Таран В.Н., Трофименко В.Н., Трофименко Е.Н.

Микропроцессорные устройства РЭО

Исследования микроконтроллеров

Ростов-на-Дону

УДК 681.3.06

Таран В.Н., Трофименко В.Н., Трофименко Е.Н. Микропроцессорные устройства РЭО: Исследования микроконтроллеров. Учебно-методическое пособие по проведению лабораторных занятий.

Содержит теоретические сведения по архитектуре микроконтроллеров. На примере микроконтроллеров *AVR* семейства *Classic* рассмотрены особенности архитектуры современных микроконтроллеров. Рассмотрена среда создания приложений - *AVR Studio* и представлены методические материалы по исследованию микроконтроллеров *AVR*: арифметических и логических операций, команд ветвления, портов ввода/вывода, таймера-счетчика в курсе дисциплины "Микропроцессорные устройства РЭО". Материалы пособия также могут использоваться в курсовом и дипломном проектировании при разработке микроконтроллерных устройств управления.

Предназначено для студентов заочного обучения по специальности 160903 “Техническая эксплуатация авиационных электросистем и пилотажно-навигационных комплексов".

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Микропроцессорная революция оказала влияние на архитектуру систем управления не только благодаря снижению стоимости управляющих компьютеров. Произошел отказ от полностью централизованного управления и переход к модульной архитектуре, появились удаленные микропроцессорные модули, интегрирующие отдельные авиационные системы в единое целое, получившее в 70-е годы прошлого века наименование "авионика", и обеспечившие высокие технико-экономические показатели систем. Следует отметить, что доля авионики в стоимости современного самолета составляет 70 – 80% [1, 2].

Термин «авионика» связан с появлением интегральных микроэлектронных технологий и созданием принципиально новых автоматизированных систем контроля и управления на основе компактных бортовых высокопроизводительных компьютеров (БЦВМ). В настоящее время основой построения БЦВМ являются микроконтроллеры различного функционального назначения и различной производительности, которые, в отличие от универсальных микропроцессоров, сочетают в себе не только операционный блок и устройство управления, но и память программ и данных, набор периферийных устройств, перестраиваемые порты ввода-вывода. Интеграция в одном кристалле перечисленных элементов стало тем преимуществом, позволившим снизить габариты, потребляемую энергию, повысить универсальность устройств управления и обработки сигналов. Поэтому знание принципов работы микропроцессоров, основ проектирования систем на их основе, а также способов программирования микропроцессорных систем необходимо не только разработчикам вычислительной техники, но и специалистам отраслей, где используются микропроцессоры, в частности, инженерам, разрабатывающих и эксплуатирующих авиационные электронные системы.

В настоящем методическом пособии представлен теоретический материал по архитектуре микроконтроллеров, представлены массовые универсальные 8-разрядные микроконтроллеры – AVR-микроконтроллеры фирмы Atmel, отличающиеся высокой производительностью при общих свойствах архитектуры и структуры в своем классе устройств. Представлены методические указания по выполнению лабораторных работ по исследованию среды разработки приложений на основе микроконтроллеров, по исследованию системы команд и по исследованию периферийных модулей (портов ввода-вывода, таймеров в режиме формирования временного интервала и в режиме сравнения).

1 МИКРОКОНТРОЛЛЕРЫ

1.1 МИКРОПРОЦЕССОРНЫЕ КОМПЛЕКТЫ БИС/СБИС

Микропроцессором (МП) называют построенное на одной или нескольких БИС/СБИС программно-управляемое устройство, осуществляющее процесс обработки информации и управление им.

МП – центральный процессорный элемент ***микропроцессорной системы*** (МПС или микро-ЭВМ), в которую также входят память и устройства ввода/вывода (внешние устройства).

Решаемая задача определяется реализуемой программой, структура микропроцессорной системы остается неизменной, что и определяет ее универсальность.

Совокупность больших/сверхбольших интегральных схем (БИС/СБИС), пригодных для совместного применения в составе микро-ЭВМ, называют ***микропроцессорным комплектом БИС/СБИС*** (МПК).

Понятие МПК задает номенклатуру микросхем с точки зрения возможностей их совместного применения (совместимость по архитектуре, электрическим параметрам, конструктивным признакам и др.). В состав МПК могут входить микросхемы различных серий и схемотехнологических типов при условии их совместимости.

1.2 КЛАССИФИКАЦИЯ И СТРУКТУРА МИКРОКОНТРОЛЛЕРОВ

Основной особенностью современного этапа развития микропроцессорных систем является завершение перехода от систем, выполненных на основе нескольких БИС, к однокристальным микроконтроллерам (МК). Микроконтроллеры – разновидность МПС, ориентированная на реализацию алгоритмов управления техническими устройствами и технологическими процессами. Первые МК выпущены фирмой *Intel* в 1976 г (восьмиразрядный МК 8048).

МК реализуют заранее известные несложные алгоритмы, и для размещения программ им требуются меньшей, чем у микро-ЭВМ широкого назначения, емкости памяти. Набор внешних устройств также существенно сужается, а сами они значительно проще. Это позволяет все модули микро-ЭВМ разместить на одном кристалле.

МК объединяют в одном кристалле все основные элементы МПС: центральный процессор (ЦП), постоянное запоминающее устройство (ПЗУ), оперативное запоминающее устройство (ОЗУ), последовательные и параллельные порты ввода/выводы, таймеры, аналого-цифровые и цифро-аналоговые преобразователи, широтно-импульсные модуляторы и другие узлы вычислительной системы. Поэтому вторым названием МК стало название "однокристальная микро-ЭВМ".

Для большинства приложений упрощенной структуры МК оказывается достаточной. Это и определяет их преобладание в таких областях, как бытовая аппаратура, станкостроение, автомобильная промышленность и т. д.

В настоящее время выпускается целый ряд типов МК. Все эти приборы можно условно разделить на три основных класса [3]:

- 8-разрядные МК для встраиваемых приложений;
- 16- и 32-разрядные МК;
- цифровые сигнальные процессоры (DSP).

Наиболее распространенным представителем семейства МК являются 8-разрядные приборы, широко используемые в промышленности, бытовой и компьютерной технике. Они прошли в своем развитии путь от простейших приборов с относительно слаборазвитой периферией до современных многофункциональных контроллеров, обеспечивающих реализацию сложных алгоритмов управления в реальном масштабе времени. Причиной жизнеспособности 8-разрядных МК является использование их для управления реальными объектами, где применяются, в основном, алгоритмы с преобладанием логических операций, скорость обработки которых практически не зависит от разрядности процессора.

Росту популярности 8-разрядных МК способствует постоянное расширение номенклатуры изделий, выпускаемых такими известными фирмами, как *Motorola, Microchip, Intel, Zilog, Atmel* и многими другими.

Современные 8-разрядные МК обладают, как правило, рядом отличительных признаков. Перечислим основные из них:

- ***модульная организация***, при которой на базе одного процессорного ядра (центрального процессора) проектируется ряд (линейка) МК, различающихся объемом и типом памяти программ, объемом памяти данных, набором периферийных модулей, частотой синхронизации;
- использование ***закрытой архитектуры МК***, которая характеризуется отсутствием линий магистралей адреса и данных на выводах корпуса МК. Таким образом, МК представляет собой законченную систему обработки данных, наращивание возможностей которой с использованием параллельных магистралей адреса и данных не предполагается;
- использование ***типовых функциональных периферийных модулей*** (таймеры, процессоры событий, контроллеры последовательных интерфейсов, аналого-цифровые преобразователи и др.), имеющих незначительные отличия в алгоритмах работы в МК различных производителей;
- расширение числа режимов работы периферийных модулей, которые задаются в процессе инициализации регистров специальных функций МК.

1.3 МОДУЛЬНАЯ ОРГАНИЗАЦИЯ МИКРОКОНТРОЛЛЕРОВ

При модульном принципе построения все МК одного семейства содержат ***процессорное ядро***, одинаковое для всех МК данного семейства, и изменяемый

функциональный блок, который отличает МК разных моделей. Структура модульного МК приведена на рисунке 1.1 [3].

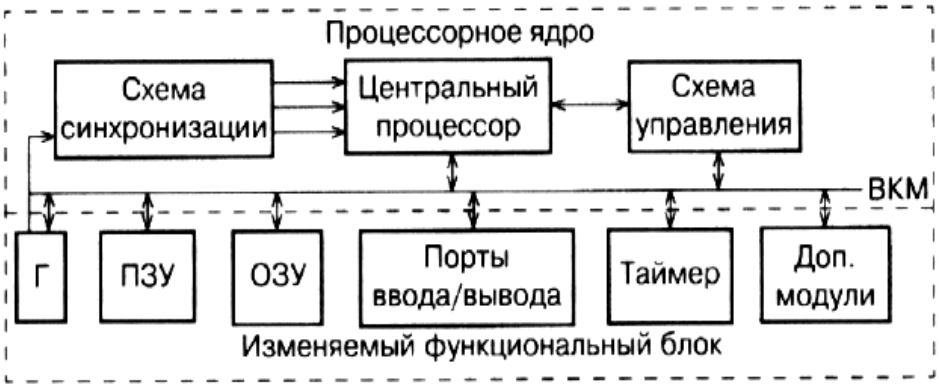


Рисунок 1.1 – Модульная организация МК

Процессорное ядро включает в себя:

– центральный процессор;

– внутреннюю контроллерную магистраль (ВКМ) в составе шин адреса, данных и управления;

– схему синхронизации МК;

– схему управления режимами работы МК, включая поддержку режимов пониженного энергопотребления, начального запуска (сброса) и т.д.

Схема синхронизации МК обеспечивает формирование сигналов синхронизации, необходимых для выполнения командных циклов центрального процессора, а также обмена информацией по внутренней магистрали. В зависимости от исполнения центрального процессора командный цикл может включать в себя от одного до нескольких (4 – 6) тактов синхронизации. Схема синхронизации формирует также метки времени, необходимые для работы таймеров МК. В состав схемы синхронизации входят делители частоты, которые формируют необходимые последовательности синхросигналов.

Основными характеристиками, определяющими производительность процессорного ядра МК, являются:

– набор регистров для хранения промежуточных данных;

– система команд процессора;

– способы адресации операндов в пространстве памяти;

– организация процессов выборки и исполнения команды.

Изменяемый функциональный блок включает в себя модули памяти различного типа и объема, порты ввода/вывода, модули тактовых генераторов (Г), таймеры. В относительно простых МК модуль обработки прерываний входит в состав процессорного ядра. В более сложных МК он представляет собой отдельный модуль с развитыми возможностями. В состав изменяемого функционального блока могут входить и такие дополнительные модули как компараторы напряжения, аналого-цифровые преобразователи (АЦП) и другие. Каждый модуль проектируется для работы в составе МК с учетом протокола ВКМ. Данный подход позволяет создавать разнообразные по структуре МК в пределах одного семейства.

1.4 АРИХИТЕКТУРА МИКРОКОНТРОЛЛЕРОВ

Под ***архитектурой*** [3] микроконтроллера (микропроцессора) понимается комплекс его аппаратных и программных средств, предоставляемых пользователю. В это общее понятие входит набор программно-доступных регистров и исполнительных (операционных) устройств, система основных команд и способов адресации, объем и структура адресуемой памяти, виды и способы обработки прерываний. Например, все модификации микроконтроллеров Classic фирмы Atmel имеют AVR архитектуру которая характеризуется стандартным набором регистров, предоставляемых пользователю, общей системой основных команд и способов организации и адресации памяти, одинаковой реализацией защиты памяти и обслуживания прерываний.

При описании архитектуры и функционирования процессора обычно используется его представление в виде совокупности программно-доступных регистров, образующих ***регистровую*** или ***программную модель***. В этих регистрах содержатся обрабатываемые данные (операнды) и управляющая информация. Соответственно, в регистровую модель входит группа регистров общего назначения, служащих для хранения операндов, и группа служебных регистров, обеспечивающих управление выполнением программы и режимом работы процессора, организацию обращения к памяти (защита памяти, сегментная и страничная организация и др.).

Регистры общего назначения образуют РЗУ - внутреннюю регистровую память процессора. Состав и количество служебных регистров определяется архитектурой микропроцессора. Обычно в их состав входят:

- программный счетчик *PC*;
- регистр состояния *SR* (или *EFLAGS*);
- регистры управления режимом работы процессора *CR* (*Control Register*);
- регистры, реализующие сегментную и страничную организацию памяти;
- регистры, обеспечивающие отладку программ и тестирование процессора.

Кроме того, различные модели микропроцессоров содержат ряд других специализированных регистров.

1.4.1 Принстонская и гарвардская архитектура

С точки зрения организации процессов выборки и исполнения команды в современных 8-разрядных МК применяется одна из двух уже упоминавшихся архитектур МПС: фон-неймановская (принстонская) или гарвардская.

Основной особенностью фон-неймановской архитектуры является использование общей памяти для хранения программ и данных, как показано на рисунке 1.2.

Рисунок 1.2 – Структура МПС с фон-неймановской архитектурой

Основное преимущество архитектуры Фон-Неймана - упрощение устройства МПС, так как реализуется обращение только к одной общей памяти. Кроме того, использование единой области памяти позволяло оперативно перераспределять ресурсы между областями программ и данных, что существенно повышало гибкость МПС с точки зрения разработчика программного обеспечения. Размещение стека в общей памяти облегчало доступ к его содержимому. Неслучайно поэтому фон-неймановская архитектура стала основной архитектурой универсальных компьютеров, включая персональные компьютеры.

Основной особенностью гарвардской архитектуры является использование раздельных адресных пространств для хранения команд и данных, как показано на рисунке 1.3.

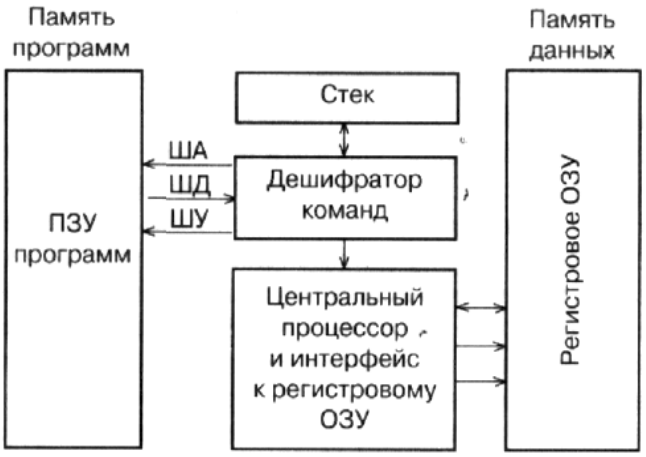


Рисунок 1.3 – Структура МПС с гарвардской архитектурой

Гарвардская архитектура почти не использовалась до конца 70-х годов, пока производители МК не поняли, что она дает определенные преимущества разработчикам автономных систем управления.

Дело в том, что, судя по опыту использования МПС для управления различными объектами, для реализации большинства алгоритмов управления такие преимущества фон-неймановской архитектуры как гибкость и универсальность не имеют большого значения. Анализ реальных программ управления показал, что необходимый объем памяти данных МК, используемый для хранения промежуточных результатов, как правило, на порядок меньше требуемого объема памяти программ. В этих условиях использование единого адресного пространства приводило к увеличению формата команд за счет увеличения числа разрядов для адресации операндов. Применение отдельной небольшой по объему памяти данных способствовало сокращению длины команд и ускорению поиска информации в памяти данных.

Кроме того, гарвардская архитектура обеспечивает потенциально более высокую скорость выполнения программы по сравнению с фон-неймановской за счет возможности реализации параллельных операций. Выборка следующей

команды может происходить одновременно с выполнением предыдущей, и нет необходимости останавливать процессор на время выборки команды. Этот метод реализации операций позволяет обеспечивать выполнение различных команд за одинаковое число тактов, что дает возможность более просто определить время выполнения циклов и критичных участков программы.

Большинство производителей современных 8-разрядных МК используют гарвардскую архитектуру. Однако гарвардская архитектура является недостаточно гибкой для реализации некоторых программных процедур. Поэтому сравнение МК, выполненных по разным архитектурам, следует проводить применительно к конкретному приложению.

1.4.2 *CISC*- и *RISC*-архитектура

С точки зрения системы команд и способов адресации операндов процессорное ядро современных 8-разрядных МК реализует один из двух принципов построения процессоров:

- процессоры с *CISC*-архитектурой, реализующие так называемую полную систему команд (*Complicated Instruction Set Computer*);
- процессоры с *RISC*-архитектурой, реализующие сокращенную систему команд (*Reduced Instruction Set Computer*).

CISC-процессоры выполняют большой набор команд с развитыми возможностями адресации, давая разработчику возможность выбрать наиболее подходящую команду для выполнения необходимой операции. В применении к 8-разрядным МК процессор с *CISC*-архитектурой может иметь однобайтовый, двухбайтовый и трехбайтовый (редко четырехбайтовый) формат команд. При этом система команд, как правило, неортогональна, то есть не все команды могут использовать любой из способов адресации применительно к любому из регистров процессора. Выборка команды на исполнение осуществляется побайтно в течение нескольких циклов работы М К. Время выполнения команды может составлять от 1 до 12 циклов. К МК с *CISC*-архитектурой относятся МК фирмы *Intel* с ядром *MCS-51*, которые поддерживаются в настоящее время целым рядом производителей, МК семейств *HC05, HC08* и *HCII* фирмы *Motorola* и ряд других.

В процессорах с *RISC*-архитектурой набор исполняемых команд сокращен до минимума. Для реализации более сложных операций приходится комбинировать команды. При этом все команды имеют формат фиксированной длины (например, 12, 14 или 16 бит), выборка команды из памяти и ее исполнение осуществляется за один цикл (такт) синхронизации. Система команд *RISC*-процессора предполагает возможность равноправного использования всех регистров процессора. Это обеспечивает дополнительную гибкость при выполнении ряда операций. К МК с *RISC*-процессором относятся МК *AVR* фирмы - AtmeI, МК *PIC16* и *PIC17* фирмы *Microchip* и другие.

1.4.3 Система команд МК

Так же, как и в любой микропроцессорной системе, набор команд процессора МК включает в себя четыре основные группы команд:

- команды пересылки данных;
- арифметические команды;
- логические команды;
- команды переходов.

Для реализации возможности независимого управления разрядами портов (регистров) в большинстве современных МК предусмотрена также группа команд битового управления (булевый или битовый процессор). Наличие команд битового процессора позволяет существенно сократить объем кода управляющих программ и время их выполнения.

