения числового значения в текстовом поле. Значения как VTARGET, так и AREF считываются путем нажатия кнопки Read Voltages, а записываются нажатием кнопки *Write Voltages*.

Невозможно установить напряжение AREF выше напряжения VTARGET, т.к. это может привести к выходу из строя микроконтроллера.

В поле настроек генератора *Oscillator*из раскрывающегося списка можно выбрать фиксированные частоты 32,7 кГц; 1,23 МГц; 1,84 МГц; 3,69 МГц (макс.

частота) или вообще остановить генератор. Чтение/запись частот генератора осуществляется нажатием соответствующих кнопок *Read Osc./Write Osc*.

Окно управления функциями автоматизации *Auto* (рис. 14.23) предназначено для программирования нескольких микроконтроллеров одним и тем же программным кодом. Для разрешения выполнения действия его необходимо пометить флажком. Например, если отмечено только действие *Program FLASH* , то после нажатия кнопки *Start* будет произведена запись во флэш-память в соответствии с настройками в окне *Program*. Все действия используют настройки в пределах пользовательского интерфейса STK500.

Рис. 14.23. Окно управления функциями автоматизации

На рисунке 14.23 показана типичная последовательность действий, при которой сначала стирается память, а затем выполняется программирование.

Вначале произведено стирание, затем программирование обоих видов памяти с проверкой после записи и в завершение программируются биты защиты и конфигурации. Для повторного выполнения заданной последовательности достаточно нажать на кнопку *Start*. Это позволяет исключить ошибки, возникающие при многократном выполнении повторяющихся действий.

Контрольные вопросы

- Какие типовые операции выполняются при программировании микроконтроллера?
- Какие операции поддерживает последовательное программирование?
- Какие операции поддерживает только параллельное программирование?
- Какие программно-аппаратные средства используются при программировании микроконтроллеров?
- Какие языки программирования используются при программировании микроконтроллеров?
- Какие файлы создаются при компиляции программы в отладочной среде?
- Какие режимы отладки поддерживает отладочная среда AVRStudio?

- Какие элементы микроконтроллера доступны при отладке программы в среде AVR Studio?
- Какие режимы программирования поддерживает стартовый комплект STK 500?
- Какие компоненты схемы можно подключить к микроконтроллеру в стартовом комплекте STK 500?

15. КУРСОВОЙ ПРОЕКТ

В проекте студентам предлагается реализовать на микроконтроллере один из функциональных узлов системы управления электроприводом или промышленной автоматики: разработать принципиальную схему и программное обеспечение системы.

Задание на курсовой проект

Курсовой проект предусматривает решение следующих задач:

- разработка технического задания;
- выбор микроконтроллера;
- разработка функциональной схемы микропроцессорной системы на базе выбранного микроконтроллера;
- разработка алгоритма работы системы;
- разработка программного обеспечения на языке ассемблера;
- разработка принципиальной схемы микропроцессорной системы;
- анализ результатов проектирования.

Результатом проектирования должно быть законченное устройство с вполне определенными техническими свойствами, например, электронный термометр, измеритель давления, счетчик деталей и др.

Примеры заданий

Задание 1 - Измеритель перемещения с потенциометрическим датчиком пути

Упрощенная функциональная схема системы приведена на рисунке 15.1.

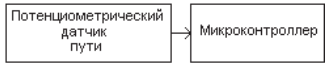


Рис. 15.1. Структура измерителя перемещений с потенциометрическим датчиком

Потенциометрический датчик пути представляет собой линейное изменяющееся сопротивление (рис. 15.2). При движении ползунок перемещается по направляющей сопротивления.

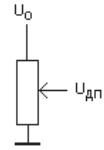


Рис. 15.2. Схема потенциометрического датчика пути

Если потенциометр подсоединить к источнику опорного напряжения U_o, то выходное напряжение потенциометрического датчика рассчитывается по формуле:

$$U_{\partial n} = U_o \cdot \frac{X}{L},$$

где X - текущее положение датчика,

L - максимальное перемещение (диапазон измерения).

Микроконтроллер в системе должен производить периодический (с заданным интервалом) опрос датчика, вычисление фактического перемещения, накопление результатов и хранение последних отсчетов в оперативной памяти. Разрядность представления результата – 1 байт.

Таблица 15.1

Варианты заданий на разработку измерителя перемещения

Тип датчика	Диапазон измерения (мм)	Сопротивление датчика (кОм)	Интервал опроса (мс)	Память отсчетов (байт)
MM10	10	1	1	50
MM30	30	10	2	100
CLP13	50	15	3	128
CLP21	100	20	4	256
SPI	25	12	5	64
SPR18	150	16	6	512
LP	55	500	7	32
CI18	25	2	8	64
CD18	50	4	9	128
CR18	10	5	10	256
RC13	40	5	11	64
RC20	200	10	12	128
MMS33	900	10	13	256
RC35	750	5	14	512

Задание 2 - Измеритель скорости с импульсным датчиком

Разработать микропроцессорную систему измерения скорости вращения электропривода с импульсным датчиком скорости. Структурная схема системы приведена на рисунке 15.3.