В ряде МК выделяют также группу команд управления ресурсами контроллера, используемую для настройки режимов работы портов ввода/вывода, управления таймером и т.п. В большинстве современных МК внутренние ресурсы контроллера отображаются на память данных, поэтому для целей управления ресурсами используются команды пересылки данных.

Система команд МК по сравнению с системой команд универсального МП имеет, как правило, менее развитые группы арифметических и логических команд, зато более мощные группы команд пересылки данных и управления. Эта особенность связана со сферой применения МК, требующей, прежде всего, контроля окружающей обстановки и формирования

1.4.4 Память программ и данных МК

В МК используется три основных вида памяти. Память программ представляет собой постоянную память (ПЗУ), предназначенную для хранения программного кода (команд) и констант. Ее содержимое в ходе выполнения программы не изменяется. Память данных предназначена для хранения переменных в процессе выполнения программы и представляет собой ОЗУ. Регистры МК — этот вид памяти включает в себя внутренние регистры процессора и регистры, которые служат для управления периферийными устройствами (регистры специальных функций).

1.4.4.1 Память программ

Основным свойством памяти программ является ее энергонезависимость, то есть возможность хранения программы при отсутствии питания. С точки зрения пользователей МК следует различать следующие типы энергонезависимой памяти программ:

- ПЗУ масочного типа — *mask-ROM*. Содержимое ячеек ПЗУ этого типа заносится при ее изготовлении с помощью масок и не может быть впоследствии заменено или перепрограммировано. Поэтому МК с таким типом памяти программ следует использовать только после достаточно длительной опытной экс-

плуатации. Основным недостатком данной памяти является необходимость значительных затрат на создание нового комплекта фотошаблонов и их внедрение в производство. Обычно такой процесс занимает 2-3 месяца и является экономически выгодным только при выпуске десятков тысяч приборов. ПЗУ масочного типа обеспечивают высокую надежность хранения информации по причине программирования в заводских условиях с последующим контролем результата.

– ПЗУ, программируемые пользователем, с ультрафиолетовым стиранием — *EPROM* (*Erasable Programmable ROM*). ПЗУ данного типа программируются электрическими сигналами и стираются с помощью ультрафиолетового облучения. Ячейка памяти EPROM представляет собой МОП-транзистор с «плавающим» затвором, заряд на который переносится с управляющего затвора при подаче соответствующих электрических сигналов. Для стирания содержимого ячейки она облучается ультрафиолетовым светом, который сообщает заряду на плавающем затворе энергию, достаточную для преодоления потенциального барьера и стекания на подложку. Этот процесс может занимать от нескольких секунд до нескольких минут. МК с *EPROM* допускают многократное программирование и выпускаются в керамическом корпусе с кварцевым окошком для доступа ультрафиолетового света. Такой корпус стоит довольно дорого, что значительно увеличивает стоимость МК. Для уменьшения стоимости МК с *EPROM* его заключают в корпус без окошка (версия *EPROM* с однократным программированием).

– ПЗУ, однократно программируемые пользователем, — *OTPROM* (*One-Time Programmable ROM*). Представляют собой версию *EPROM*, выполненную в корпусе без окошка для уменьшения стоимости МК на его основе. Сокращение стоимости при использовании таких корпусов настолько значительно, что в последнее время эти версии *EPROM* часто используют вместо масочных ПЗУ.

– ПЗУ, программируемые пользователем, с электрическим стиранием - *EEPROM* (*Electrically Erasable Programmable ROM*). ПЗУ данного типа можно считать новым поколением *EPROM*, п которых стирание ячеек памяти производится также электрическими сигналами за счет использования туннельных механизмов. Применение *EEPROM* позволяет стирать и программировать МК, не снимая его с платы. Таким способом можно производить отладку и модернизацию программного обеспечения. Это дает огромный выигрыш на начальных стадиях разработки микроконтроллерных систем или в процессе их изучения, когда много времени уходит на поиск причин неработоспособности системы и выполнение циклов стирания-программирования памяти программ. По цене *EEPROM* занимают среднее положение между *OTPROM* и *EPROM*. Технология программирования памяти *EEPROM* допускает побайтовое стирание и программирование ячеек. Несмотря на очевидные преимущества *EEPROM*, только в редких моделях МК такая память используется для хранения программ. Связано это с тем, что, во-первых, *EEPROM* имеют ограниченный объем памяти. Во-вторых, почти одновременно с *EEPROM* появились *Flash*-ПЗУ, которые при

сходных потребительских характеристиках имеют более низкую стоимость;

– ПЗУ с электрическим стиранием типа *Rash — Flash-ROM*. Функционально *Flash*-память мало отличается от *EEPROM*. Основное различие состоит в способе стирания записанной информации. В памяти *EEPROM* стирание производится отдельно для каждой ячейки, а во *Flash*-памяти стирать можно только целыми блоками. Если необходимо изменить содержимое одной ячейки Flash-памяти, потребуется перепрограммировать весь блок. Упрощение декодирующих схем по сравнению с *EEPROM* привело к тому, что МК с *Flash*-памятью становятся конкурентоспособными по отношению не только к МК с однократно программируемыми ПЗУ, но и с масочными ПЗУ также.

1.4.4.2 Память данных

Память данных МК выполняется, как правило, на основе статического ОЗУ. Термин "статическое" означает, что содержимое ячеек ОЗУ сохраняется при снижении тактовой частоты МК до сколь угодно малых значений (с целью снижения энергопотребления). Большинство МК имеют такой параметр, как «напряжение хранения информации» - $U_{STANDBY}$. При снижении напряжения питания ниже минимально допустимого уровня U_{DDMIN}, но выше уровня $U_{STANDBY}$ работа программы МК выполняться не будет, но информация в ОЗУ сохраняется. При восстановлении напряжения питания можно будет сбросить МК и продолжить выполнение программы без потери данных. Уровень напряжения хранения составляет обычно около 1 В, что позволяет в случае необходимости перевести МК на питание от автономного источника (батареи) и сохранить в этом режиме данные ОЗУ.

Объем памяти данных МК, как правило, невелик и составляет обычно десятки и сотни байт. Это обстоятельство необходимо учитывать при разработке программ для МК. Так, при программировании МК константы, если возможно, не хранятся как переменные, а заносятся в ПЗУ программ. Максимально используются аппаратные возможности МК, в частности, таймеры. Прикладные программы должны ориентироваться на работу без использования больших массивов данных.

1.4.4.3 Регистры МК

Как и все МПС, МК имеют набор регистров, которые используются как для временного хранения данных (регистры общего назначения или РОН), так и для управления его ресурсами (специальные регистры).

РОН используются для временного хранения операндов и результатов выполнения команд, а также используются при выполнении команд пересылок данных между в качестве источников или приемников двоичных кодов.

В число специальных регистров входят обычно регистры процессора (аккумулятор, регистры состояния, индексные регистры), регистры управления (регистры управления прерываниями, таймером), регистры, обеспечивающие ввод/вывод данных (регистры данных портов, регистры управления параллельным, последовательным или аналоговым вводом/выводом). Обращение к этим регистрам может производиться по-разному.

В МК с *RISC*-процессором все регистры (часто и аккумулятор) располагаются по явно задаваемым адресам. Это обеспечивает более высокую гибкость при работе процессора.

По принципам отображения адресов регистров на общее адресное пространство памяти МК разделяют на МК с общим адресным пространством и МК с раздельным адресным пространством.

В первом типе МК все регистры и память данных располагаются в одном адресном пространстве. Это означает, что память данных совмещена с регистрами. Такой подход называется еще называется "отображением ресурсов МК на память".

Во втором типе МК адресное пространство регистров, как общего назначения, так и специальных отделено от общего пространства памяти. Отдельное пространство ввода/вывода дает некоторое преимущество процессорам с гарвардской архитектурой, обеспечивая возможность считывать команду во время обращения к регистру ввода/вывода.

1.4.4.4 Стек МК

Память для стека или стек (*Stack*) это часть оперативной памяти, предназначенная для временного хранения данных в режиме *LIFO* (*Last In - First Out*). В МК стек используется для организации вызова подпрограмм и обработки прерываний. При этих операциях содержимое программного счетчика и основных регистров (аккумулятор, регистр состояния и другие) сохраняется и затем восстанавливается при возврате к основной программе.

Особенность стека по сравнению с другой оперативной памятью – это заданный и неизменяемый способ адресации. При записи любого числа (кода) в стек число записывается по адресу, определяемому как содержимое регистра указателя стека, предварительно уменьшенное (декрементированное) на единицу (или на два, если 16-разрядные слова расположены в памяти по четным адресам). При чтении из стека число читается из адреса, определяемого содержимым указателя стека, после чего это содержимое указателя стека увеличивается (инкрементируется) на единицу (или на два), В результате получается, что число, записанное последним, будет прочитано первым, а число, записанное первым, будет прочитано последним. Такая память называется *LIFO* или памятью магазинного типа (например, в магазине автомата патрон, установленный последним, будет извлечен первым).

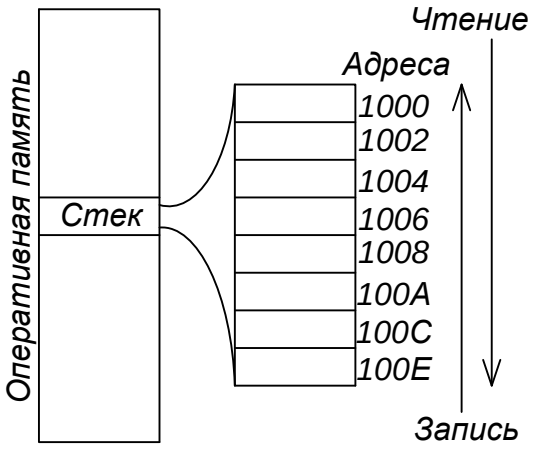


Рисунок 1.4 – Принцип работы стека

Принцип действия стека показан на рисунке 1.4 (адреса ячеек памяти выбраны условно).

Пусть, например, текущее состояние указателя стека 1000008, и в него надо записать два числа (слова). Первое слово будет записано по адресу 1000006 (перед записью указатель стека уменьшится на два). Второе

– по адресу 1000004. После записи содержимое указателя стека – 1000004. Если затем прочитать из стека два слова, то первым будет прочитано слово из адреса 1000004, а после чтения указатель стека станет равным 1000006. Вторым будет прочитано слово из адреса 1000006, а указатель стека станет равным 1000008. Все вернулось к исходному состоянию. Первое записанное слово читается вторым, а второе – первым.

Необходимость такой адресации становится очевидной в случае многократно вложенных подпрограмм. Пусть, например, выполняется основная программа, и из нее вызывается подпрограмма 1. Если нам надо сохранить значения данных и внутренних регистров основной программы на время выполнения подпрограммы, мы перед вызовом подпрограммы сохраним их в стеке (запишем в стек), а после ее окончания извлечем (прочитаем) их из стека. Если же из подпрограммы 1 вызывается подпрограмма 2, то ту же самую операцию мы проделаем с данными и содержимым внутренних регистров подпрограммы 1. Понятно, что внутри подпрограммы 2 крайними в стеке (читаемыми в первую очередь) будут данные из подпрограммы 1, а данные из основной программы будут глубже. При этом в случае чтения из стека автоматически будет соблюдаться нужный порядок читаемой информации. То же самое будет и в случае, когда таких уровней вложения подпрограмм гораздо больше. То есть то, что надо хранить подольше, прячется поглубже, а то, что скоро может потребоваться – с краю.

В системе команд любого процессора для обмена информацией со стеком предусмотрены специальные команды записи в стек (*PUSH*) и чтения из стека (*POP*). В стеке можно прятать не только содержимое всех внутренних регистров процессоров, но и содержимое регистра признаков (слово состояния процессора, *PSW*). Это позволяет, например, при возвращении из подпрограммы контролировать результат последней команды, выполненной непосредственно перед вызовом этой подпрограммы. Можно также хранить в стеке и данные, для того чтобы удобнее было передавать их между программами и подпрограммами. В общем случае, чем больше область памяти, отведенная под стек, тем больше свободы у программиста и тем более сложные программы могут выполняться.

В фон-неймановской архитектуре единая область памяти используется, в том числе, и для реализации стека. При этом снижается производительность устройства, так как одновременный доступ к различным видам памяти невозможен. В частности, при выполнении команды вызова подпрограммы следующая команда выбирается после того, как в стек будет помещено содержимое программного счетчика.

В гарвардской архитектуре стековые операции производятся в специально выделенной для этой цели памяти. Это означает, что при выполнении программы вызова подпрограмм процессор с гарвардской архитектурой производит несколько действий одновременно.

Необходимо помнить, что МК обеих архитектур имеют ограниченную емкость памяти для хранения данных. Если в процессоре имеется отдельный стек и объем записанных в него данных превышает его емкость, то происходит

циклическое изменение содержимого указателя стека, и он начинает ссылаться на ранее заполненную ячейку стека. Это означает, что после слишком большого количества вызовов подпрограмм в стеке окажется неправильный адрес возврата. Если МК использует общую область памяти для размещения данных и стека, то существует опасность, что при переполнении стека произойдет запись в область данных либо будет сделана попытка записи загружаемых в стек данных в область ПЗУ.

1.4.4.5 Внешняя память

Несмотря на существующую тенденцию по переходу к закрытой архитектуре МК, в некоторых случаях возникает необходимость подключения дополнительной внешней памяти (как памяти программ, так и данных).

Если МК содержит специальные аппаратные средства для подключения внешней памяти, то эта операция производится штатным способом (как для МП).

Второй, более универсальный, способ заключается в том, чтобы использовать порты ввода/вывода для подключения внешней памяти и реализовать обращение к памяти программными средствами. Такой способ позволяет задействовать простые устройства ввода/вывода без реализации сложных шинных интерфейсов, однако приводит к снижению быстродействия системы при обращении к внешней памяти.

1.5 ПОПУЛЯРНЫЕ СЕМЕЙСТВА 8-РАЗРЯДНЫХ МК

Число различных модификаций 8-разрядных МК, представленных на мировом рынке, столь велико, что лишь одно их перечисление может занять несколько десятков страниц. Поэтому кратко охарактеризуем лишь те семейства МК, которые получили широкое распространение в России на протяжении последних десяти лет.

1.5.1 МК с ядром *MCS-51*

Начало мощному клану с ядром *MCS*-51 положила фирма *Intel*, выпустив в 1980 г. МК *8051AH* (отечественный аналог - 1816ВЕ51). Для своего времени МК *8051AH,* имевший гарвардскую архитектуру и *CISC*-набор команд, был очень сложным изделием - на кристалле размещалось 128 тыс. транзисторов. Этот микроконтроллер содержал процессорное ядро *MCS-51*, резидентные ПЗУ объемом 4 Кбайта, ОЗУ в 128 байт, 4 порта ввода/вывода, 2 таймера и асинхронный порт. Быстродействие центрального процессора *MCS-51* в МК *8051AH* по нынешним меркам было невелико. Частота внутренней шины составляла 1 МГц. Однако само ядро *MCS-51* оказалось настолько удачным, что на два десятилетия стало стандартом "де-факто" в области 8-разрядных МК.

Фирма *Intel* непрерывно совершенствовала МК с архитектурой *MCS*-51:

- частота внутренней шины в последних моделях возросла до 3 МГц;
- появились модели с объемом памяти программ 8, 16 и 32 Кбайта;
- в составе МК появились новые периферийные модули (АЦП, про-

граммируемый счетный массив, сторожевой таймер).

Одновременно ряд других фирм разработали МК, программно совместимые с *MCS-51*, обладающие современными типами памяти программ и данных (*Flash* и *EEPROM*), имеющие расширенный набор периферийных модулей, работающие в расширенном диапазоне напряжения питания. Фирма *Intel* постепенно свернула производство 8-разрядных МК. В результате основными производителями в мире 51-го семейства оказались фирмы *Philips, Infineon, Atmel, Dallas Semiconductor, Temic*. В 1999 г. фирма *Analog Devices* представила совершенно новый МК *Adu812* на основе 51-го ядра. Отличия в технических характеристиках встроенных модулей ЦАП и АЦП этого изделия столь велики, что семейство *AduSxx* было названо семейством интеллектуальных преобразователей или микроконверторами.

1.5.2 МК семейства *HC05* фирмы *Motorola*

Одновременно с первым МК семейства *MCS-51* появился первый МК популярного до настоящего времени семейства *HC05* фирмы *Motorola*. В рамках этого семейства фирма *Motorola* провозгласила и успешно реализует стратегию «заказных» МК.

Процессорное ядро построено на основе CISC-архитектуры. Многие модели этого семейства своим рождением обязаны крупным потребителям, которые заказывали конфигурацию МК под конкретную продукцию. Сейчас семейство *HC05* насчитывает около 180 различных МК, начиная с простейшего *68HC05KJ1* в корпусе *DIP16* и заканчивая 128-выводным бескорпусным *68HC05L10* со встроенным контроллером управления 960 сегментами ЖКИ.

На протяжении всего своего еще не оконченного периода жизни семейство *HC05* является сильным и успешным оппонентом семейству *MCS-51*. Выполненное на основе принстонской архитектуры в противовес *MCS-51* с гарвардской архитектурой, это семейство демонстрирует, во-первых, многообразие возможных технических решений даже для очень несложных задач управления, во-вторых, успех стратегии полного удовлетворения технических требований, пользователя без избыточности в архитектуре и производительности. Долгое пребывание семейства *HC05* на столь динамичном рынке МК определяется отнюдь не сверхбыстродействием (частота внутренней шины для большинства моделей равна 2 МГц) или уникальным набором команд. Причина успеха кроется в очень точной ориентации на различные сектора рынка массового потребления. Широчайшее разнообразие периферийных модулей при неизменном, очень простом ядре *HC05* позволяет разработчику для каждой задачи найти МК практически без избыточных ресурсов архитектуры, что обусловливает низкую стоимость изделия.

В дополнение к дешевым «заказным» МК семейства *HC05* фирма *Motorola* еще в 1980-х гг. предложила семейство универсальных и более производительных МК семейства *HC11*. Это семейство насчитывает около 40 моделей. Процессорное ядро семейства *HC11* отличается от *HC05* возможностью выполнения операций над 16-разрядными операндами, наличием дополнительных методов адресации, повышенной частотой внутренней шины (до 4 МГц).

МК семейства *HC11* выгодно отличает наличие трех типов памяти на кристалле: однократно программируемого ПЗУ программ, статического ОЗУ данных и электрически программируемого и электрически стираемого ПЗУ данных.

1.5.3 МК семейства *HC08*

В конце 1990-х годов фирма *Motorola* представила новое 8-разрядное семейство *HC08*, которое должно постепенно заменить МК семейства *HC05* и стать новым «промышленным стандартом» 8-разрядных МК фирмы. Отличительные особенности и направления развития семейства *HC08* перечислены ниже:

– Высокопроизводительное 8-разрядное АЛУ. Увеличение производительности достигается повышением частоты обмена внутренней шины до 8,0 МГц, совмещением цикла исполнения и цикла выборки следующей команды, введением специальных команд просмотра таблиц и организации циклов, расширением числа способов адресации операндов. Указанные меры позволили повысить производительность центрального процессора *HC08* в 6 раз по сравнению с процессором семейства *HC05*.

– Программная совместимость «снизу вверх» как на уровне исходного текста, так и на уровне объектных кодов с процессорным ядром семейства HC05.

– Переход к *FLASH*-технологии для ПЗУ программ пользователя. Для большинства типов МК проектируется создание двух моделей с возможностью замены «корпус в корпус». Эти МК полностью идентичны по функциональному составу и различаются только технологией занесения информации в ПЗУ программ (*maskROM* или *FLASH*).

– Библиотека периферийных модулей имеет расширенный набор контроллеров последовательного обмена. Кроме стандартных для МК фирмы *Motorola* портов асинхронного (*SCI*) и синхронного (*SPI*) обмена, разработаны контроллеры для работы в промышленных сетях с протоколом *CAN* и для перспективной шины вычислительной техники *USB*.

– Существенно улучшены возможности отладки МК, Встроенный монитор и специальный порт позволяют производить отладку прикладных программ управления непосредственно на плате конечного изделия без использования дорогостоящих схемных эмуляторов.

– МК с памятью программ типа *FLASH* позволяют реализовать режим программирования в системе, при котором прикладная программа заносится в память МК, который стационарно расположен на плате изделия. Коды программы передаются по последовательному интерфейсу от персонального компьютера.

– Специальные схемотехнические решения повышают надежность работы МК в условиях электромагнитных помех и неблагоприятной внешней среды.

1.5.4 МК фирмы *Microchip*

В конце 1980-х годов фирма *Microchip* выпустила МК *PIC16C5X*, основанные на гарвардской реализации. Эти МК основали ныне широко распространенное семейство *PIC16*. Благодаря высокой производительности, малому потреблению и низкой стоимости это семейство с *RISC*-архитектурой составило серьезную конкуренцию производимым в то время 8-разрядным МК с *CISC*-архитектурой. В основу концепции PIC была положена *RISC*-архитектура с системой простых однословных команд. Система команд базового семейства *PIC16C5x* содержит только 33 команды. Все команды, кроме команд перехода, выполняются за один машинный цикл с перекрытием по времени выборок команд из памяти и их исполнения. Производительность *P1C16C5x* при частоте тактирования в 20 МГц составляет 5 *MIPS*. В настоящее время фирма *Microchip* выпускает пять семейств МК с *RISC*-архитектурой:

– *P1C15C5x* включает недорогие контроллеры с минимальным набором периферии;

– *PIC12Cxxx* включает МК в миниатюрном 8-выводном корпусе со встроенным тактовым генератором; однако «миниатюрность» не мешает некоторым моделям этого семейства иметь встроенный модуль 8-разрядного АЦП;

– *P1C1бx/7x/8x/9x* объединяет МК с развитой периферией; в число периферийных модулей входят таймеры-счетчики с опциями захвата/сравнения, широтно-импульсные модуляторы, аналоговые компараторы, АЦП, контроллеры различных последовательных интерфейсов;

– *P1C17C4x/5xx* включает высокопроизводительные МК с расширенной системой команд и обширной периферией; МК этого семейства имеют встроенный аппаратный умножитель 8x8, выполняющий операцию умножения за один машинный цикл;

– *P1C18Cxxx* - новое семейство с оптимизированным под использование Си-компилятора *RISC*-ядром и частотой внутренней шины до 10 МГц.

1.5.5 МК семейства *AVR*

В1997 г. фирма *Atmel* представила первые МК семейства *AVR*. Семейство *AVR AT90S* объединяет мощный гарвардский RISC-процессор с раздельным доступом к памяти программ и данных, 32 регистра общего назначения и развитую систему команд. Последние версии семейства *AVR* имеют в составе АЛУ аппаратный умножитель. Базовый набор команд *AVR* содержит 120 инструкций. Большинство команд выполняется за один машинный цикл, производительность ряда моделей составляет 20 *MIPS*. Периферия *AVR* включает параллельные порты, таймеры-счетчики, различные последовательные интерфейсы, АЦП, аналоговые компараторы. МК *AVR* подразделяются на три серии:

– *tiny AVR* - МК в 8-выводном корпусе низкой стоимости;

– *classic AVR* - основная линия МК с производительностью до 16 *MIPS*, *Flash* память программ объемом до 8 Кбайт и статическим ОЗУ данных 128...512 байт;

– *mega AVR* - МК для сложных приложений, требующих большого объ-

ема памяти (*Flash ПЗУ* до 128 Кбайт), ОЗУ до 4 Кбайт, производительностью до 16 *MIPS*.

Приведенная краткая аннотация семейств 8-разрядных МК является далеко не полной, 8-разрядные МК выпускают также фирмы *ST-Microelectronics* (семейства *ST6, ST7* и *ST9*), *National Semiconductor* (семейство *COP8*), *Zilog, NEC, Mitsubishi, Hitachi, Toshiba, Scenix* и др. Продукция этих фирм постепенно появляется на российском рынке, но пока не получила широкого распространения.

1.6 МИКРОКОНТРОЛЛЕРЫ *AVR*

Фирма *ATMEL* выпускает несколько серий 8-разрядных высокопроизводительных *RISC* микроконтроллеров общего назначения, объединенных общей маркой *AVR* [4].

Окончательный выбор разработчиком той или иной микропроцессорной платформы для реализации своей задачи зависит, естественно, от большого числа разнообразных факторов, включая экономические. Но обычно первостепенным условием остается получение максимально выгодного соотношения "цена-производительность–энергопотребление", определяемого сложностью решаемой задачи. Видимо, это обстоятельство и послужило толчком к разработке в середине 1990-х нового 8-разрядного микроконтроллера.

На настоящий момент соотношение "цена – производительность – энергопотребление" для *AVR* является одним из лучших на мировом рынке 8-разрядных микроконтроллеров. Объемы продаж *AVR* в мире удваиваются ежегодно. В геометрической прогрессии растет число сторонних фирм, разрабатывающих и выпускающих разнообразные программные и аппаратные средства поддержки разработок для них. Можно считать, что *AVR* постепенно становится еще одним индустриальным стандартом среди 8-разрядных микроконтроллеров общего назначения.

Серийное производство *AVR* началось в 1996 году, а в настоящее время в серийном производстве у Atmel находятся три семейства *AVR* – "*tiny*", "*classic*" и "*mega*". Многие российские специалисты и разработчики уже по достоинству оценили высокую скорость работы и мощную систему команд *AVR*, наличие двух типов энергонезависимой памяти на одном кристалле и активно развивающуюся периферию.

Области применения *AVR* многогранны. Для семейства "*tiny*" - это интеллектуальные автомобильные датчики различного назначения, игрушки, игровые приставки, материнские платы персональных компьютеров, контроллеры защиты доступа в мобильных телефонах, зарядные устройства, детекторы дыма и пламени, бытовая техника, разнообразные инфракрасные пульты дистанционного управления. Для семейства "*classic*" - это модемы различных типов, современные зарядные устройства, изделия класса *Smart Cards* и устройства чтения для них, спутниковые навигационные системы для определения местоположения автомобилей на трассе, сложная бытовая техника, пульты дистанционного управления, сетевые карты, материнские платы компьютеров, сотовые

телефоны нового поколения а также различные и разнообразные промышленные системы контроля и управления. Для "mega" *AVR* - это аналоговые (*NMT, ETACS, AMPS*) и цифровые (*GSM, CDMA*) мобильные телефоны, принтеры и ключевые контроллеры для них, контроллеры аппаратов факсимильной связи и ксероксов, контроллеры современных дисковых накопителей, *CD-ROM* и т.д.

В линии *AVR* микроконтроллеров общего назначения постоянно появляются новые кристаллы, обновляются версии уже существующих микросхем, совершенствуется и расширяется программное обеспечение поддержки. Так, первое официальное издание – каталог *Atmel*, посвященный *AVR* - датирован маем 1997 года. В него были включены всего четыре первых *AVR* - микроконтроллера семейства *AT90S* "*classic*". Второе, существенно расширенное издание каталога вышло в августе 2004 года, и в него уже были включены три семейства *AVR* - "*tiny*", "*classic*" и "*mega*". Технические данные постоянно обновляются в электронном виде (*Data Sheet*), которые *Atmel Corp*. размещает на своей информационной странице в Интернете [5].

1.6.1 Память микроконтроллеров *AVR*

Все *AVR* имеют *Flash*-память программ, которая может быть загружена как с помощью обычного программатора, так и с помощью *SPI*-интерфейса, в том числе непосредственно на целевой плате. Число циклов перезаписи - не менее 1000. Версии кристаллов семейства "*mega*" выпуска 2001-2002 года имеют возможность самопрограммирования. Это означает, что микроконтроллер способен самостоятельно, без какого-либо внешнего программатора, изменять содержимое ячеек памяти программ. То есть, новые *AVR*, как реакция на некоторые события, могут менять алгоритмы своего функционирования и программы, заложенные в них, и далее работать уже по измененному алгоритму или новой программе.

Все *AVR* имеют также блок энергонезависимой электрически стираемой памяти данных *EEPROM*. Этот тип памяти, доступный программе микроконтроллера непосредственно в ходе ее выполнения, удобен для хранения промежуточных данных, различных констант, таблиц перекодировок, калибровочных коэффициентов и т.п. *EEPROM* также может быть загружена извне как через SPI интерфейс, так и с помощью обычного программатора. Число циклов перезаписи – не менее 100000. Два программируемых бита секретности позволяют защитить память программ и энергонезависимую память данных *EEPROM* от несанкционированного считывания. Внутренняя оперативная память *SRAM* имеется у всех *AVR* семейств "*classic*" и "*mega*". Для некоторых микроконтроллеров возможна организация подключения внешней памяти данных объемом до 64К.

1.6.2 Рабочая частота и циклы команд микроконтроллеров *AVR*

Внутренний тактовый генератор *AVR* может запускаться от нескольких источников опорной частоты (внешний генератор, внешний кварцевый резонатор, внутренняя или внешняя *RC*-цепочка). Поскольку *AVR*-микроконтроллеры

полностью статические, минимальная допустимая частота ничем не ограничена (вплоть до пошагового режима). Максимальная рабочая частота определяется конкретным типом микроконтроллера и составляет 4, 8, 14, 16 Мгц у разных образцов. Верхние границы частотного диапазона гарантируют устойчивую работу микроконтроллеров при работе во всем температурном диапазоне.

Например, у микроконтроллеров семейства *MCS51* короткая команда выполняется за 12 тактов генератора (1 машинный цикл), в течение которого процессор последовательно считывает код операции и исполняет ее. В *PIC*-контроллерах фирмы *Microchip*, где уже реализован конвейер, короткая команда выполняется в течение 8 периодов тактовой частоты (2 машинных цикла). За это время последовательно дешифрируется и считывается код операции, исполняется команда, фиксируется результат и одновременно считывается код следующей операции (одноуровневый конвейер). Поэтому в общем потоке команд одна короткая команда реализуется за 4 периода тактовой частоты или за один машинный цикл. В микроконтроллерах *AVR* тоже используется одноуровневый конвейер при обращении к памяти программ и короткая команда в общем потоке выполняется, как и в *PIC*-контроллерах, за один машинный цикл. Главное же отличие состоит в том, что этот цикл у *AVR* составляет всего один период тактовой частоты. Для сравнения, на рисунке 1.5 приведены временные диаграммы при выполнении типовой команды для различных микроконтроллерных платформ.

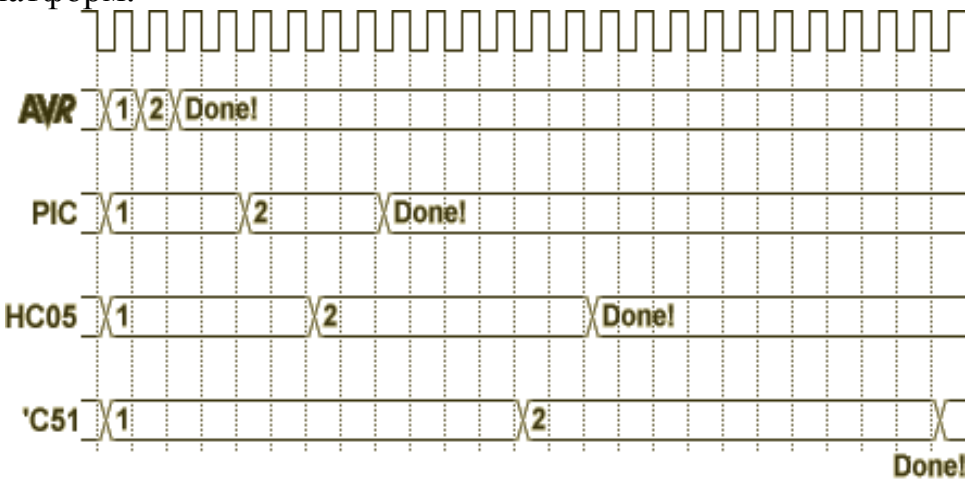


Рисунок 1.5 – Сравнительная характеристика некоторых микропроцессорных платформ

1.6.3 Таймеры микроконтроллеров *AVR*

Сторожевой (*WATCHDOG*) таймер предназначен для защиты микроконтроллера от сбоев в процессе работы. Он имеет свой собственный *RC*-генератор, работающий на частоте 1 МГц. Эта частота является приближенной и зависит, прежде всего, от величины напряжения питания микроконтроллера и от температуры. *WATCHDOG*-таймер снабжен собственным предделителем входной частоты с программируемым коэффициентом деления, что позволяет подстраивать временной интервал переполнения таймера и сброса микроконтроллера. *WATCHDOG*-таймер может быть отключен программным образом

во время работы микроконтроллера как в активном режиме, так и в любом из режимов пониженного энергопотребления. В последнем случае это приводит к значительному снижению потребляемого тока.

Микроконтроллеры *AVR* имеют в своем составе от 1 до 4 таймеров/счетчиков общего назначения с разрядностью 8 или 16 бит, которые могут работать и как таймеры от внутреннего источника опорной частоты, и как счетчики внешних событий с внешним тактированием.

Общие черты всех таймеров/счетчиков следующие.

– наличие программируемого предделителя входной частоты с различными градациями деления. Отличительной чертой является возможность работы таймеров/счетчиков на основной тактовой частоте микроконтроллера без предварительного ее понижения, что существенно повышает точность генерации временных интервалов системы;

– независимое функционирование от режима работы процессорного ядра микроконтроллера (т.е. они могут быть как считаны, так и загружены новым значением в любое время);

– возможность работы или от внутреннего источника опорной частоты, или в качестве счетчика событий. Верхний частотный порог определен в этом случае как половина основной тактовой частоты микроконтроллера. Выбор перепада внешнего источника (фронт или срез) программируется пользователем;

– наличие различных векторов прерываний для нескольких различных событий (переполнение, захват, сравнение).

1.6.4 Порты ввода-вывода

Порты ввода/вывода *AVR* имеют число независимых линий "Вход/Выход" от 3 до 53. Каждый разряд порта может быть запрограммирован на ввод или на вывод информации. Выходные драйверы обеспечивают токовую нагрузочную способность 20 мА на линию порта (втекающий ток) при максимальном значении 40 мА, что позволяет, например, непосредственно подключать к микроконтроллеру светодиоды и биполярные транзисторы. Общая токовая нагрузка на все линии одного порта не должна превышать 80 мА (все значения приведены для напряжения питания 5 В).

Интересная архитектурная особенность построения портов ввода/вывода у AVR заключается в том, что для каждого физического вывода существует 3 бита контроля/управления, а не 2, как у распространенных 8-разрядных микроконтроллеров (Intel, Microchip, Motorola и т.д.).

1.6.5 Ввод аналоговых данных

Аналоговый компаратор входит в состав большинства микроконтроллеров *AVR*. Типовое напряжение смещения равно 10 мВ, время задержки распространения составляет 500 нс и зависит от напряжения питания микроконтроллера. Так, например, при напряжении питания 2,7 Вольт оно равно 750 нс. Аналоговый компаратор имеет свой собственный вектор прерывания в общей си-

стеме прерываний микроконтроллера. При этом тип перепада, вызывающий запрос на прерывание при срабатывании компаратора, может быть запрограммирован пользователем как фронт, срез или переключение. Логический выход компаратора может быть программным образом подключен ко входу одного из 16-разрядных таймеров/счетчиков, работающего в режиме захвата. Это дает возможность измерять длительность аналоговых сигналов а также максимально просто реализовывать АЦП двухтактного интегрирования.

Аналого-цифровой преобразователь построен по классической схеме последовательных приближений с устройством выборки/хранения (УВХ). Каждый из аналоговых входов может быть соединен со входом УВХ через аналоговый мультиплексор. Устройство выборки/хранения имеет свой собственный усилитель, гарантирующий, что измеряемый аналоговый сигнал будет стабильным в течение всего времени преобразования. Разрядность АЦП составляет 10 бит при нормируемой погрешности +/- 2 младших значащих разряда (МЗР). АЦП может работать в двух режимах - однократное преобразование по любому выбранному каналу и последовательный циклический опрос всех каналов. Время преобразования выбирается программно с помощью установки коэффициента деления частоты специального предделителя, входящего в состав блока АЦП. Оно составляет 70...280 мкс для ATmega103 и 65...260 мкс для всех остальных микроконтроллеров, имеющих в своем составе АЦП. Важной особенностью аналого-цифрового преобразователя является функция подавления шума при преобразовании. Пользователь имеет возможность, выполнив короткий ряд программных операций, запустить АЦП в то время, когда центральный процессор находится в одном из режимов пониженного энергопотребления. При этом на точность преобразования не будут оказывать влияние помехи, возникающие при работе процессорного ядра.