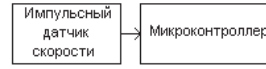


Рис.15.3. Структура измерителя скорости вращения с импульсным датчиком

Датчик скорости генерирует импульсные выходные сигналы. Число импульсов k, формируемых за один оборот вала, определяется разрешением датчика. Сигнал на выходе импульсного датчика скорости представляет собой двухфазную последовательность импульсов X1 и X2, сдвинутых на 90 электрических градусов (рис. 15.4).

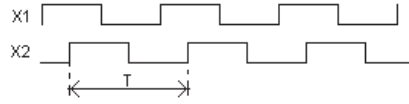


Рис. 15.4. Диаграммы сигналов на выходе импульсного датчика скорости

Частота f и период T следования импульсов на выходе датчика однозначно связана со скоростью вращения вала:

$$f = 1/T = n \cdot k ,$$

где n – скорость вращения вала (оборот/с).

Знак скорости определяется по фазовому сдвигу последовательностей:

- если сигнал X1 опережает X2, скорость считается положительной,
- если сигнал X1 отстает от X2 , скорость считается отрицательной.

Микроконтроллер осуществляет подсчет импульсов за фиксированный интервал времени, оценивает скорость вращения и накапливает результаты расчетов в памяти. Погрешность измерения – 1 об/мин.

Таблица 15.2

Варианты заданий на разработку измерителя скорости вращения с импульсным датчиком перемещения

Вариант	Тип датчика	Разрешение (имп/об)	Максимальная скорость (об/мин)	Память отсчетов (байт)
15	PR24	24	500	50
16	MSP20S\L	20	600	100
17	MRT25	25	700	128
18	MRX20	20	800	256
19	M101	50	900	64
20	SP	100	1000	512
21	SPB	512	1200	32
22	OP	1024	1500	64
23	M350	2500	2000	128
24	MOM20	500	2500	256
25	MBL	60	3000	64
26	MOZ30	300	2000	128
27	MBD	1000	1500	200
28	MOB	2048	1000	400
29	MOL200	5000	500	800
30	MH350	1200	200	1000

Задание 3 - Разработка измерителя перемещения с кодовым датчиком угла

Разработать микропроцессорную систему измерения угловых перемещений с кодовым (абсолютным) датчиком. Структура системы изображена на рисунке 15.5.

Рис. 15.5. Структура измерителя углового положения с кодовым датчиком угла

Кодовый датчик угла формирует двоичный код, соответствующий фактическому угловому положению вала. Основными параметрами кодового датчика являются разрядность представления результата и система кодирования. Обычно используется либо обычный двоичный код, либо код Грея.

Код Грея является разновидностью двоичного кода. В нем каждый бит формируется как сумма по модулю 2 от двух соседних битов двоичного кода:

$$g_i = b_i \oplus b_{i+1},$$

где g_i - бит числа в коде Грея,

b_i – бит числа в двоичном коде,

$i = 0...7$ - номер бита.

Микроконтроллер производит периодический опрос кодового датчика, вычисление фактического положения вала, накопление результатов в памяти.

Таблица 15.3

Варианты заданий на разработку измерителя угловых перемещений с кодовым датчиком угла

Вариант	Тип датчика	Разрядность (бит)	Максимальное перемещение (об)	Максимальная скорость (об/мин)	Интервал опроса (мс)	Память отсчетов (байт)
31	MRP10S/L	4	100	3000	100	50
32	MAM-5	5	50	1500	200	100
33	MAM-8	8	60	1000	500	128
34	MRP10S/L	4	200	750	256	256
35	MAM-5	5	150	375	50	64
36	MAM-8	8	120	200	20	512

Задание 4 - Разработка контроллера клавиатуры

Разработать контроллер матричной клавиатуры с функциональной схемой, изображенной на рисунке 15.6.

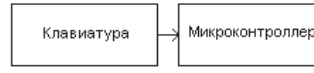


Рис. 15.6. Схема сопряжения клавиатуры с микроконтроллером

Клавиатура содержит набор контактов, объединенных в матрицу. При нажатии любой клавиши клавиатуры микроконтроллер должен произвести её опрос, определить нажатую клавишу и занести скан-код клавиши в ячейку оперативной памяти.

Клавиатура представляет собой набор контактов, объединенных в матрицу (рис. 15.7).

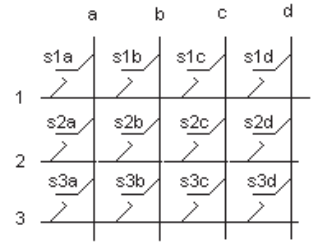


Рис. 15.7. Матричная клавиатура

Контакты клавиатуры размещены на пересечении строк и столбцов матрицы. При замыкании контакта линия выбранной строки соединяется с линией столбца. Например, при замыкании контакта s2b вторая строка матрицы соединяется со столбцом b. Количество строк и столбцов матрицы зависит от типа клавиатуры. Восьмибитный скан-код клавиши формируется из двух тетрад: старшая тетрада содержит информацию о строке, младшая – о столбце. Например, клавиша на пересечении столбца b и строки 2 кодируется двоичным кодом 0b00101011, где 0010 – двоичный эквивалент шестнадцатеричного числа 0x2, 1011 – двоичный эквивалент шестнадцатеричного числа 0xB.

Таблица 15.4

Варианты заданий на разработку схемы сопряжения микроконтроллера с клавиатурой

Вариант	Тип клавиатуры	Память отсчетов (байт)	Интервал опроса (мс)
37	СК04	50	2
38	СК06	100	5
39	СК07	128	10
40	СК09	256	15
41	СК10	64	20
42	СК11	512	25
43	СК12	1024	30

Задание 5 - Сопряжение микроконтроллера с индикатором

Разработать схему сопряжения микроконтроллера с цифровым индикатором (рис. 15.8).

Рис. 15.8. Сопряжение микроконтроллера с цифровым индикатором

Цифровой индикатор состоит из отдельных светодиодных сегментов с объединенными анодами или с объединенными катодами (рис. 15.9).

а) *б)*

Рис. 15.9. Светодиодные индикаторы с объединенными анодами (а) и с объединенными катодами (б)

Количество светодиодов в индикаторе может быть различно. Например, в семисегментном индикаторе семь светодиодов *(a, b, c, d, e ,f, g)*, расположенных в определенном порядке (рисунок 15.10). Индикатор способен отобразить все десятичные цифры от 0 до 9.

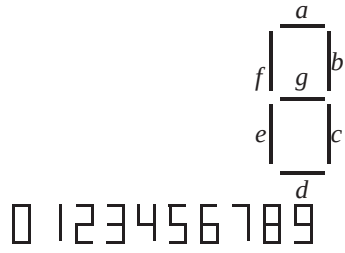


Рис. 15.10. Семисегментный индикатор

В матричных индикаторах светодиоды размещаются на пересечении строк и столбцов матрицы (рис. 15.11).

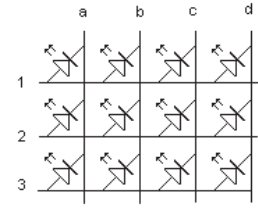


Рис. 15.11. Матричный индикатор с общими анодами

Свечение светодиода обеспечивается при токе порядка 10 мА.

Задачей микроконтроллера является периодическое извлечение данных из памяти, преобразование их в коды индикатора и формирование всех сигналов, необходимых для управления индикатором.

Таблица 15.5

Варианты заданий на разработку схемы сопряжения микроконтроллера с индикатором

Вариант	Тип индикатора	Периодичность обновления (мс)
44	BA56	50
45	BC56	100
46	CA25	128
47	CC25	256
48	TA07	20
49	TC07	10
50	TA15-11	64
51	TC15-11	25
52	TA12-11	75
53	TC12-11	50
54	ITS-E0803	100
55	ITS-E0233	20

Задание 6 - Разработка системы управления широтно-импульсным регулятором постоянного напряжения

Разработать систему управления широтно-импульсным регулятором постоянного напряжения. Структура системы приведена на рисунке 15.12.

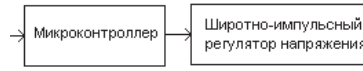


Рис. 15.12. Структура системы управления широтно-импульсным регулятором постоянного напряжения

Широтно-импульсный регулятор собран по транзисторной мостовой схеме (рис. 15.13) и подключается к микроконтроллеру через микросхемы-драйверы.

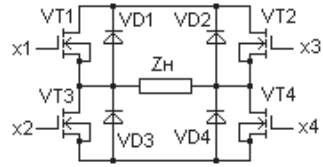


Рис. 15.13. Транзисторный широтно-импульсный регулятор

Активно-индуктивная нагрузка $z_н$ включена в диагональ моста. Микроконтроллер формирует четыре импульсные последовательности x1….x4 для управления ключами (рис. 15.14).

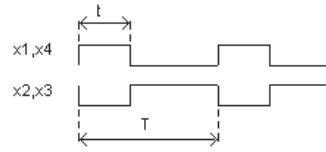


Рис. 15.14. Временная диаграмма управления ключами ШИР

Частота следования импульсов $f = 1/T$ определена в задании (табл. 15.6), соотношение t/T, определяющее среднее значение выходного напряжения ШИР, задается числом, хранящимся в памяти контроллера. Разрядность заданий – 8 бит. Входной сигнал микроконтроллера позволяет блокировать импульсы управления.