1.6.6 Пониженное энергопотребление

AVR - микроконтроллеры могут быть переведены программным путем в один из шести режимов пониженного энергопотребления. Для разных семейств *AVR* и разных микроконтроллеров в пределах каждого семейства изменяются количество и сочетание доступных режимов пониженного энергопотребления.

Режим холостого хода (*IDLE*), в котором прекращает работу только процессор и фиксируется содержимое памяти данных, а внутренний генератор синхросигналов, таймеры, система прерываний и *WATCHDOG*-таймер продолжают функционировать.

Режим микропотребления (*Power Down*), в котором сохраняется содержимое регистрового файла, но останавливается внутренний генератор синхросигналов. Выход из *Power Down* возможен либо по общему сбросу микроконтроллера, либо по сигналу (уровень) от внешнего источника прерывания. При включенном *WATCHDOG*-таймере ток потребления в этом режиме составляет около 60...80 мкА, а при выключенном - менее 1 мкА для всех типов *AVR*. Вышеприведенные значения справедливы для величины питающего напряжения 5 В.

Режим сохранения энергии (*Power Save*), который реализован только у тех *AVR*, которые имеют в своем составе систему реального времени. В основном, режим Power Save идентичен Power Down, но здесь допускается независимая работа дополнительного таймера/счетчика *RTC*. Выход из режима Power Save возможен по прерыванию, вызванному или переполнением таймера/счетчика *RTC*, или срабатыванием блока сравнения этого счетчика. Ток потребления в этом режиме составляет 6...10 мкА при напряжении питания 5 В на частоте 32,768 кГц.

Режим подавления шума при работе аналого-цифрового преобразователя (*ADC Noise Reduction*). Как уже отмечалось, в этом режиме останавливается процессорное ядро, но разрешена работа АЦП, двухпроводного интерфейса I^2C и сторожевого таймера.

Основной режим ожидания (*Standby*). Идентичен режиму *Power Down*, но здесь работа тактового генератора не прекращается. Это гарантирует быстрый выход микроконтроллера из режима ожидания всего за 6 тактов генератора.

Дополнительный режим ожидания (*Extended Standby*). Идентичен режиму *Power Save*, но здесь работа тактового генератора тоже не прекращается. Это гарантирует быстрый выход микроконтроллера из режима ожидания всего за 6 тактов генератора.

1.6.7 Питающее напряжение

AVR функционируют в широком диапазоне питающих напряжений от 1,8 до 6,0 Вольт. Энергопотребление в активном режиме зависит от величины напряжения питания, от частоты, на которой работает *AVR* и от конкретного типа микроконтроллера. Подробные спецификации обычно приводятся в оригинальной технической документации *Atmel Corp*. Температурные диапазоны работы микроконтроллеров *AVR* - коммерческий (0C...70C), индустриальный (-40C...+85C), в автомобильный (-40C...+125C) и военный (-55C...+125C).

1.6.8 Программная модель

С точки зрения программиста *AVR* представляет собой 8-разрядный RISC микроконтроллер, имеющий быстрый Гарвардский процессор, память программ, память данных, порты ввода/вывода и различные интерфейсные схемы. Структурная схема микроконтроллера приведена на рисунке 1.10. Гарвардская архитектура *AVR* реализует полное логическое и физическое разделение не только адресных пространств, но и информационных шин для обращения к памяти программ и к памяти данных, причем способы адресации и доступа к этим массивам памяти также различны. Подобное построение уже ближе к структуре цифровых сигнальных процессоров и обеспечивает существенное повышение производительности. Центральный процессор работает одновременно как с памятью программ, так и с памятью данных; разрядность шины памяти программ расширена до 16 бит. Следующим шагом на пути увеличения быстродействия

AVR является использование технологии конвейеризации, вследствие чего цикл "выборка – исполнение" команды заметно сокращен.

Следующая отличительная черта архитектуры микроконтроллеров *AVR* - регистровый файл быстрого доступа. Каждый из 32-х регистров общего назначения длиной 1 байт непосредственно связан с арифметико-логическим устройством (*ALU*) процессора. Другими словами, в *AVR* существует 32 регистра – аккумулятора (сравните, например, с *MCS51*). Это обстоятельство позволяет в сочетании с конвейерной обработкой выполнять одну операцию в *ALU* за один машинный цикл. Так, два операнда извлекаются из регистрового файла, выполняется команда и результат записывается обратно в регистровый файл в течение только одного машинного цикла.

Шесть из 32-х регистров файла (с 26-го по 31-й) могут использоваться как три 16-разрядных указателя адреса при косвенной адресации данных (рисунок 1.6). Один из этих указателей (*Z Pointer*) применяется также для доступа к данным, записанным в памяти программ микроконтроллера. Использование трех 16-битных указателей (*X, Y* и *Z Pointers*) существенно повышает скорость пересылки данных при работе прикладной программы.

Регистровый файл занимает младшие 32 байта в общем адресном пространстве *SRAM AVR*. Такое архитектурное решение позволяет получать доступ к быстрой "регистровой" оперативной памяти микроконтроллера двумя путями - непосредственной адресацией в коде команды к любой ячейке и другими способами адресации ячеек *SRAM*.

В технической документации фирмы *Atmel* это полезное свойство носит название "быстрое контекстное переключение" и является еще одной отличительной особенностью архитектуры *AVR*, повышающей эффективность работы микроконтроллера и его производительность.

Особенно заметно данное преимущество при реализации процедур целочисленной 16-битной арифметики, когда исключаются многократные пересылки между различными ячейками памяти данных при обработке арифметических операндов в *ALU*.

7 0	7 0	Адрес
RO	RO	$00
R1	R1	$01
R2	R2	$02
: .	: .	
R13	R13	$0D
R14	R14	$OE
R15	R15	$OF
R16	R16	$10
R17	R17	$11
. . .	. .	
R26	R26	$1A регистр X, мл байт
R27	R27	$1B регистр X, ст. байт
R28	R28	$1C регистр Y, мл байт
R29	R29	$1D регистр Y, ст. байт
R30 (регистр Z)	R30	$1E регистр Z, мл байт
R31	R31	$1F регистр Z, ст.байт
AT90S1200	*Ост. модели*	

Рисунок 1.6 – Структура регистрового файла

1.6.8.1 Регистры общего назначения

Все регистры общего назначения объединены в файл, структура которого показана на рисунке 1.6.

Как уже было сказано, в микроконтроллерах *AVR* все 32 РОН непосредственно доступны АЛУ в отличие от микроконтроллеров других фирм, в которых имеется только один такой регистр – рабочий регистр *W* (аккумулятор). Благодаря этому любой РОН может использоваться во всех командах и как операнд-источник и как операнд-приемник. Исключение составляют лишь пять арифметических и логических команд, выполняющих действия между константой и регистром (*SBCI, SUBI, CPI, ANDI, ORI*), а также команда загрузки константы в регистр (*LDI*). Эти команды могут обращаться только ко второй половине регистров (*R16...R31*). Ряд регистров общего назначения используется в качестве указателей при косвенной адресации памяти данных. В модели *AT90S1200* таким регистром является регистр *R30* (регистр *Z*). Поскольку объем адресуемой памяти данных этой модели составляет всего 96 байт, для хранения адреса достаточно одного 8-разрядного регистра. Во всех других моделях для косвенной адресации используются три 16-разрядных регистра (регистры *X, Y* и *Z*), каждый из которых получается объединением двух РОН (рисунок 1.7).

Таблица 1.1 – Дополнительные символические имена индексных регистров

Регистр	Символическое имя
R26	*XL*
R27	*XH*
R28	*YL*
R29	*YH*
R30	*ZL*
R31	*ZH*

Для *AT90S1200* определены только два регистра-указателя: *R30 — ZL и R31 — ZH.*

Как показано на рисунке 1.6, каждый регистр файла имеет свой собственный адрес в пространстве памяти данных (кроме *AT90S1200*). Поэтому к ним можно обращаться как к памяти, несмотря на то, что физически эти регистры не являются ячейками ОЗУ.

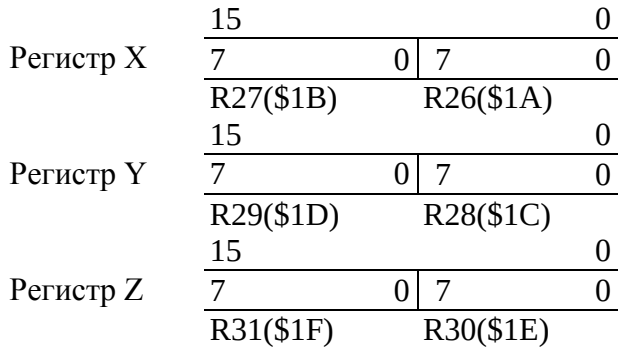


Рисунок 1.7 – Регистры-указатели X, Y и Z

1.6.8.2 Регистры ввода/вывода

Регистры ввода/вывода располагаются в так называемом пространстве ввода/вывода размером 64 байта. Все РВВ можно разделить на две группы: служебные регистры микроконтроллера и регистры, относящиеся к периферийным устройствам (в т.ч. порты ввода/вывода). Размер каждого регистра – 8 разрядов.

Распределение адресов пространства ввода/вывода зависит от конкретной модели микроконтроллера, т.к. разные модели имеют различный состав периферийных устройств и, соответственно, разное количество регистров. Размещение РВВ в адресном пространстве ввода/вывода для всех моделей семейства приведено в таблицах П.1-П.4.

Общее замечание к таблицам: если адрес в таблице не указан, это означает, что для данной модели он зарезервирован, и запись по этому адресу запрещена (крайне не рекомендуется).

К любому регистру ввода/вывода можно обратиться с помощью команд *IN* и *OUT*, выполняющих пересылку данных между одним из 32 РОН и пространством ввода/вывода. Кроме того, имеются 4 команды поразрядного доступа, использующие в качестве операндов регистры ввода/вывода: команды установки/сброса отдельного разряда (*SBI* и *CBI*) и команды проверки состояния отдельного разряда (*SBIS* и *SBIC*). Обратите внимание, что эти команды могут обращаться только к 1-й половине регистров ввода/вывода (адреса *$00...$1F*).

Так же, как и к РОН, к регистрам ввода/вывода можно обращаться двумя способами: как собственно к регистрам (с помощью команд *IN* и *OUT*) и как к ячейкам ОЗУ (кроме *AT90S1200*). В первом случае используются адреса РВВ, принадлежащие пространству ввода/вывода *($00...$3F*). Во втором случае адрес РВВ необходимо увеличить на *$20* (в таблицах при указании адресов РВВ в скобках указываются соответствующие им адреса ячеек ОЗУ).

Все служебные регистры перечислены в таблице 1.2, знак "+" в таблице означает, что тот или иной регистр присутствует в данной модели микроконтроллера. Обратите внимание, что адреса служебных регистров не меняются от модели к модели (т.е. регистр SREG всегда расположен по адресу $3F ($5F), GIMSK — по адресу $3B ($5B) и т.д.).

Таблица 1.2 – Служебные регистры микроконтроллеров семейства Classic

Название регистра	Адрес	*AT90S1200*	*AT90S2313*	*AT90S/LS2323*	*AT90S/LS2343*	*AT90S/LS2333*	*AT90S/LS4433*	*AT90S/LS4434*	*AT90S/LS8535*	*AT90S4414*	*AT90S8515*	*AT90C8534*
SREG	*$3F ($5F)*	+	+	+	+	+	+	+	+	+	+	+
SPH	*$3E ($5E)*							+	+	+	+	+
SPL	*$3D ($5D)*		+	+	+	+	+	+	+	+	+	+
G1MSK	*$3B ($5B)*	+	+	+	+	+	+	+	+	+	+	+
GIFR	*$3A ($5A)*		+	+	+	+	+	+	+	+	+	+
TIMSK	*$39 ($59)*	+	+	+	+	+	+	+	+	+	+	+
TIFR	*$38 ($58)*	+	+	+	+	+	+	+	+	+	+	+
MCUCR	*$35 ($55)*	+	+	+	+	+	+	+	+	+	+	+
MCUSR	*$34 ($54)*			+	+			+	+			

1.6.8.3 Регистр состояния *SREG*

Регистр состояния располагается по адресу *$3F ($5F)*. Этот регистр представляет собой набор флагов, показывающих текущее состояние микроконтроллера. Эти флаги автоматически устанавливаются в "1" или в "0" при наступлении определенных событий (в соответствии с результатом выполнения команд). Все разряды этого регистра доступны как для чтения, так и для записи в любой момент времени. После сброса микроконтроллера все разряды реги-

стра сбрасываются в "0". Содержимое этого регистра показано на рисунке 1.8, а его описание приведено в таблице 1.3.

	7	6	5	4	3	2	1	0
	I	*T*	*H*	*S*	*V*	*N*	*Z*	*C*
Чтение(R)/Запись(*W*)	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начальное значение	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Рисунок 1.8 – Регистр состояния SREG

Таблица 1.3 – Регистр состояния SREG

Разряд	Название	Описание
7	*I*	Общее разрешение прерываний. Для разрешения прерываний этот флаг должен быть установлен в "1". Разрешение/запрещение отдельных прерываний производится установкой или сбросом соответствующих разрядов регистров масок прерываний. Если флаг сброшен (0), то прерывания запрещены независимо от состояния этих регистров. Флаг сбрасывается аппаратно после входа в прерывание и восстанавливается командой *RETI* для разрешения обработки следующих прерываний
6	*T*	Хранение копируемого бита. Этот разряд регистра используется в качестве источника или приемника командами копирования битов *BLD (Bit LoaD)* и *BST (Bit STore)*. Заданный разряд любого РОН может быть скопирован в этот разряд командой *BST* или установлен в соответствии с содержимым данного разряда командой *BLD*
5	*H*	Флаг половинного переноса. Этот флаг устанавливается в "1", если имел место перенос из младшей половины байта (из 3-го разряда в 4-й) или заем из старшей половины байта при выполнении некоторых арифметических операций
4	*S*	Флаг знака. Этот флаг равен результату операции "Исключающее ИЛИ" *(XOR)* между флагами *N* (отрицательный результат) и *V* (переполнение числа в дополнительном коде) Соответственно этот флаг устанавливается в "1", если результат выполнения арифметической операции меньше нуля
3	*V*	Флаг переполнения дополнительного кода. Этот флаг устанавливается в "1" при переполнении разрядной сетки знакового результата. Используется при работе со знаковыми числами (представленными в дополнительном коде). Более подробно см. описание системы команд
2	*N*	Флаг отрицательного значения. Этот флаг устанавливается в "1", если старший разряд (7-й) результата операции равен "1". В противном случае флаг равен "0"
1	*Z*	Флаг нуля. Этот флаг устанавливается в "1", если результат выполнения операции равен нулю
0	*C*	Флаг переноса. Этот флаг устанавливается в "1", если в результате выполнения операции произошел выход за границы байта

1.6.8.4 SP (указатель стека)

В моделях, имеющих объем ОЗУ до 128 байт (адресное пространство ОЗУ – *$000...$ODF*), указатель стека реализован на одном регистре *SPL*, расположенном по адресу *$3D ($5D)*. В остальных моделях указатель стека реализован на паре регистров *SPH:SPL*, расположенных по адресам *$3E ($5E)* и *$3D ($5D)* соответственно. Причем для всех моделей, кроме *AT9084414* и *AT90S8515*, в ре-

гистре *SPH* (старший байт указателя стека) используются только 1 (объем ОЗУ – 256 байт) или 2 (объем ОЗУ – 512 байт) младших разряда, остальные разряды доступны только для чтения и содержат "0". В моделях *AT90S4414* и *AT90S8515* оба регистра *SPH:SPL* используются полностью, т.к. максимальный объем памяти в этих моделях равен 64 Кбайт. Все используемые разряды регистров доступны как для чтения, так и для записи в любой момент времени. После сброса микроконтроллера содержимое регистров равно 0, поэтому в самом начале программы указатель стека необходимо проинициализировать каким-либо значением (как правило, это наибольший для конкретного микроконтроллера адрес памяти данных).

1.6.8.5 Регистры управления прерываниями

Четыре регистра предназначены для управления внешними прерываниями (регистры *GIMSK* и *GIFR*) и прерываниями от таймеров (регистры *TIMSK* и *TIFR*). Регистры масок *GIMSK* (общий регистр маски прерываний) и *TIMSK* (регистр маски прерываний от таймеров) используются при разрешения/запрещения отдельных прерываний, а регистры флагов *GIFR* (общий регистр флагов прерываний) и *TIFR* (регистр флагов прерываний от таймеров) содержат флаги, показывающие, произошло или нет соответствующее прерывание.

1.6.8.6 *MCUCR* (регистр управления микроконтроллером)

Регистр управления микроконтроллером (рисунок 1.9, таблицы П.5-П.7) расположен по адресу *$35 ($55)*. Этот регистр содержит ряд флагов, используемых для общего управления микроконтроллером. Состав флагов, размещенных в регистре MCUCR, несколько меняется от модели к модели, соответственно в некоторых моделях некоторые разряды не используются. Неиспользуемые разряды регистра доступны только для чтения и содержат "0". Все используемые разряды регистра доступны как для чтения, так и для записи в любой момент времени. После сброса микроконтроллера во всех разрядах регистра записано "0".

Разряды	7	6	5	4	3	2	1	0	Модели
	-	-	*SE*	*SM*	-	-	*ISC01*	*ISC00*	*AT90S1200*
Чтение(*R*)/Запись(*W*)	*R*	*R*	*R/W*	*R/W*	*R*	*R*	*R/W*	*R/W*	*AT90S/LS2323*
Начальное значение	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*AT90S/LS2343*
	-	-	*SE*	*SM*	*ISC11*	*ISC10*	*ISC01*	*ISCOO*	*AT90S2313*
Чтение(*R*)/Запись(*W*)	*R*	*R*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*AT90S/LS2333*
Начальное значение	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*AT90S/LS4433*
	SRE	*SRW*	*SE*	*SM*	*ISC11*	*ISC10*	*ISC01*	*ISCOO*	*AT90S4414*
Чтение(*R*)/Запись(*W*)	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*AT90S8515*
Начальное значение	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	
	-	*SE*	*SM1*	*SM0*	*ISC11*	*ISC10*	*ISC01*	*ISCOO*	*AT90S/LS4434*
Чтение(*R*)/Запись(*W*)	*R*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*AT90S/LS8535*
Начальное значение	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	
	-	-	*SE*	*SM*	-	*ISC1*	-	*ISCO*	

Чтение(*R*)/Запись(*W*)	*R*	*R*	*R/W*	*R/W*	*R*	*R/W*	*R*	*R/W*	*AT90S8534*
Начальное значение	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	

Рисунок 1.9 – Регистр управления микроконтроллером *MCUCR*

1.6.8.7 MCUSR (регистр состояния микроконтроллера)

Регистр управления микроконтроллером расположен по адресу *$34 ($54).* Этот регистр содержит флаги, состояние которых позволяет определить причину, по которой произошел сброс микроконтроллера.

1.6.8.8 4. Способы адресации памяти данных

Все микроконтроллеры *AVR* семейства *Classic*, за исключением модели *AT90S1200*, поддерживают 8 способов адресации для доступа к различным областям памяти данных (РОН, РВВ, ОЗУ). Модель *AT90S1200* в связи с отсутствием у нее встроенного ОЗУ и из-за наличия единственного индексного регистра поддерживает только 4 способа адресации из восьми. В действительности способов адресации всего два: прямая адресация и косвенная. Однако каждый способ адресации имеет несколько разновидностей в зависимости от того, к какой области памяти производится обращение (для прямой адресации) или какие дополнительные действия выполняются над индексным регистром (для косвенной адресации).

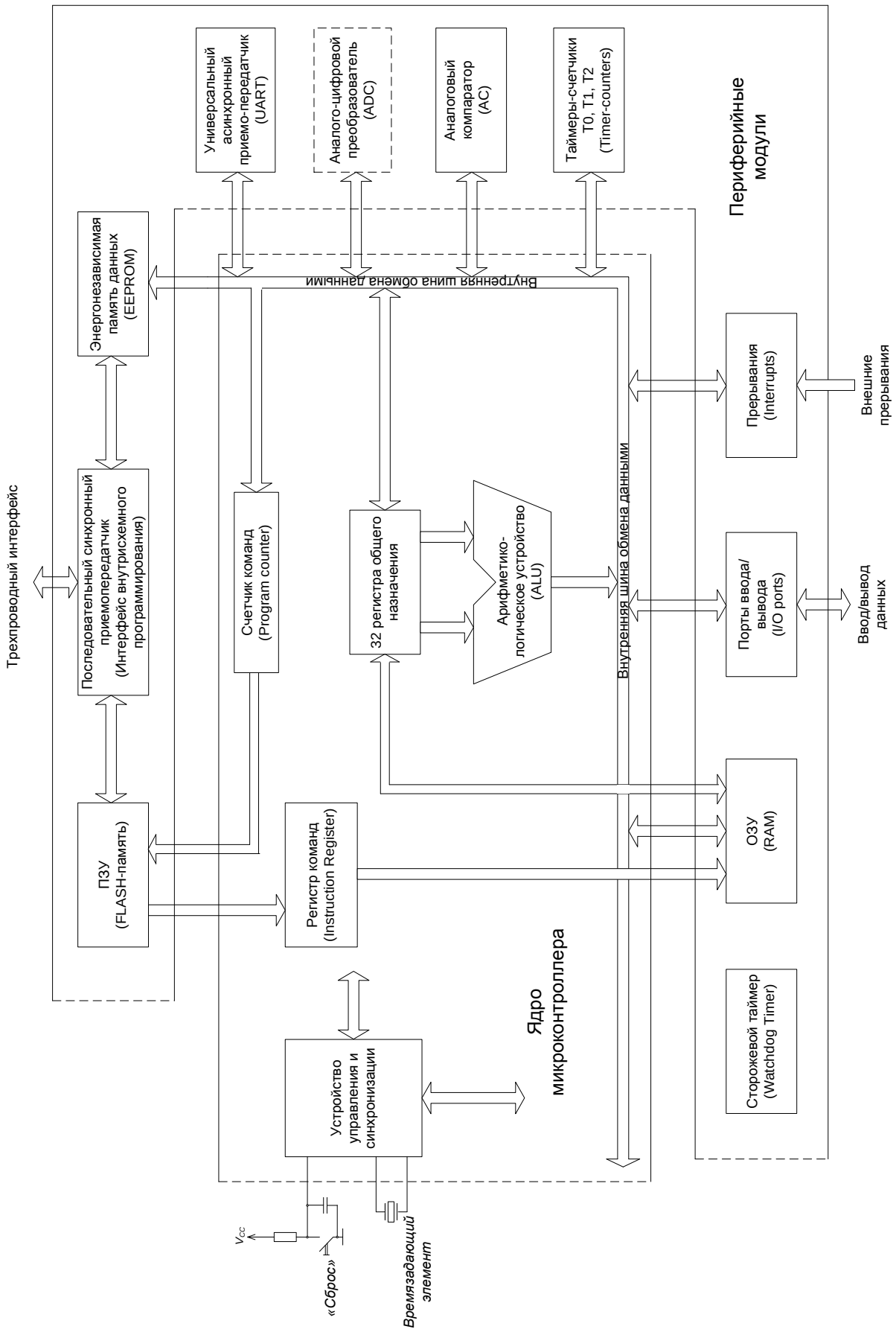
1.6.9 Система команд

Система команд *AVR* весьма развита и насчитывает до 133 различных инструкций. Почти все команды имеют фиксированную длину в одно слово (16 бит), что позволяет в большинстве случаев объединять в одной команде и код операции, и операнд(ы).

Лишь немногие команды имеют размер в 2 слова (32 бит) и относятся к группе команд вызова процедуры *CALL*, длинных переходов в пределах всего адресного пространства *JMP*, возврата из подпрограмм *RET* и команд работы с памятью программ *LPM.*

Различают пять групп команд *AVR*: условного ветвления, безусловного ветвления, арифметические и логические операции, команды пересылки данных, команды работы с битами. В последних версиях кристаллов *AVR* семейства "*mega*" реализована функция аппаратного умножения, что придает новым микроконтроллерам еще больше привлекательности с точки зрения разработчика.

По разнообразию и количеству реализованных инструкций *AVR* больше похожи на *CISC*, чем на *RISC* процессоры. Например, у *PIC*-контроллеров система команд насчитывает до 75 различных инструкций, а у *MCS51* она составляет 111. В целом, прогрессивная *RISC* архитектура *AVR* в сочетании с наличием регистрового файла и расширенной системы команд позволяет в короткие сроки создавать работоспособные программы с эффективным кодом как по компактности реализации, так и по скорости выполнения.


Универсальный асинхронный приемо-передатчик (UART)
Аналого-цифровой преобразователь (ADC)
Аналоговый компаратор (AC)
Таймеры-счетчики T0, T1, T2 (Timer-counters)
Периферийные модули
Энергонезависимая память данных (EEPROM)
Внутренняя шина обмена данными
Трехпроводный интерфейс
Последовательный синхронный приемопередатчик (Интерфейс внутрисхемного программирования)
Счетчик команд (Program counter)
32 регистра общего назначения
Арифметико-логическое устройство (ALU)
Прерывания (Interrupts)
Внешние прерывания
Порты ввода/вывода (I/O ports)
Ввод/вывод данных
ПЗУ (FLASH-память)
Регистр команд (Instruction Register)
ОЗУ (RAM)
Ядро микроконтроллера
Устройство управления и синхронизации
Сторожевой таймер (Watchdog Timer)
V_{CC}
«Сброс»
Времязадающий элемент

Рис

2 СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ AVR-КОНТРОЛЛЕРОВ

2.1 Ассемблер

Как известно, наиболее эффективные программы получаются при использовании языка Ассемблер. Стоит, правда, отметить, что при этом также увеличивается сложность и время разработки программы. Для микроконтроллеров семейства *AVR* имеется свободно распространяемый транслятор ассемблера — *wavrasm.*

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

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

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

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

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

Строка исходного текста может иметь один из следующих видов:

[*метка:*] *директива* [*аргументы директивы*] [комментарий]
[*метка:*] *мнемоника команды* [*аргументы команды*] [комментарий]
Комментарий
Пустая строка

Комментарии всегда начинаются с символа ";". Элементы, заключенные в квадратные скобки, могут отсутствовать. Текст, расположенный после символа «точка с запятой» до конца строки, полностью игнорируется ассемблером.

Примеры записи строк:

```
label1:   .EQU var1=100      ; Директива определения символьного
                             ; имени var1, эквивалентного записи "100"
```

```
        .EQU var2=200      ; Определение имени var2, соответствующего "200"
test:   rjmp test          ;Бесконечный цикл (мнемоника команды МК)
                           ; Пустая строка
```

Не имеет значение, в каких местах расположены метки, команды ассемблера и директивы, важен только их порядок.

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

2.1.1 Команды микроконтроллера

Команды микроконтроллеров семейства *AVR* делятся на группы:

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

Краткое описание команд микроконтроллера можно найти в [6].

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

2.1.2 Директивы транслятора ассемблера

Транслятор ассемблера поддерживает достаточно много директив [7]. Директивы не транслируются в программу для микроконтроллера. Они используются для указания транслятору ассемблера данных о расположении программы в памяти микроконтроллера, определения макросов и т. д. Все директивы должны начинаться с точки.

В таблице 2.1 приведен перечень директив транслятора ассемблера.

2.2 Среда разработки приложений – *AVR Studio*

AVR Studio - это интегрированная отладочная среда разработки приложений (IDE) для микроконтроллеров, представленная компанией *Atmel* для свободного использования [5]. В данной разработке используется *AVR Studio* версии 4.10. Ожидается появление версии 4.11.

IDE AVR Studio содержит:

- средства создания и управления проектом;
- редактор кода на языке ассемблер;
- транслятор языка ассемблера (*Atmel AVR macroassembler*);
- отладчик (*Debugger*);
- программное обеспечение верхнего уровня для поддержки внутрис-

хемного программирования (*In-System Programming, ISP*) с использованием стандартных отладочных средств *Atmel AVR*.

Таблица 2.1 – Перечень директив транслятора ассемблера

Обозначение директивы	Описание директивы
.BYTE	резервирует 1 байт для использования в качестве переменной
.CSEG	сегмент программ
.DB	определяет байт-константу
.DEF	определяет символическое имя для регистра
.DEVICE	задает тип целевого микроконтроллера
.DSEG	сегмент данных
.DW	определяет слово-константу
.ENDMACRO	конец определения макроса
.EQU	сопоставляет символьному имени арифметическое выражение
.ESEG	сегмент EEPROM
.EXIT	выйти из файла (конец текста программы)
.INCLUDE	загрузить исходный текст из другого файла
.LIST	включить генерацию листинга
.LISTMAC	включить печать содержимого макросов в листинге
.MACRO	начать определение макроса
.NOLIST	выключить генерацию листинга
.ORG	установить расположение
.SET	сопоставить символу выражение

Работа с *AVR Studio* начинается с создания проекта. При создании проекта необходимо указать используемый микроконтроллер и платформу, на которой будет производиться отладка программы.

Написание программы производится в окне редактора текста программы (рисунок 2.1). Для использования символических имен регистров специального назначения вместо их адресов, необходимо подключить (директива .include) к проекту файл определения регистров специального назначения (например, *m16def.inc* для *ATmega16*). Включаемые файлы входят в прикладное программное обеспечение *AVR Studio* и при инсталляции помещаются в папку *Appnotes* в директории установки *AVR Studio*.

Примеры программ доступны в большом количестве в качестве приложений к руководствам по применению микроконтроллеров *AVR* (см. раздел *Application Notes* на сайте [8]).

Написание программы в *AVR Studio* производится на языке ассемблер. Система команд микроконтроллера описана в упомянутых выше руководствах по применению в документе *AVR Instruction Set* либо в файле справки, встроенном в *AVR Studio* (меню *Help\AVR Tools User Guide\AVR Assembler*), в котором содержатся достаточно подробные комментарии к каждой команде.

Последние версии *AVR Studio* содержат тестовую версию *AVR* ассемблера второй версии, который в дополнение к стандартному ассемблеру поддерживает новые директивы ассемблера, Си - подобные директивы препроцессору, со-

здание переменных определенного типа. Более подробную информацию можно найти в файле справки.

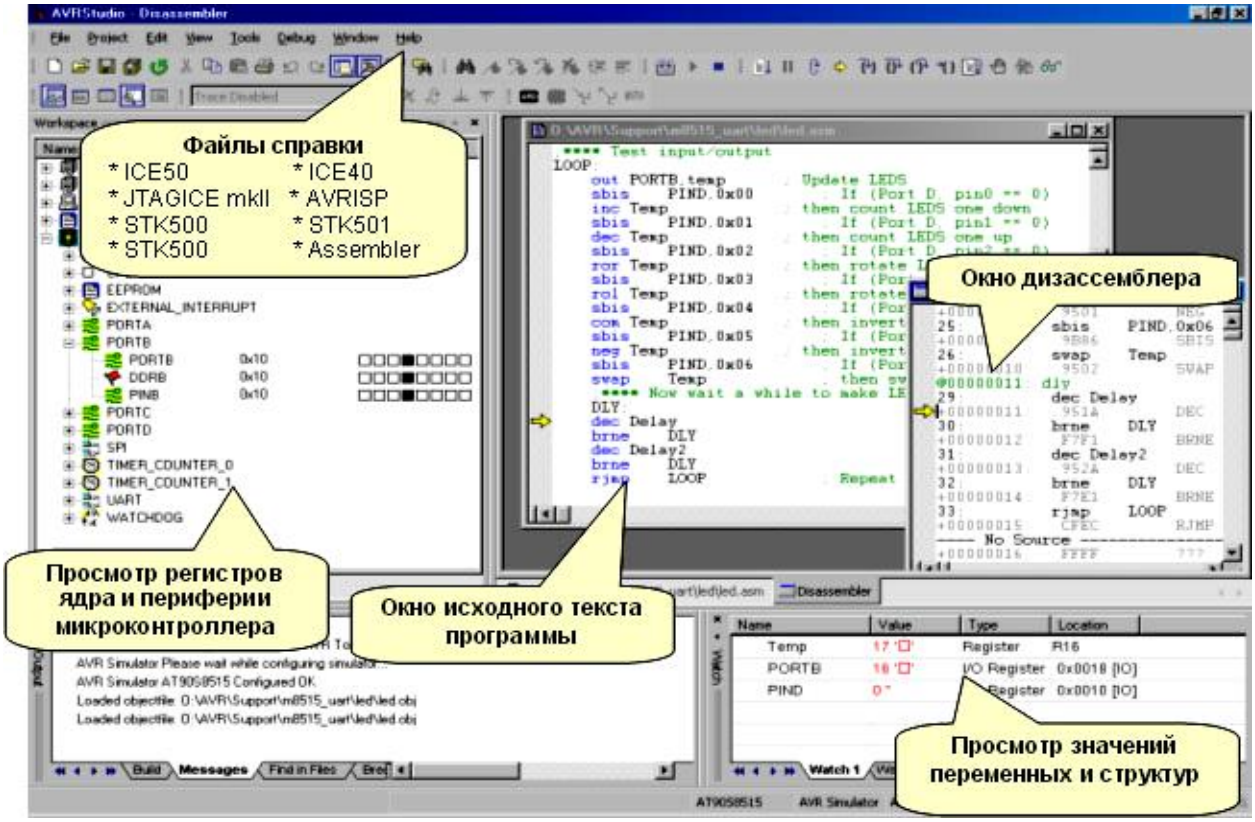


Рисунок 2.1 – Интерфейс интегрированной среды разработки AVR Studio

Перед трансляцией программы можно задать установки проекта (меню *Project\AVR Assembler Setup* показано на рисунке 2.2), указать необходимый формат выходного файла. Там же возможно установить использование *AVR* ассемблера 2-ой версии. Так как вторая версия ассемблера проходит стадию тестирования, то по умолчанию он отключен. Если не требуется каких-либо особых настроек, то можно использовать установки по умолчанию.

В результате трансляции создается выходной файл в указанном формате. Если исходный ассемблерный текст содержал сегмент энергонезависимых данных (объявленный директивой *.eseg*), то при трансляции будет создан также файл с расширением *.eep*. Этот файл содержит данные для внутренней *EEPROM* микроконтроллера и имеет тот же формат, что и выходной файл.

Если в результате трансляции не выдается сообщений об ошибках, можно приступать к отладке проекта.

Отладчик *AVR Studio* поддерживает все типы микроконтроллеров *AVR* и имеет два режима работы: режим программной симуляции и режим управления различными типами внутрисхемных эмуляторов (*In-Circuit Emulators*) производства фирмы *Atmel*. Важно отметить, что интерфейс пользователя не изменяется в зависимости от выбранного режима отладки.

Отладочная среда поддерживает выполнение программ, как в виде ассемблерного текста, так и в виде исходного текста языка Си, загруженного в объектном коде.