Таблица 15.6

Варианты заданий на разработку системы управления широтно-импульсным регулятором постоянного напряжения

Вариант	Частота следования импульсов (Гц)	Разрядность задания
56	100	12
57	200	11
58	250	10
59	50	9
60	100	8
61	120	8
62	70	9
63	90	10
64	160	11

Задание 7 - Разработка системы управления однофазным автономным инвертором напряжения

Разработать систему управления однофазным автономным инвертором напряжения (рис. 15.15). Инвертор собран по транзисторной мостовой схеме (рис. 15.13) и подключен к микроконтроллеру через драйверы управления ключами.

Рис. 15.15. Структура системы управления однофазным автономным инвертором

Микроконтроллер формирует импульсные сигналы управления ключами x1…x4 (рис. 15.16).

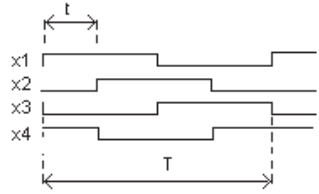


Рис. 15.16. Временные диаграммы импульсов управления АИН

Частота следования импульсов $f = 1/T$ и их фазовый сдвиг t в процессе работы системы могут изменяться. Частота сигнала задается 8-битным числом, поступающим в параллельном виде на порт микроконтроллера, а фазовый сдвиг определяется числом, хранящимся в ячейке его оперативной памяти. Разрядность задания переменных – 8 бит.

Входной сигнал микроконтроллера позволяет блокировать импульсы управления.

Таблица 15.7

Варианты заданий на разработку системы управления широтно-импульсным регулятором постоянного напряжения

Вариант	Максимальная частота (Гц)
65	30
66	40
67	50
68	60
69	70
70	80
71	90
72	100
73	110

Задание 8 - Система управления трехфазным автономным инвертором напряжения

Разработать микропроцессорную систему управления трехфазным автономным инвертором напряжения, по схеме на рисунке 15.17.

Рис. 15.17. Структура системы управления трехфазным автономным инвертором

Трехфазный автономный инвертор собран на шести МОП-транзисторах по мостовой схеме (рисунок 15.18).

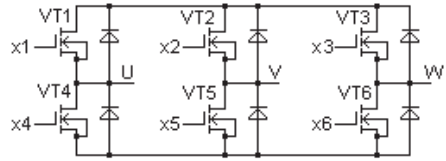


Рис. 15.18. Трехфазный автономный инвертор напряжения

Микроконтроллер формирует импульсные сигналы управления ключами (рис. 15.19).

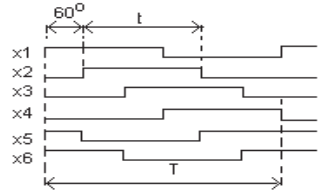


Рис. 15.19. Временные диаграммы управления трехфазным АИН

Частота следования импульсов задается числом, поступающим в параллельном коде на порт микроконтроллера. Максимальная частота и ширина импульсов *t* определены в задании. Дополнительный бит входного сигнала задает порядок следования фаз выходного трехфазного напряжения.

Таблица 15.8

Варианты заданий на разработку системы управления широтно-импульсным регулятором постоянного напряжения

Вариант	Ширина импульса (град)	Максимальная частота (Гц)
74	120	50
75	125	55
76	130	60
77	135	65
78	140	70
79	150	75
80	160	80
81	170	85
82	175	90

Задание 9 - Разработка системы управления тиристорным выпрямителем

Разработать систему управления для тиристорного преобразователя. Структурная схема системы приведена на рисунке 15.20.

Рис. 15.20. Структура системы управления тиристорным выпрямителем

Микроконтроллер формирует сигналы управления тиристорами. Импульсы управления сдвинуты по фазе относительно сигналов синхронизации с сетью на определенный угол. Величина угла управления хранится в памяти в виде 8-битного числа. Длительность импульсов управления и схема преобразователя заданы в таблице.

Таблица 15.9

Варианты заданий на разработку системы управления тиристорным выпрямителем

Вариант	Схема выпрямителя	Ширина импульса (град)
83	Однофазная мостовая нереверсивная	5
84	Трехфазная нулевая нереверсивная	10
85	Трехфазная мостовая нереверсивная	15
86	Однофазная нулевая нереверсивная	5
87	Однофазная нулевая реверсивная	10
88	Однофазная мостовая реверсивная	15
89	Трехфазная нулевая реверсивная	20

Задание 10 - Разработка измерителя токов

Разработать микропроцессорную систему измерения токов в системе электропривода. Структурная схема измерителя показана на рисунке 15.21.