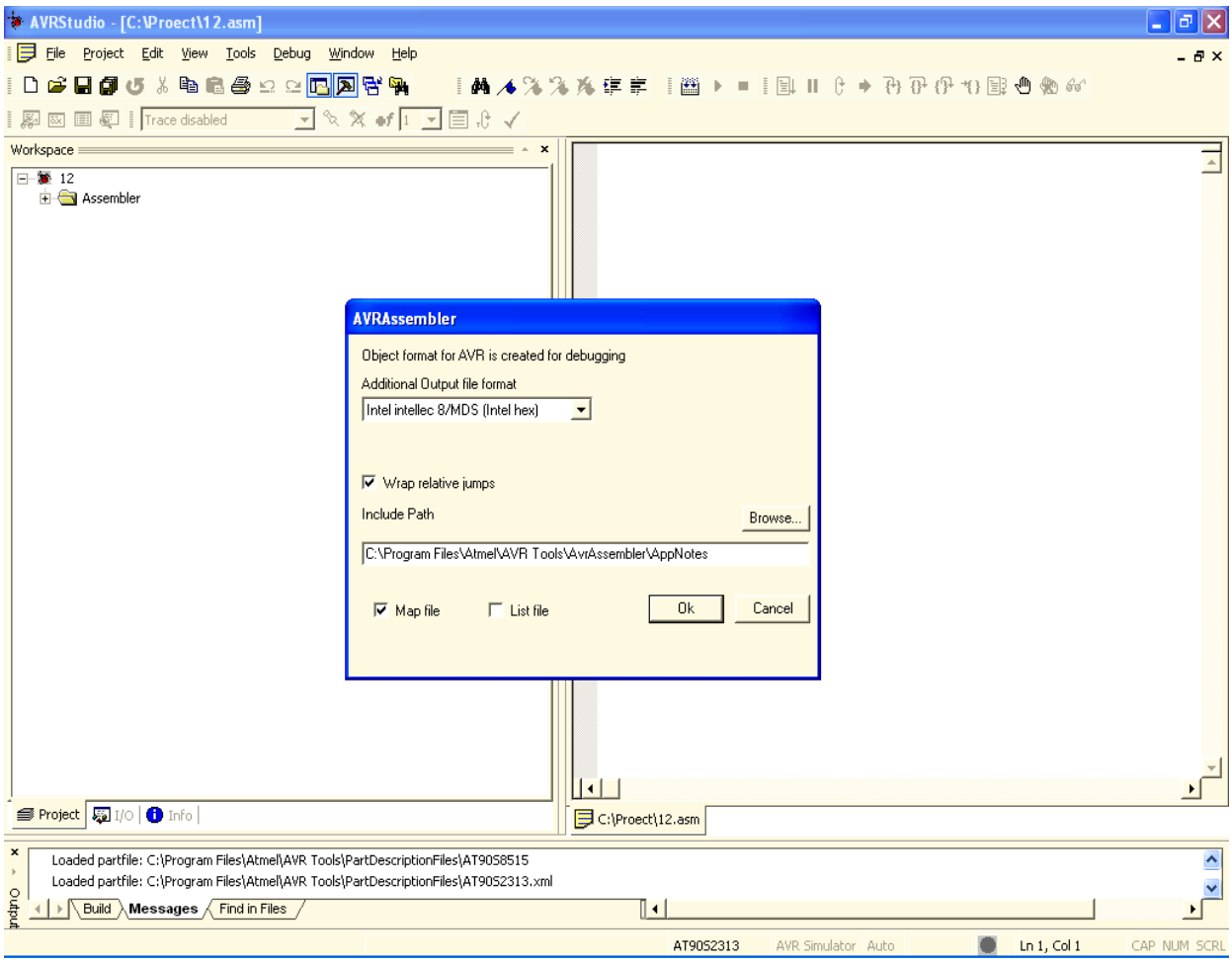


Рисунок 2.2 – Задание установок проекта

Объектный файл может быть загружен в различных форматах, а именно:

– в формате *UBROF* (расширение файла .d90). Формат *UBROF* - это патентованный формат компании IAR. На текущий момент среда разработки *AVR* Studio поддерживает все версии формата до восьмой версии (*UBROF5/6/7/8*).

– в формате *ELF/DWARF* (расширение файла *.elf*). Формат разработан сообществом открытого программного обеспечения для поддержки бесплатно распространяемых компиляторов (*GNU GCC*, для микроконтроллеров *AVR* последние версии данного компилятора имеют название *WinAVR*). На текущий момент среда разработки AVR Studio поддерживает формат версии *DWARF2*.

– в формате *AVR COFF* (расширение файла *.cof*). Это открытый формат, введенный в среду разработки *AVR Studio* для поддержки сторонних разработчиков компиляторов. Отладочный код в данном формате может быть сформирован с использованием таких компиляторов как, например, *CodeVision AVR*.

– в формате *Extended Intel HEX* (расширение *.hex*). Данный формат не приспособлен для отладки и используется для отладки в крайнем случае.

После загрузки объектного кода производится выбор отладочной платформы и используемого микроконтроллера (рисунок 2.3). При отладке с использованием внутрисхемных эмуляторов, микроконтроллеры, поддержка которых не осуществляется, автоматически подсвечиваются серым. После загрузки проекта система готова к старту отладки.

Управление отладкой производится командами меню *DEBUG*, либо соответствующими иконками на панели управления *AVR Studio*.

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

В *AVR Studio* для отладки программы предусмотрены две команды пошагового режима: *Step Over* и *Trace into*. Разница между ними в том, что при выполнении команды *Step Over* выполнение подпрограмм происходит до полного окончания без отображения процесса выполнения. К командам шагового режима также относятся команда *Auto Step*.

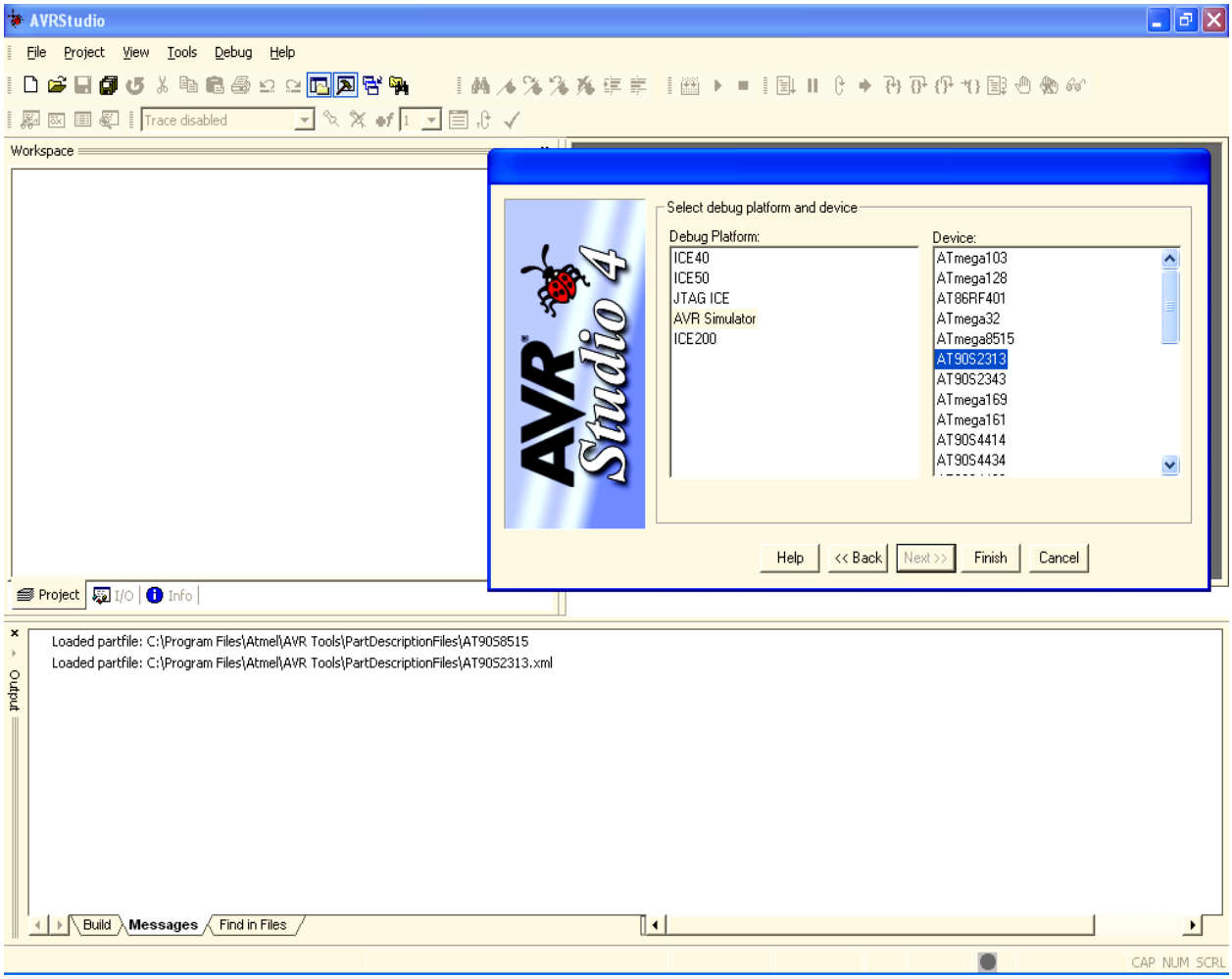


Рисунок 2.3 – Выбор платформы и прибора

С помощью команд пошагового режима можно проследить изменения значений в переменных, регистрах ввода/вывода, памяти и регистрового файла. Для этого предназначены раздел *I/O* рабочей области *AVR Studio* (см. картинку), окно *Watch* (меню *Debug\Quickwatch*).

Интегрированная среда разработки *AVR Studio* также поддерживает просмотр (меню *View\Memory*) ячеек памяти программ, памяти данных, *EEPROM* и регистров портов ввода/вывода в ходе исполнения. Выпадающее меню диало-

гового окна позволяет выбрать один из четырех массивов ячеек памяти: *Data, IO, Eeprom, Program Memory*. Для одновременного просмотра нескольких областей окно *Memory* может быть открыто несколько раз. Информация в диалоговом окне может быть представлена в виде байтов или в виде слов в шестнадцатеричной системе счисления, а также в виде *ASCII* - символов.

В процессе отладки пользователь может инициализировать внутреннее ОЗУ или *EEPROM* микроконтроллера (например, данными, содержащимися в полученном при трансляции файле *.eep*), или сохранить содержимое ОЗУ и *EEPROM* в виде файлов в формате *Intel Hex* (меню *File->Up/Download Memory*).

Помимо шагового режима, возможна отладка программы с использованием точек останова (меню *Breakpoints->Toggle Breakpoint*). Командой *Start Debugging* запускается исполнение программы. Программа будет выполняться до остановки пользователем или до обнаружения точки останова.

Работая с программным симулятором пакета *AVR Studio*, следует помнить, что он пока не поддерживает некоторые режимы работы микроконтроллеров *AVR* и их периферийные узлы:

– аналого-цифровой преобразователь;
– аналоговый компаратор;
– режим часов реального времени;
– режим пониженного энергопотребления (инструкция "*sleep*" интерпретируется программным симулятором как "*nop*");

Возможно, в последующих версиях *AVR Studio* поддержка этих узлов и режимов будет реализована, но в настоящий момент при работе программы с аналоговой периферией, необходимо производить ручной ввод данных в регистры результата.

Для наблюдения за работой программы можно открыть несколько окон, отображающих состояние различных узлов микроконтроллера (см. рисунок). Окна открываются нажатием соответствующих кнопок на панели инструментов или при выборе соответствующего пункта меню *View*. Если в процессе выполнения программы в очередном цикле значение какого-либо регистра изменится, то этот регистр будет выделен красным цветом. При этом, если в следующем цикле значение регистра останется прежним, то цветовое выделение будет снято. Такое же цветовое выделение реализовано в окнах устройств ввода/вывода, памяти и переменных.

Состояние встроенных периферийных устройств микроконтроллера, а также состояния программного счетчика, указателя стека, содержимого регистра статуса SREG и индексных регистров *X, Y* и *Z* отображено в окне *I/O Window*. В этом окне отражаются все функциональные блоки микроконтроллера. Любой блок может быть раскрыт нажатием на его значок. При раскрытии блока в окне отражаются адреса и состояния всех его регистров и отдельных, доступных для модификации, битов. Каждый доступный для модификации бит может быть установлен или сброшен как программой по ходу ее исполнения, так и пользователем вручную (указав курсором мыши нужный бит и, щелкнув левой кнопкой мыши, пользователь может изменить значение бита на обрат-

ное), а в режиме программной симуляции это является способом имитации входного воздействия на микроконтроллер.

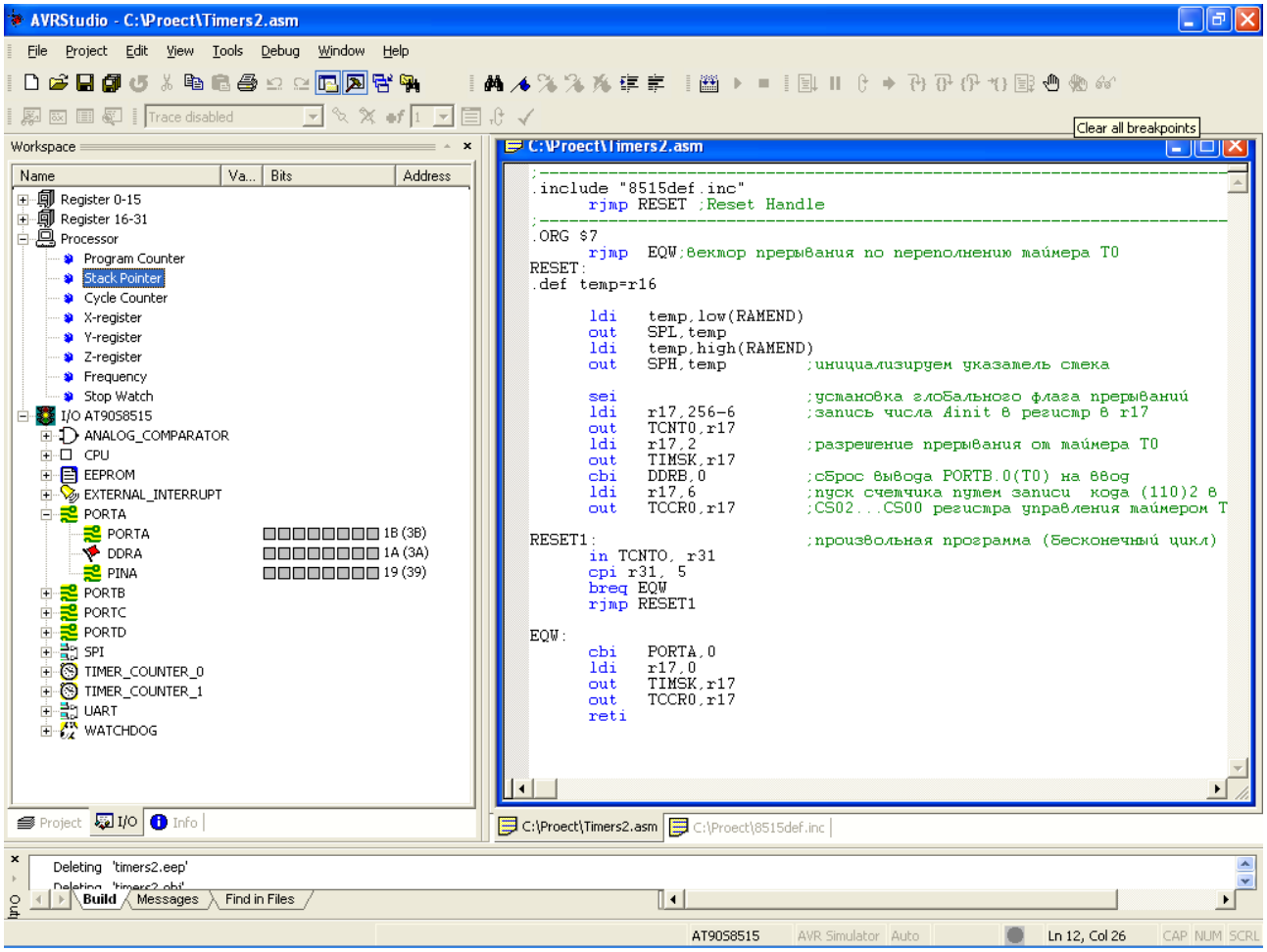


Рисунок 2.4 – Отображение состояния узлов микроконтроллера

Другим способом задания входного воздействия на микроконтроллер в режиме симулятора является использование внешних файлов входных воздействий. Формат файла входного воздействия очень прост:

```
000000000:00
000000039:01
000000040:00
9999999999:FF
```

Здесь значение, указанное после разделителя ":" - это шестнадцатеричное представление сигналов, воздействующих на порт микроконтроллера. Значение, указанное до разделителя - это десятичный номер цикла (с момента сброса микроконтроллера), в котором указанное воздействие поступает на выводы порта микроконтроллера. Файл входного воздействия должен заканчиваться строкой с заведомо большим номером цикла, в противном случае будет выдано сообщение об ошибке. Для подключения файла входного воздействия служит пункт меню *Debug\Options* в разделе *Stimuli and Logging*. В открывшемся окне

нужно указать порт микроконтроллера, на который нужно подавать воздействие, и файл этого воздействия. Пользователь может создавать файлы воздействий, а также записывать изменения значений на выходах портов микроконтроллера в файл (формат этого файла тот же, что и у файла входных воздействий).

Для записи служит функция *Logging*. В открывшемся окне нужно указать порт микроконтроллера и имя файла для записи. Записываемый файл будет удаляться и создаваться вновь при каждом выполнении сброса микроконтроллера (*Debug->Reset*). Подключать файл входного воздействия или задавать имя файла для записи пользователь должен сам при каждом запуске симулятора.

Следует отметить, что *AVR Studio* имеет очень мощную встроенную документацию как по использованию *AVR Studio*, так и по использованию стандартных отладочных средств производства компании *Atmel*, а также системе команд и всему, что касается программирования с использованием *AVR Studio*. Таким образом, использование встроенной справочной информации избавляет разработчика от накопления разобщенной информации, учитывая, что встроенная информация обновляется с выходом новых версий *AVR Studio*.

AVR Studio включает в себя программное обеспечение верхнего уровня для управления аппаратными средствами поддержки разработок, которое было описано выше. По сравнению с программой Ассемблер имеет большие функциональные возможности (например, автоматически подключает директивы Ассемблера), содержит отладчик программ и симулятор многих типов микроконтроллеров *AVR* и поддерживает многие типы программаторов.

3 ЛАБОРАТОРНЫЕ РАБОТЫ ПО ИССЛЕДОВАНИЮ МИКРОКОНТРОЛЛЕРОВ

AVR-микроконтроллерам присущи все основные черты и характеристики современных восьмиразрядных универсальных микроконтроллеров. Однако они имеют преимущества перед аналогичными микроконтроллерами (например, PIC-контроллерами) по производительности, составу команд и, кроме того, фирма-производитель свободно распространяет удобную среду разработки – *AVR Studio*.

Поэтому, с одной стороны, эти микроконтроллеры можно эффективно использовать в различных разработках, включая курсовое и дипломное проектирование, а, с другой стороны, могут являться удобными объектами для исследования микроконтроллеров в лабораторных условиях.

Эти соображения стали основой выбора этих микроконтроллеров в качестве объектов лабораторных исследований.

3.1 Исследование арифметических и логических команд

Эти команды позволяют выполнять стандартные логические операции над байтами, такие как "логическое умножение" (И), "логическое сложение" (ИЛИ), операцию "исключающее ИЛИ", а также вычисление обратного и дополнительного кодов числа (таблица 3.1). К этой группе можно отнести также команды очистки/установки регистров и команду перестановки тетрад. Все операции производятся над регистрами общего назначения, результат сохраняется в одном из РОН. Все логические операции выполняются за один машинный цикл.