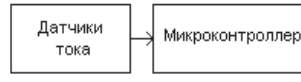


Рис. 15.21. Структурная схема измерителя тока

Датчики напряжения обеспечивают гальваническую развязку электрических цепей и преобразуют токовые сигналы в напряжение. Микроконтроллер опрашивает датчики, оцифровывает сигналы и хранит массив последних от-

счетов в памяти. Количество датчиков, характер сигналов, интервал опроса и разрядность измерений заданы в таблице 15.10.

Таблица 15.10

Варианты заданий на разработку системы измерения тока

Вариант	Характер	Максимальная величина (А)	Количество датчиков	Разрядность (бит)	Интервал опроса (мс)
90	DC	250	2	8	100
91	AC	10	3	8	1
92	DC	10	4	8	200
93	AC	25	5	8	2
94	DC	25	6	8	12
95	AC	50	7	10	3
96	DC	50	8	10	25
97	AC	100	6	10	1,5
98	DC	100	4	10	2
99	AC	250	3	10	1

Примечание:

DC – измерение постоянного тока,

AC – измерение переменного тока 50 Гц.

Методические указания

Проектирование микропроцессорной системы является многоэтапным процессом. На каждом этапе работы решаются определенные задачи, от степени проработки которых зависит успех при решении всей задачи.

В составе микропроцессорной системы одни задачи решаются аппаратным путем, другие программным. Максимальное использование программных средств позволяет получить конкурентоспособное изделие с хорошим соотношением цена/качество.

Разработка функциональной схемы изделия

На первом этапе работы необходимо прояснить все требования технического характера.

Во-первых, необходимо рассмотреть все устройства, подключаемые к микроконтроллеру, их электрические характеристики, схемы включения, временные диаграммы и способы обработки информации.

Во-вторых, необходимо рассмотреть набор внутренних устройств ввода/вывода микроконтроллера: порты, аналоговые и аналого-цифровые элементы, систему прерываний, интерфейсы, таймеры и др.

В результате сопоставления требований устройства и возможностей микроконтроллера принимается решение об использовании тех или иных функциональных блоков для решения поставленной задачи. Например, сигнал с потенциометрического датчика несложно преобразовать в напряжение, а для обработки аналоговых сигналов целесообразно использовать аналоговые компараторы или аналого-цифровые преобразователи. Импульсные датчики, в свою очередь, удобно подключать к таймерам или к внешним входам системы прерываний.

Одна и та же задача может решаться разными способами, все из них можно рассмотреть в первом разделе работы, сопоставить и объяснить выбор одного из них.

На функциональной схеме устройства необходимо показать все используемые блоки микроконтроллера, их взаимосвязь и параметры, обеспечивающие решение поставленной задачи. Функциональная схема оформляется как чертеж с соблюдением всех требований стандартов.

В тексте пояснительной записки должны быть обоснованы все принятые решения, рассчитаны все параметры функциональной схемы. На рисунке 15.22 приведен пример структурной схемы системы

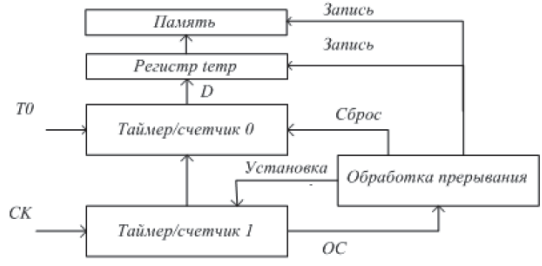


Рис. 15.22. Пример функциональной схемы измерителя скорости с импульсным датчиком

Результаты анализа оформляются в первом разделе пояснительной записки. В конце раздела должна быть приведена взятая за основу проекта функциональная схема, показывающая все использованные узлы микроконтроллера и их сопряжение с объектом. Тут же может быть приведен укрупненный алгоритм, поясняющий функционирование системы, элементов, содержащих все комплектующие схемы.

Разработка алгоритма

Алгоритм работы системы должен наглядно отображать все действия микроконтроллера по решению поставленной задачи. Действия микроконтроллера изображаются на схеме алгоритма отдельными блоками, объединенными линиями связи.

Разработанный алгоритм может быть представлен отдельными, связанными между собой подпрограммами.

Все действия, изображенные на схеме алгоритма, должны быть понятны, они не должны повторять текст программы. В пояснительной записке необходимо обосновать разработанный алгоритм.

Разработка программы

Программа на языке ассемблера должна соответствовать разработанному алгоритму.

В пояснительной записке должно быть подробно обосновано использование тех или иных регистров ввода/вывода, выбор режимов работы функциональных блоков, программирование регистров.

Каждая инструкция программы должна быть прокомментирована, причем комментарий должен объяснять суть команды и необходимость выполнения этих действий. Комментарии, просто переводящие команду на русский язык, недопустимы.

Допускается одним комментарием объяснять несколько действий.