Таблица 3.1 – Группа команд логических операций

Мнемоника	Описание	Операция	Циклы	Флаги
AND Rd, Rr	"Логическое И" двух РОН	*Rd = Rd • Rr*	1	*Z,N,V*
ANDI Rd, K	"Логическое И" РОН и константы	*Rd = Rd • K*	1	*Z,N,V*
EOR Rd, Rr	"Исключающее ИЛИ" двух РОН	*Rd = Rd ® Rr*	1	*Z,N,V*
OR Rd, Rr	"Логическое ИЛИ" двух РОН	*Rd - Rd v Rr*	1	*Z,N,V*
ORI Rd, K	"Логическое ИЛИ" РОН и константы	*Rd = Rd v K*	1	*Z,N,V*
COM Rd	Перевод в обратный код	*Rd = $FF - Rd*	1	*Z,C,N,V*
NEG Rd	Перевод в дополнительный код	*Rd = $00-Rd*	1	*Z,C,N,V,H*
CLR Rd	Сброс всех разрядов РОН	*Rd = Rd ® Rd*	1	*Z,N,V*
SER Rd	Установка всех разрядов РОН	*Rd = $FF*	1	—
TST Rd	Проверка РОН на отрицательное или нулевое значение	*Rd.Rd*	1	*Z,N,V*
SWAP Rd	Обмен местами тетрад в РОН	*Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)*	1	—

3.1.1 Программа исследования основных логических команд

```
.include "2313def.inc"
        rjmp RESET;

    RESET:
        ldi r31, low(ramend)        ; формирование стека
        out SPL, r31                ; в верхней части ÎÇÓ

    ;****   Логическое умножение содержимого регистров r16 и r17
        ldi r16, 0xff               ; занесение кода $FF  в регистр r16
        ldi r17, 0x55               ; занесение кода $55 в регистр r17
        and r16, r17                ; результат логического умножения регистре  r16

    ;****   Логическое сложение содержимого регистра r17 и  константы
        ldi r17, 0x55               ; занесение кода $55 в регистр r17
        ori r17, 0x0f               ;лог. сложение содержимого регистра r17 и  константы $0f

    ;****   Логическое сложение содержимого регистров r18 и r19
        ldi r18, 0x05               ; занесение кода $05 в регистр r18
        ldi r19, 0x50               ; занесение кода $50 в регистр r19
        or r18, r19                 ; логическое сложение содержимого регистров r18 и r19

    ;****   Сложение по модулю два содержимого регистров  r20 и r21
        ldi r20, 0x55               ; занесение кода $05 в регистр r20
        ldi r21, 0xf0               ; занесение кода $50 в регистр r21
        eor r20, r21                ; сложение "мод 2" содержимого регистров r20 и r21

        ;****   Обнуление и установка всех разрядов регистра r18
        clr r18                     ;обнуление регистра r18
        ser r18                     ; регистра r18
.EXIT
```

На рисунке 3.1 представлено рабочее поле программы, отражающее состояние регистров общего назначения и регистров процессора после выполнения команды логического умножения. В ходе выполнения программы в пошаговом режиме студенты должны отслеживать изменение состояния узлов микроконтроллера в рабочем поле программы.

К группе арифметических команд относятся команды, выполняющие такие базовые операции, как сложение, вычитание, сдвиг (вправо и влево), инкремент и декремент. Все операции производятся только над регистрами общего назначения. При этом микроконтроллеры *AVR* позволяют легко оперировать как знаковыми, так и беззнаковыми числами, а также работать с числами, представленными в дополнительном коде.

Все команды рассматриваемой группы выполняются за один машинный цикл, за исключением команд, оперирующих двухбайтовыми значениями, которые выполняются за два цикла.

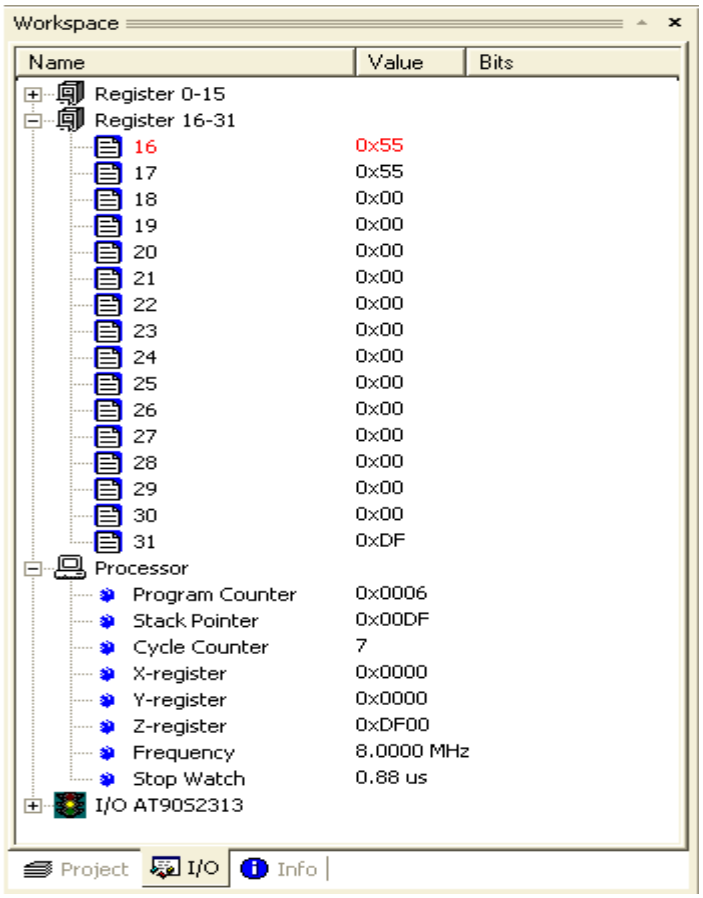


Рисунок 3.1 – Отображение состояния узлов микроконтроллера после выполнения команды логического умножения

3.1.2 Программа исследования основных арифметических команд

```
.include "2313def.inc"
        rjmp RESET;

RESET:
        ldi r31, low(ramend)        ; формирование стека
        out SPL, r31                ; в верхней части ОЗУ

**      Сложение содержимого регистров r16 и r17
        ldi r16, 22                 ;занесение 22 в регистр r16
        ldi r17, 100                ;занесение 100 в регистр r17
        add r16, r17                ;результат сложения в r16

;****   Сложение чисел с разными знаками в дополнительном коде
        ldi r16, -7                 ;занесение -7 в регистр r16
        ldi r17, 5                  ;занесение 05 в регистр r17
        add r16, r17                ;результат сложения в r16
                                    ;флаги N=1 (отрицат. знач.) S=1 (меньше нуля)
```

```
;****   Сложение пары регистров с константой 0=<K=<63
        ldi r24, 0xff          ;занесение числа 511 в пару
        ldi r25, 01            ;регистров  r25 r24
        adiw r24,01            ;результат сложения в  r25 r24
;****                          ;сложение возможно только с парами r24, r26, r28, r30

;****   Вычитание из регистра константы 0=<K=<63
        ldi r22, 0x05          ;занесение кода $05 в регистр r22
        subi r22, 1            ;вычитание 1 из содержимого r22

;****   Декремент и инкремент содержимого регистра
        dec r22                ;уменьшение на 1 содержимого r22
        inc r22                ;увеличение на 1 содержимого r22

;****   сдвиг содержимого регистра
        lsl r22                ;сдвиг содержимого r22 влево на 1 разряд
        lsr r22                ;сдвиг содержимого r22 вправо на 1 разряд
    .EXIT
```

3.2 Исследование ветвящихся участков программ

3.2.1 Команды типа «проверка/пропуск»

В командах этого типа производится проверка условия, результат которой влияет на выполнение следующей команды. Если условие истинно, следующая команда игнорируется. Например, команда *SBRS Rd.b* проверяет разряд b регистра *Rd* и игнорирует следующую команду, если этот разряд равен «1». В действительности переход к следующей инструкции производится увеличением счетчика команд на 1, а пропуск команды требует загрузки нового значения в счетчик команд. Следовательно, когда проверяемое условие истинно, в конвейере возникает задержка. Длительность задержки зависит от пропускаемой команды и составляет от одного до двух машинных циклов.

3.2.2 Команды условного перехода

В этих командах производится проверка условия, результат которой влияет на состояние счетчика команд. Если условие истинно, происходит переход по заданному адресу. Если же условие ложно, выполняется следующая команда.

Команды условного перехода имеют ограничение по области действия, В действительности новое значение счетчика команд получается прибавлением к нему или вычитанием из него некоторого смещения. А поскольку под значение смещения в слове команды отводится всего 7 бит, максимальная величина перехода составляет от -64 до +64 слов.

Так как переход по заданному адресу осуществляется загрузкой нового значения в счетчик команд, то в случае истинности проверяемого условия в конвейере возникает задержка длительностью в один машинный цикл.

3.2.3 Команды безусловного перехода

Для безусловного перехода по требуемому адресу в памяти программ используются команды относительного (*RJMP*) и косвенного (*IJMP*) переводов, т.к. микроконтроллеры *AVR* семейства *Classic* не имеют команды абсолютного перехода (такая команда имеется в микроконтроллерах других семейств, *Tiny* и *Mega*).

3.2.4 Относительный переход – команда *RJMP*

Деятельность команды заключается в изменении содержимого счетчика команд путем прибавления к нему или вычитания из него некоторого значения, являющегося операндом команды, как показано на рисунке 3.2.

Следует помнить, что данная команда имеет ограничение по области действия. Так как операнд представляет собой 12-разрядное число, максимальная величина перехода составляет от -2047 до +2048 слов (±4 Кбайт).

В программах в качестве операндов этой команды вместо констант используются метки. Ассемблер сам вычисляет величину перехода и подставляет это значение в слово команды. Проиллюстрируем сказанное следующим примером:

```
        cpi   rl6,$42          ;Сравниваем регистр r16 с числом $42
        brne  error            ;Переход, если r16 <> $42
        rjmp  ok               ;Безусловный переход

error:

        …

        ok:   nop              ;Место перехода по команде RJMP
```

Поскольку команда относительного перехода изменяет содержимое счетчика команд, она выполняется за 2 машинных цикла.

3.2.5 Косвенный переход – команда *IJMP*

В результате выполнения этой команды программа продолжает выполняться с адреса, находящегося в индексном регистре *Z*. Таким образом, деятельность команды сводится к загрузке содержимого индексного регистра в счетчик команд.

В отличие от команды относительного перехода данная команда не имеет ограничений по области действия. Действительно, поскольку индексный регистр 16-разрядный, максимально возможная величина перехода составляет 64

Кслов (128 Кбайт), а наибольший объем памяти программ микроконтроллеров семейства – всего 8 Кбайт.

Как и команда относительного перехода, команда косвенного перехода выполняется за 2 машинных цикла.

3.2.6 Команды вызова подпрограмм

С командами вызова подпрограмм в микроконтроллерах *AVR* семейства *Classic* дело обстоит так же, как и с командами безусловного перехода. Для вызова подпрограмм имеется две команды: команда относительного вызова (*RCALL*) и команда косвенного вызова (*ICALL*).

3.2.7 Относительный вызов подпрограммы – команда *RCALL*

Если не принимать во внимание некоторые отличия, описанные ниже, эта команда работает так же, как и команда относительного безусловного перехода *RJMP*.

Команда *RCALL* сохраняет в стеке значение счетчика команд. Затем содержимое счетчика команд увеличивается или уменьшается на некоторое значение, являющееся операндом команды (рисунок 3.2). Поскольку операнд представляет собой 12-разрядное число, максимальная величина перехода составляет от -2047 до +2048 слов (±4 Кбайт).

В программах в качестве операндов этой команды, как и в случае команды *RJMP*, используются метки. Ассемблер сам вычисляет величину перехода и подставляет это значение в слово команды

Команда относительного вызова подпрограмм выполняется за 3 машинных цикла, два из которых затрачиваются на сохранение в стеке двух байт счетчика команд.

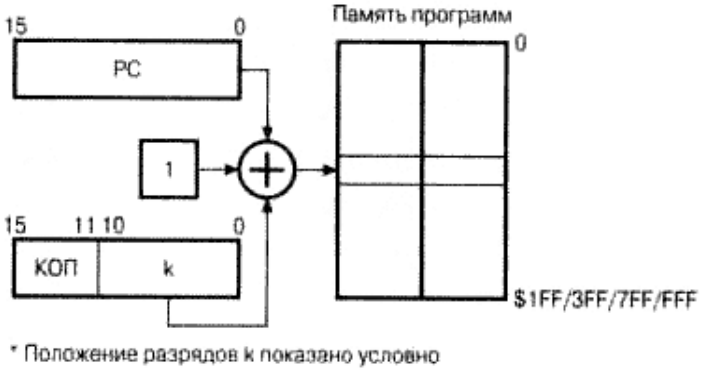


Рисунок 3.2 – Относительная адресация памяти программ

3.2.8 Косвенный вызов подпрограммы – команда *ICALL*

Если не принимать во внимание некоторые отличил, описанные ниже, эта команда работает так же, как и команда косвенного безусловного перехода *IJMP*.

Команда *ICALL* сохраняет в стеке значение счетчика команд. Затем в счетчик команд загружается содержимое индексного регистра. Поскольку индексный регистр 16-разрядный, максимально возможная величина перехода составляет 64 Кслов (128 Кбайт). Поэтому данная команда не имеет ограничений по области действия, т.к. наибольший объем памяти программ микроконтроллеров семейства – всего 8 Кбайт. Как и команда *RCALL*, команда косвенного вызова подпрограмм выполняется за 3 машинных цикла.

3.2.9 Команды возврата из подпрограмм

В конце каждой подпрограммы обязательно должна находиться команда возврата из нее. В системе команд микроконтроллеров семейства имеются две таких команды. Для возврата из обычной подпрограммы, вызываемой командами *RCALL* и *ICALL*, используется команда *RET*. Для возврата из подпрограммы обработки прерывания используется команда *RETI.*

Обе команды восстанавливают из стека содержимое счетчика команд, сохраненное там перед переходом к подпрограмме. Команда возврата из подпрограммы *RETI* дополнительно устанавливает в "1" флаг общего разрешения прерываний I регистра *SREG*, сбрасываемый аппаратно при возникновении прерывания.

На выполнение каждой из команд возврата из подпрограммы требуется 4 машинных цикла.

3.2.10 Программы исследования ветвящихся участков программ

```
;************************************************************
;****    Исследование ветвящихся участков программ
.include "2313def.inc"
    rjmp RESET;

RESET:
    ldi r31, low(ramend)        ; формирование стека
    out SPL, r31                ; в верхней части ОЗУ

;************************************************************
;****    Организация циклов при помощи команд условного перехода
;****    и относительного безусловного перехода

    ldi r16, 0x05           ; занесение кода $05 в регистр r16
loop1:
    dec r16                 ;уменьшение содержимого r16 íà1
    breq equal              ;если равно нулю, то переход на "equal"
    rjmp loop1              ;иначе – на начало цикла "loop1"
equal:
    nop                      ;пустая операция

loop2:
    inc r16                 ;увеличение  содержимого r16 на1
    cpi r16,$05             ;сравнение r16 с числом $05
```

```
    brne loop2          ;если не равно нулю, то переход на "loop2"
    rjmp ok             ;иначе – выход из цикла "ok"
ok:
    nop                 ;пустая операция

;***********************************************************
;****   Организация ветвления при помощи команды  косвенного
;****    безусловного перехода ijmp через регистр  z

.EQU loop3h = 0x01
.EQU loop3l = 0xff

    ldi zl, loop3l   ; занесение кода перехода в регистр zl (младш. часть кода)
    ldi zh, loop3h   ; занесение кода перехода в регистр zl (ст. ÷ часть кода)
    nop

    nop

.ORG 0x1ff
    nop
    ijmp loop3
.EXIT
```

3.3 Исследование портов ввода-вывода

3.3.1 Общие сведения

Как и любые другие микроконтроллеры, микроконтроллеры *AVR* семейства *Classic* имеют порты ввода/вывода. Каждый порт состоит из определенного числа выводов, через которые микроконтроллер может принимать или передавать цифровые сигналы.

Количество доступных портов, или, если точнее, количество контактов ввода/вывода, является одним из основных параметров, влияющих на выбор конкретной модели микроконтроллера.

– *AT90S1200* имеет два порта ввода/вывода: *B* (8-разрядный) и *D* (7-разрядный). Общее количество контактов ввода/вывода равно 15;

– *AT90S2313* также имеет два порта ввода/вывода: *B* (8-разрядныЙ) и *D* (7-разряд-ный). Общее количество контактов ввода/вывода равно 15;

– *AT90S/LS2323* имеет один 3-разрядный порт ввода/вывода *B*;

– *AT90S/LS2343* имеет один 5-разрядный порт ввода/вывода *B*;

– *AT90S/LS2333, AT90S/LS4433* имеют по три порта ввода/вывода: *B* (6-разряд-ный), *C* (6-разрядный) и *D* (8-разрядный). Общее количество контактов ввода/вывода равно 20;

– *AT90S/LS4434, AT90S/LS8535* имеют по четыре 8-разрядных порт ввода/вывода *A, B, C* и *D*. Общее количество контактов ввода/вывод равно 32;

– *AT90S4414, AT90S8515* также имеют по четыре 8-разрядных портов ввода/вывода *A, B, C* и *D*. Общее количество контактов ввода/выводе в этих моделях равно 32;

– *AT90C8534* имеет один 7-разрядный порт *A* и два входа внешних прерываний. В данном микроконтроллере присутствует также и 6-разрядный аналоговый входной порт. Таким образом, общее количество контактов ввода/вывода в этой модели равно 15;

Во всех микроконтроллерах семейства, за исключением *AT90C8534*, большинство контактов ввода/вывода имеет дополнительные функции, поскольку эти выводы также используются периферийными устройствами микроконтроллера.

Конфигурирование каждой лини порта (задание направления передачи данных) может быть произведено программно в любой момент времени. Входные буферы портов построены по схеме триггера Шмитта. Для линий, сконфигурированных как входные, имеется возможность подключения внутреннего подтягивающего резистора сопротивлением 35...120 кОм между входом и проводом питания *Ucc*. Кроме того, если вывод (вход) с подключенным внутренним подтягивающим резистором подключить к общему проводу, он может служить источником тока.

Максимальная нагрузочная способность выходных буферов портов ввода/вывода при логическом "0" на выходе составляет 20 мА. Благодаря этому микроконтроллер может непосредственно управлять светодиодными индикаторами, биполярными и полевыми транзисторами, а также непосредственно быть подключенным ко многим типам датчиков.