Отладка программы

Отладку программы следует производить на персональном компьютере с использованием интегрированной отладочной среды AVRStudio 3.2. В результате отладки должны быть созданы работоспособные файлы проекта. Листинг отлаженной программы необходимо распечатать и поместить в приложении.

Используя интегрированную отладочную среду студент должен смоделировать работу спроектированной системы и доказать её работоспособность. Отдельные окна программы, иллюстрирующие состояние микропроцессорной системы в различные моменты времени (состояние входов/выходов, регистров) должны быть приведены в тексте пояснительной записки. Отсутствие такой информации существенно снижает ценность проекта.

Разработка принципиальной схемы

Принципиальная схема устройства содержит все элементы системы и связи между ними. Основной задачей при разработке принципиальной схемы является выбор компонентов и их электрическое согласование.

Основным элементом системы является микроконтроллер Atmega16 в корпусе PDIP (рис. 15.23), электрические характеристики которого определяют требования к другим элементам.

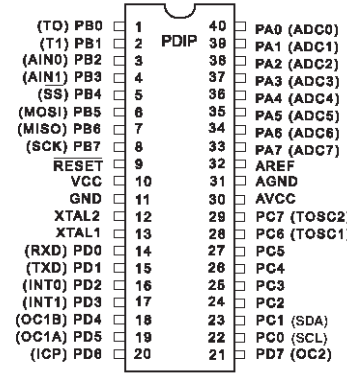


Рис. 15.23. Микроконтроллер Atmega16

Необходимо разобраться с назначением выводов, подключить питание (+5В) к микроконтроллеру, согласовать его выводы с другими элементами схемы. Уровни входных сигналов должны соответствовать уровням микросхем ТТЛ (логическая единица = 2,4…2,5 В). Нагрузочную способность выходов можно принять равной 10 входам элементов ТТЛ, а выходы портов выдерживают ток до 20 мА, допуская прямое подключение светодиодов.

При включении микросхемы необходимо выбрать один из возможных генераторов синхронизации: внешний или внутренний. Внешний генератор необходимо подключить к микроконтроллеру. Ко входу сброса RESET следует подсоединить кнопку или источник внешнего сигнала сброса.

Аналоговые сигналы различных датчиков, сопрягаемых с микроконтроллером, по амплитуде не должны превышать 2,5 В. При подсоединении высоковольтных цепей необходимо использовать цепи электромагнитные или оптоэлектронные цепи гальванической развязки, либо специальные высоковольтные драйверы.

Принципиальная схема должна быть оформлена в соответствии с ГОСТ «Правила выполнения электрических схем». Она должна сопровождаться перечнем элементов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Водовозов, А.М. Микроконтроллеры для систем автоматики : учебное пособие/ А.М.Водовозов. – Вологда : ВоГТУ, 2001. – 123 с.

2. Водовозов, А.М. Элементы систем автоматики : учебное пособие/ А.М.Водовозов. – Москва : Академия, 2008. – 224 с.

3. Гребнев, В. В. Микроконтроллеры семейства AVR фирмы Atmel/ В.В.Гребнев. – Москва : ИП РадиоСофт, 2002. – 176 с:

4. Евстифеев, А.В.Микроконтроллеры AVR семейства Mega : Руководство пользователя/ А.В. Евстифеев. – Москва : Додэка-XXI, 2007. – 592 с.

5. Мортон, Д. Микроконтроллеры AVR. Вводный курс: пер. с англ./ Д.Мортон. – Москва : Додэка-XXI, 2006. – 272 с.

6. Применение микроконтроллеров AVR: схемы, алгоритмы, программы/В.Н. Баранов. – Москва : Додэка-XXI, 2004. — 288 с:

7. Ревич, Ю. В. Практическое программирование микроконтроллеров AtmelAVR на языке ассемблера / Ю.В. Ревич. – Санкт-Петербург : БХВ-Петербург, 2011. – 352 с.

8. Самоучитель разработчика устройств на микроконтроллерах AVR/ А.В.Белов. – Санкт-Петербург : Наука и Техника, 2008. – 544 с.

9. Трамперт, В. Измерение, управление и регулирование с помощью AVR-микроконтроллеров.: пер. с нем./В. Трамперт. – Киев : МК-Пресс, 2006. – 208 с.

10. Трамперт, В. AVR-RISC микроконтроллеры: пер. с нем./ В. Трамперт. – Киев : МК-Пресс, 2006. – 464 с.

11. Хартов, В.Я. Микроконтроллеры AVR. Практикум для начинающих : учеб. пособие / В. Я. Хартов. – Москва : Изд-во МГТУ им. Н. Э. Баумана, 2012. – 280 с.

12. Хартов, В.Я. Микроконтроллеры AVR. Практикум для начинающих/ В.Я. Хартов. – Москва : Изд-во МГТУ им. Н.Э. Баумана, 2007.

13. Atmel Corporation. / Rev.: Atmel-8154C-8-bit-AVR-ATmega16A_Datasheet-07/2014. [Электронный ресурс]: Компания Atmel. – Режим доступа: http://www.atmel.com/ru/ru/Images/doc2466.pdf

14. Примеры применения микроконтроллеров AVR [электронный ресурс]: Рынок микроэлектроники. Режим доступа http://www.gaw.ru/html.cgi/txt/app/micros/avr/index.htm

Приложение 1

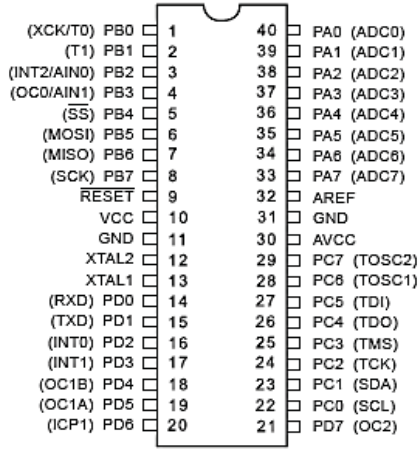
Регистры ввода/вывода микроконтроллера ATmega16

I/O адрес	*Имя*	Функция регистра
0x3F	SREG	*Status Register*
0x3E	SPH	*Stack Pointer High*
0x3D	SPL	*Stack Pointer Low*
0x3C	OCR0	*Output Compare Register Timer/Counter 0*
0x3B	GICR	*General Interrupt Control Register*
0x3A	GIFR	*General Interrupt Flag Register*
0x39	TIMSK	*Timer/Counter Interrupt Mask Register*
0x38	TIFR	*Timer/Counter Interrupt Flag Register*
0x37	SPMCR	*Store Program Memory Control Register*
0x36	TWCR	*2-wire Serial Interface Control Register*
0x35	MCUCR	*MCU Control Register*
0x34	MCUCSR	*MCU Control and Status Register*
0x33	TCCR0	*Timer/Counter0 Control Register*
0x32	TCNT0	*Timer/Counter0 (8-bit)*
0x31	OSCCAL	*Oscillator Calibration Register*
0x30	SFIOR	*Special Function I/O Register*
0x2F	TCCR1A	*Timer/Counter1 Control Register A*
0x2E	TCCR1B	*Timer/Counter1 Control Register B*
0x2D	TCNT1H	*Timer/Counter1 High-byte*
0x2C	TCNT1L	*Timer/Counter1 Low-byte*
0x2B	OCR1AH	*Timer/Counter1 Output Compare Register A High-byte*
0x2A	OCR1AL	*Timer/Counter1 Output Compare Register A Low-byte*
0x29	OCR1BH	*Timer/Counter1 Output Compare Register B High-byte*
0x28	OCR1BL	*Timer/Counter1 Output Compare Register B Low-byte*
0x27	ICR1H	*Input Capture Register High-byte*
0x26	ICR1L	*Input Capture Register Low-byte*
0x25	TCCR2	*Timer/Counter2 Control Register*
0x24	TCNT2	*Timer/Counter2 (8-bit)*
0x23	OCR2	*Timer/Counter2 Output Compare Register*

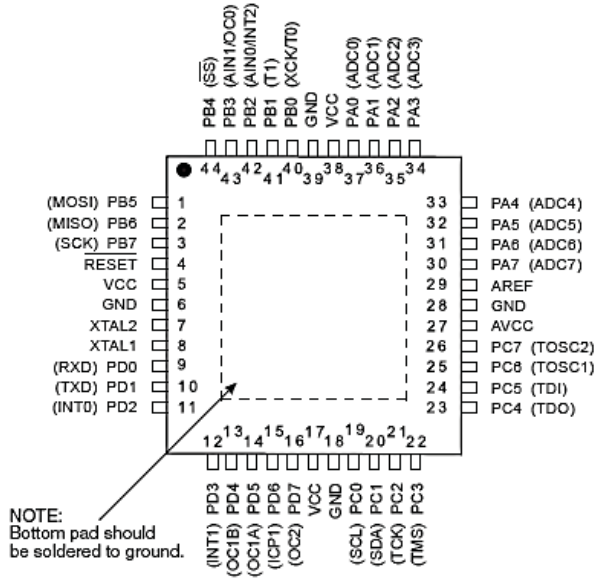
I/O адрес	*Имя*	Функция регистра
0x22	ASSR	*Asynchronous Mode Status Register*
0x21	WDTCR	*Watchdog Timer Control Register*
0x20	UBRRHI	*UART Baud Rate Register High-byte*
0x1F	EEARH	*EEPROM Address Register High-byte*
0x1E	EEARL	*EEPROM Address Register Low-byte*
0x1D	EEDR	*EEPROM Data Register*
0x1C	EECR	*EEPROM Control Register*
0x1B	PORTA	*Data Register, Port A*
0x1A	DDRA	*Data Direction Register, Port A*
0x19	PINA	*Input Pins, Port A*
0x18	PORTB	*Data Register, Port B*
0x17	DDRB	*Data Direction Register, Port B*
0x16	PINB	*Input Pins, Port B*
0x15	PORTC	*Data Register, Port C*
0x14	DDRC	*Data Direction Register, Port C*
0x13	PINC	*Input Pins, Port C*
0x12	PORTD	*Data Register, Port D*
0x11	DDRD	*Data Direction Register, Port D*
0x10	PIND	*Input Pins, Port D*
0x0F	SPDR	*SPI I/O Data Register*
0x0E	SPSR	*SPI Status Register*
0x0D	SPCR	*SPI Control Register*
0x0C	UDR	*UART I/O Data Register*
0x0B	UCSRA	*UART Control and Status Register A*
0x0A	UCSRB	*UART Control and Status Register B*
0x09	UBRR	*UART Baud Rate Register*
0x08	ACSR	*Analog Comparator Control and Status Register*
0x07	ADMUX	*ADC Multiplexer Select Register*
0x06	ADCSR	*ADC Control and Status Register*
0x05	ADCH	*ADC Data Register High*
0x04	ADCL	*ADC Data Register Low*
0x03	TWDR	*Two-wire Serial Interface Data Register*
0x02	TWAR	*Two-wire Serial Interface Address Register*
0x01	TWSR	*Two-wire Serial Interface Status Register*
0x00	TWBR	*Two-wire Serial Interface Bit Rate Register*

Корпуса микроконтроллера ATmega16

Корпус DIP (Dual In Line pin Package)



Корпус TQFP (Thin Plastic Quad Flat Package)



Вектора прерываний контроллера ATmega16

Вектор	Адрес	Обозначение источника	Описание источника	
1	0x000	RESET	Reset	Сброс
2	0x002	INT0	Interrupt 0	Внешнее прерывание 0
3	0x004	INT1	Interrupt 1	Внешнее прерывание 1
4	0x006	TIMER2 COMP	Output Compare2	Таймер/счетчик 2, сравнение
5	0x008	TIMER2 OVF	Overflow 2	Таймер/счетчик 2, переполнение
6	0x00A	TIMER1 CAPT	Input Capture1	Таймер/счетчик 1, захват
7	0x00C	TIMER1 COMPA	Timer/Counter 1 Compare Matc A	Таймер/счетчик 1, сравнение A
8	0x00E	TIMER1 COMPB	Timer/Counter 1 Compare Matc B	Таймер/счетчик 1, сравнение B
9	0x010	TIMER1 OVF	Overflow 1	Таймер/счетчик 1, переполнение
10	0x012	TIMER0 OVF	Overflow 0	Таймер/счетчик 0, переполнение
11	0x014	SPI STC	SPI Serial Transfer Complete	SPI, передача завершена
12	0x016	USART RXC	USART Receive Complete	UART, прием завершен
13	0x018	USART DRE	USART Data Register Empty	UART, регистр данных пуст
14	0x01A	USART TXC	UART Transmit Complete	UART, передача завершена
15	0x01C	ADC	ADC	Аналого-цифровой преобразователь
16	0x01E	EE_RDY	EEPROM	Память EEPROM
17	0x020	ANA_COMP	Analog Comparator	Аналоговый компаратор
18	0x022	TWI	TWI	Двухпроводной интерфейс Interface)
19	0x24	INT2	External Interrupt Request 2	Внешнее прерывание 2
20	0x26	TIMER0 COMP	Timer/Counter 0 Compare Matc	Таймер/счетчик 0 сравнение
21	0x28	SPM_RDY	Store Program Memory Ready	Загрузка программной памяти

ГЛОССАРИЙ ТЕРМИНОВ И СОКРАЩЕНИЙ

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	*Fusebit*	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
ICR1HT/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	*Lockbit*	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	*StackPointer*	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 А таймера/счетчика 1
TCCR1B	*Timer/Counter1 Control Register B*	Регистр управления В 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 А управления и контроля UART
UCSRB	*UART Control and Status Register B*	Регистр В 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

Учебное издание

Александр Михайлович Водовозов

МИКРОКОНТРОЛЛЕРЫ ДЛЯ СИСТЕМ АВТОМАТИКИ

Учебное пособие

Редактор – Л.А. Перерукова

Подписано в печать 26.06.2016
Формат 60x84/16. Бумага офсетная.
Гарнитура «Таймс».
Тираж 500 экз. Заказ №497

Издательство «Инфра-Инженерия»
Тел.: 8(911)512-48-48
E-mail: infra-e@yandex.ru
www.infra-e.ru

Издательство приглашает
к сотрудничеству авторов
научно-технической литературы