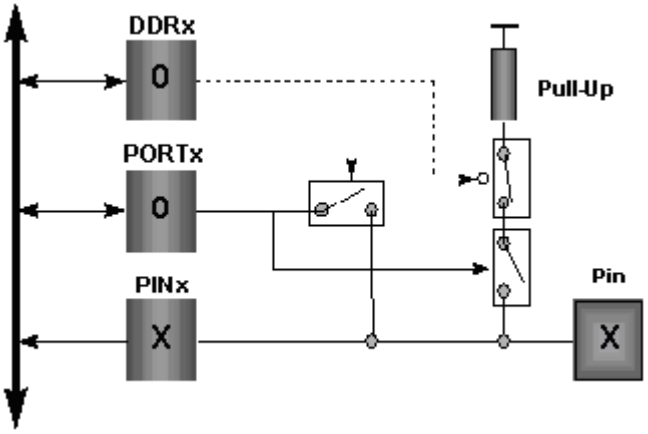
Большинство выводов портов поддерживают альтернативные функции встроенных периферийных устройств микроконтроллера. Все порты являются двунаправленными портами ввода-вывода с опциональными подтягивающими резисторами.

3.3.2 Обращение к портам ввода/вывода

Упрощенная структурная схема элемента ввода/вывода приведена на рисунке 3.3. Здесь *DDRx* - бит контроля направления передачи данных и привязки вывода к шине питания (V_{CC}), *PORTx* - бит привязки вывода к V_{CC} и бит выходных данных, *PINx* - бит для отображения логического уровня сигнала на физическом выводе микросхемы. На рисунке 3.4, в качестве примера приведена структурная схема первого разряда порта *D* (*PD1*) микроконтроллера *AT90S4434*.

Обращение к портам производится через регистры ввода/вывода, причем под каждый порт в адресном пространстве ввода/вывода зарезервировано по 3 адреса (таблица 3.2). По этим адресам размещаются три регистра - регистр данных порта *PORTx*, регистр направления данных *DDRx* и регистр выводов порта *PINx*. Разряды этих регистров имеют названия *Px7...PxO* – для регистров *PORTx*, *DDx7...DDxO* – для регистром *DDRx* и *PINx7...PINxO* – для регистров *PINx*. Действительные названия регистров (и их разрядов) получаются подстановкой названия порта вместо символа «x», соответственно для порта *A* регистры называют *PORTA, DDRA, PINA*, для порта *B* – *PORTB, DDRB, PINB* и т.д.

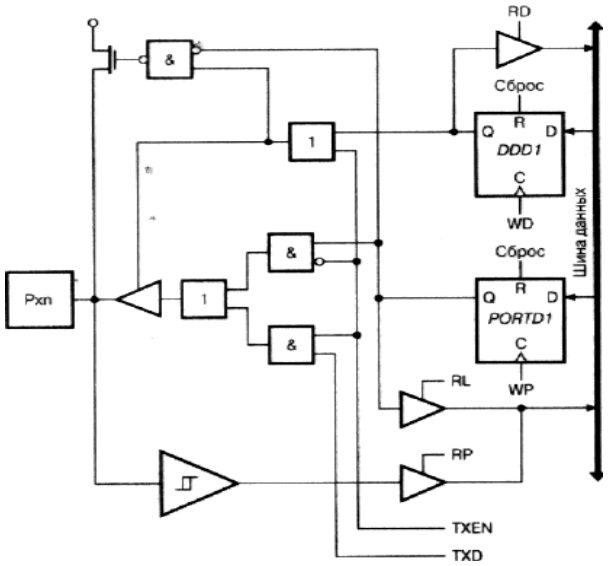
Положение этих регистров в адресном пространстве ввода/вывода приведено в таблице 3.2. При сбросе микроконтроллера регистры *DDRx* и *PORTx* очищаются, а все выводы портов после сброса устанавливаются в третье состояние.



DDRx - бит контроля направления передачи данных и привязки вывода к шине питания (V_{CC}); PORTx - бит привязки вывода к V_{CC} и бит выходных данных; PINx - бит отображения логического уровня сигнала на физическом выводе микросхемы

Рисунок 3.3 – Структура разряда порта ввода/вывода

Следует заметить, что регистры *PINx* на самом деле регистрами не являются, по этим адресам осуществляется доступ к физическим значениям сигналов на выводах порта. Соответственно, они доступны только для чтения, тогда как регистры *PORTx* и *DDRx* доступны и для чтения, и для записи. А в микроконтроллере *AT90C8534* регистр выводов порта вообще отсутствует, т.к. единственный порт этой модели является только портом вывода.



WP – Запись в PORTD; WD –Запись в DDRD; RL –Чтение регистра-защелки порта; RP –Чтение состояния вывода порта; RD –Чтение регистра DDRD; TXD – UART.передаваемые данные; TXEN– UART разрешение передачи

Рисунок 3.4 – Структурная схема канала *PD1* микроконтроллера *AT90S4434*

Порядковый номер вывода порта соответствует, разумеется, порядковому номеру разряда регистров этого порта. Поэтому если разрядность порта меньше восьми, в регистрах порта используется соответствующее число младших разрядов. Незадействованные старшие разряды регистров доступны только для чтения и всегда содержат "0".

Таким образом, запись в порт означает запись требуемого состояния для каждого вывода порта в соответствующий регистр данных порта *PORTx*. А чтение состоянии порта выполняется чтением либо регистра данных порта *PORTx*, либо регистра выводов порта *PINx*. При чтении регистра выводов порта *PINx* происходит считывание логических уровней сигналов, присутствующих на выводах порта. А при чтении регистра данных порта *PORTx* происходит считывание данных, находящихся в регистре защелке порта. Причем сказанное справедливо как для входных, так и для выходных контактов.

Таблица 3.2 – Адреса регистров портов ввода/вывода

Порт	Название регистра	Адрес	*AT9BS1200*	*AT90S2313*	*AT90S/LS2323 AT90S/LS2343*	*AT90S/LS2333 AT90S/LS4433*	*AT90S/LS4434 AT90S/LS8535*	*AT90S4414 AT90S8515*	*AT90C8534*
A	*PORTA*	*$1B ($3B)*					0	0	0
	DDRA	*$1A ($3A)*					0	0	0
	P1NA	*$19 ($39)*					0	0	
B	*PORTB*	*$I8 ($38)*	0	0	0	0	0	0	
	DDRB	*$17 ($37)*	0	0	0	0	0	0	
	PINB	*$16 ($36)*	0	0	0	0	0	0	
C	*PORTC*	*$15 ($35)*				0	0	0	
	DDRC	*$14 ($34)*				0	0	0	
	PINC	*$13 ($33)*				0	0	0	
D	*PORTD1*	*$12 ($32)*	0	0		0	0	0	
	DDRD	*$11 ($31)*	0	0		0	0	0	
	PIND	*$10 ($30)*	0	0		0	0	0	

Контрольное чтение состояния выводов порта *PINx* после записи в порт (запись в регистр *PORTx*) необходимо выполнить с задержкой, учитывающей емкостную нагрузку на выводах порта. Это можно выполнить с помощью одной или двух команд пустых команд *NOP*.

3.3.3 Конфигурирование портов ввода/вывода

Как уже было отмечено, порты имеют всего две возможности по конфигурированию:

– задание направления передачи данных (вход или выход);

– подключение/отключение внутреннего подтягивающего резистора.

Направление передачи данных определяется содержимым регистра передачи данных *DDRx*. Если разряд *DDRxn* - этого регистра установлен в "1", соответствующий *n*-й вывод порта является выходом. Если же разряд *DDRxn* этого регистра сброшен в "0", соответствующий вывод порта является входом.

А вот управление подтягивающим резистором осуществляется с помощью регистра данных порта *PORTx*. Если разряд *Pxn* регистра *PORTx* установлен в "1" и соответствующий вывод порта является входом, между этим выводом и проводом питания подключается подтягивающий резистор. Чтобы отключить подтягивающий резистор, необходимо либо сбросить соответствующий разряд регистра *PORTx*, либо сделать вывод порта выходом.

Таблица 3.3 – Влияние регистров *DDRx* и *PORTx* на конфигурацию выводов портов

Разряды регистров		функции вывода	Резистор	
DDRxn	*PORTxn*			
0	0	вход	отключен	Третье состояние
0	1	вход	подключен	При подключении вывода к общему проводу является источником тока
1		выход	отключен	Выход установлен и "0"
1	1	выход	отключен	Выход установлен в "1"

Примечание: *N= 7, ..0* *номер вывела (разряд порта).*

3.3.4 Примеры конфигурирования

Предположим, что все выводы 8-разрядного порта *A* определены как выходы и требуется установить младшие 4 разряда в "1", а старшие 4 разряда – в "0". Это можно осуществить путем записи в этот порт соответствующего значения, как показано ниже.

```
; Программа вывода числа 0Fh из порта A
include     "8515def.inc"
    rjmp    RESET               ;Reset Handle
;---------------------------------------------------------------
RESET:
;задание  направления передачи
    ldi     r0,$FF              ;запись числа FFh в регистр r0
    out     DDRA,r0             ;запись числа FFh в регистр DDRA
                                ;запись числа 0Fh в порт A
    ldi     r0,$0F              ;запись числа 0Fh в регистр r0
    out     PORTA,r0            ;запись числа 0Fh в порт A (0..3=«1», 4. .7=«0»)
```

Теперь предположим, что все выводы порта *A* определены как входы и требуется узнать их состояние. Это осуществляется следующим образом:

```
;Программа чтения состояния выводов порта A
include     "8515def.inc"
    rjmp    RESET               ;Reset Handle
;---------------------------------------------------------------
RESET:
;задание направления передачи
    ldi     r0,$FF
    out     DDRA,r0
 ;запись числа 0Fh в порт A
    ldi     r0,$0F              ;запись числа 0Fh в регистр r0
    out      PORTA,r0           ;запись числа 0Fh в порт A (0..3=«1», 4. .7=«0»)
 ;формирование задержки
    nop                         ;пустая операция
    in      r0,PINA             ;в регистре r0 - сигналы на выводах порта A
```

Приведенные примеры сознательно упрощены. Все выводы используются либо только как входы, либо только как выходы. Разумеется, можно задать конфигурацию каждого вывода независимо от остальных, так что в одном порту будут находиться одновременно как входы, так и выходы.

3.4 Исследование таймера

Большинство задач управления, которые реализуются с помощью МК, требуют исполнения их в реальном времени. Под этим понимается способность системы получить информацию о состоянии управляемого объекта, выполнить необходимые расчетные процедуры и выдать управляющие воздействия в течение интервала времени, достаточного для желаемого изменения состояния объекта.

Возлагать функции формирования управления в реальном масштабе времени только на центральный процессор неэффективно, так как это занимает ресурсы, необходимые для расчетных процедур. Поэтому в большинстве современных МК используется аппаратная поддержка работы в реальном времени с использованием таймера (таймеров).

Модули таймеров служат для приема информации о времени наступления тех или иных событий от внешних датчиков событий, а также для формирования управляющих воздействий во времени.

Рассмотрим принципы построения и режимы работы таймеров на примере восьмиразрядных универсальных микроконтроллеров *AVR* семейства *Classic*.

3.4.1 Таймеры микроконтроллеров *AVR* семейства *Classic*

Микроконтроллеры семейства *Classic*, в зависимости от модели, имеют в своем составе от одного до трех таймеров общего назначения – *T0, T1, T2* и один сторожевой таймер *WDT*, обеспечивающий перезапуск микроконтроллера при зацикливании программ. В таблице 3.4 приведен состав таймеров микроконтроллеров семейства *Classic*, а в таблице 3.5 – выводы, используемые таймерами.

8-разрядный таймер *T0* имеется во всех моделях. Он может использоваться только для отсчета и измерения временных интервалов или как счетчик внешних событий. При переполнении счетного регистра таймера генерируется запрос на прерывание. 16-разрядный таймер *T1* и 8-разрядный таймер *T2* за счет введения дополнительных аппаратных средств входного захвата (*input capture – IC*) и выходного сравнения (*output compare – OC*) обладают более широкими возможностями реализации алгоритмов реального времени. Эти таймеры также могут использоваться в качестве широтно-импульсных модуляторов. Таймер *T2*, кроме того, может работать в асинхронном (относительно тактового сигнала микроконтроллера) режиме.

В микроконтроллере *AT90C8534* 8-разрядный таймера/счетчика *T0* и 16-разрядный *T1* функционально идентичны и могут использоваться только для

формирования временных интервалов.

Каждый таймер/счетчик (кроме таймеров/счетчиков в *AT90C8534*) использует один или более выводов портов ввода/вывода микроконтроллера. При совместном использовании линий портов ввода/вывода с таймерами/счетчиками необходимо самостоятельно сконфигурировать выводы в соответствии с их функциональным назначением (вход/выход).

Таблица 3.4 – Таймеры/счетчики реального времени

Таймер/счетчик	*AT90S1200*	*AT90S2313*	*AT90S/ LS2323 AT90S/ LS2343*	*AT90S/ LS2333 AT90S/ LS4433*	*AT90S/ LS4434 AT90S/ LS8535*	*AT90S4414 AT90S8515*	*AT90C8534*
Таймер/счетчик *TO*	+	+	+	+	+	+	+
Таймер/счетчик *T1*		+		+	+	+	+
Таймер/счетчик *T2*					+		

Таблица 3.5 – Выводы, используемые таймерами/счетчиками

Название	*AT90S1200*	*AT90S2313*	*AT90S/LS2323 AT90S/LS2343*	*AT90S/LS2333 AT90S/LS4433*	*AT90S/LS4434 AT90S/LS8535*	*AT90S4414 AT90S8515*	Описание
TO	*PD4*	*PD4*	*PB2*	*PD4*	*PBO*	*PBO*	Вход внешнего сигнала таймера *TO*
T1	—	*PD5*	—	*PD5*	*PBI*	*PB1*	Вход внешнего сигнала таймера *T 1*
ICP	—	*PD6*	—	*PBO*	*PD6*	*ICP**	Вход захвата таймера *T1*
OC1	—	*PB3*	—	*PB1*	—	—	Выход схемы сравнения таймера *T1*
OC1A	—	—	—	—	*PD5*	*PD5*	Тоже
OC1B	—	—	—	—	*PD4*	*OC 1B**	Тоже
OC2	—	—	—	—	*PD7*	—	Выход схемы сравнения таймера *T2*
TOSC1	—	—	—	—	*PC6*	—	Вход для подключения резонатора
TOSC2	—	—	—	—	*PC7*	—	Выход для подключения резонатора

* Выделенный вывод микроконтроллера (не линия порта ввода/вывода).

3.4.2 Таймер *T0*

8-разрядный таймер/счетчик *T0* может использоваться для формирования временных интервалов или для подсчета числа внешних событий. Структурная схема таймера *T0* приведена на рисунке 3.5.

В состав таймера входят: регистр управления (*TCCR0*); счетный регистр (*TCNT0*); блок управления таймером. Флаг переполнения счетного регистра таймера *T0V0* находится в регистре флагов прерываний таймеров *TIFR*. Разрешение и запрещение прерываний от таймера *T0* осуществляются установкой/сбросом флага *T0IE0* регистра *TIMSK*.

Счетный регистр таймера *TCNT0* доступен в любой момент времени как для чтения, так и для записи. При записи в регистр *TCNT0* во время работы таймера счет будет продолжен в следующем за командой записи машинном

цикле. После подачи напряжения питания в регистре *TCNT0* находится нулевое значение.

Рисунок 3.5 – Структурная схема таймера/счетчика *T0*

При переходе таймера из состояния *"$FF"* в состояние *"$00"* устанавливается флаг *T0V0* регистра *TIFR* и генерируется запрос на прерывание. Разрешение прерывания осуществляется установкой в "1" разряда *T0IE0* регистра *TIMSK* при установленном флаге общего разрешения прерываний I регистра *SREG*.

Таймер *T0* может работать в двух режимах:

1. Режим таймера. В этом режиме на вход таймера поступают импульсы тактового сигнала микроконтроллера (непосредственно или через предделитель).

Разряд	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Наименование	**-**	**-**	**-**	**-**	**-**	***CS02***	***CS01***	***CS00***
Чтение(R)/Запись(W)	***R***	***R***	***R***	***R***	***R***	***R/W***	***R/W***	***R/W***
Начальное значение	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***

Рисунок 3.6 – Формат регистра *TCCRO*

2. Режим счетчика событий. В этом режиме инкремент содержимого счетного регистра производится по активному фронту сигнала на входе TO микроконтроллера.

Выбор режима работы (источника тактового сигнала), а также запуск и остановка таймера осуществляются с помощью разрядов *CS02...CS00* регистра управления таймером *TCCR0* (рисунок 3.6), расположенного по адресу $33 ($53). Соответствие между состоянием этих разрядов и режимом работы таймера/счетчика приведено в таблице 3.6. Остальные разряды регистра доступны только для чтения и содержат "0".

При использовании таймера в режиме счета внешних событий необходимо помнить, что сигнал, присутствующий на выводе *T0*, синхронизируется с частотой тактового генератора микроконтроллера (состояние вывода *T0* считывается по нарастающему фронту внутреннего тактового сигнала). Поэтому для обеспечения корректной работы таймера от внешнего сигнала промежуток

времени между соседними импульсами должен быть больше периода тактового сигнала микроконтроллера.

Обратите внимание, что инкремент содержимого счетного регистра таймера/счетчика при работе в режиме счета внешних событий производится даже в том случае, если вывод *T0* сконфигурирован как выход. Эта особенность дает пользователю возможность программно управлять процессом счета.

Таблица 3.6 – Выбор источника тактового сигнала для таймера T0

CS02	*CS01*	*CS00*	Источник тактового сигнала
0	0	0	Таймер/счетчик остановлен
0	0	1	*CK (тактовый сигнал микроконтроллера)*
0	1	0	*CK/8*
0	1	1	*CK/64*
1	0	0	*CK/256*
1	0	1	*CK/1024*
I	1	0	Спадающий фронт импульса на выводе *T0**
1	1	1	Нарастающий фронт импульса на выводе *T0**

В модели AT90C8534 значения «110» и «111» разрядов CS02...CS00 зарезервированы (режим счет внешних событий отсутствует).

3.4.3 Таймер *T1*

16-разрядный таймер/счетчик *T*1 реализует следующие функции:

– формирование временных интервалов (аналогично функции таймера *T0*);

– подсчет числа внешних событий (аналогично функции таймера *T0*);

– входной захвата (*input capture – IC*) – сохранение текущего состояния таймера в отдельном РВВ по внешнему сигналу;

– выходного сравнения (*output compare – OC*) – формирование прерывания при равенстве содержимого счетного регистра заданному значению;

– генерация ШИМ-сигнала.

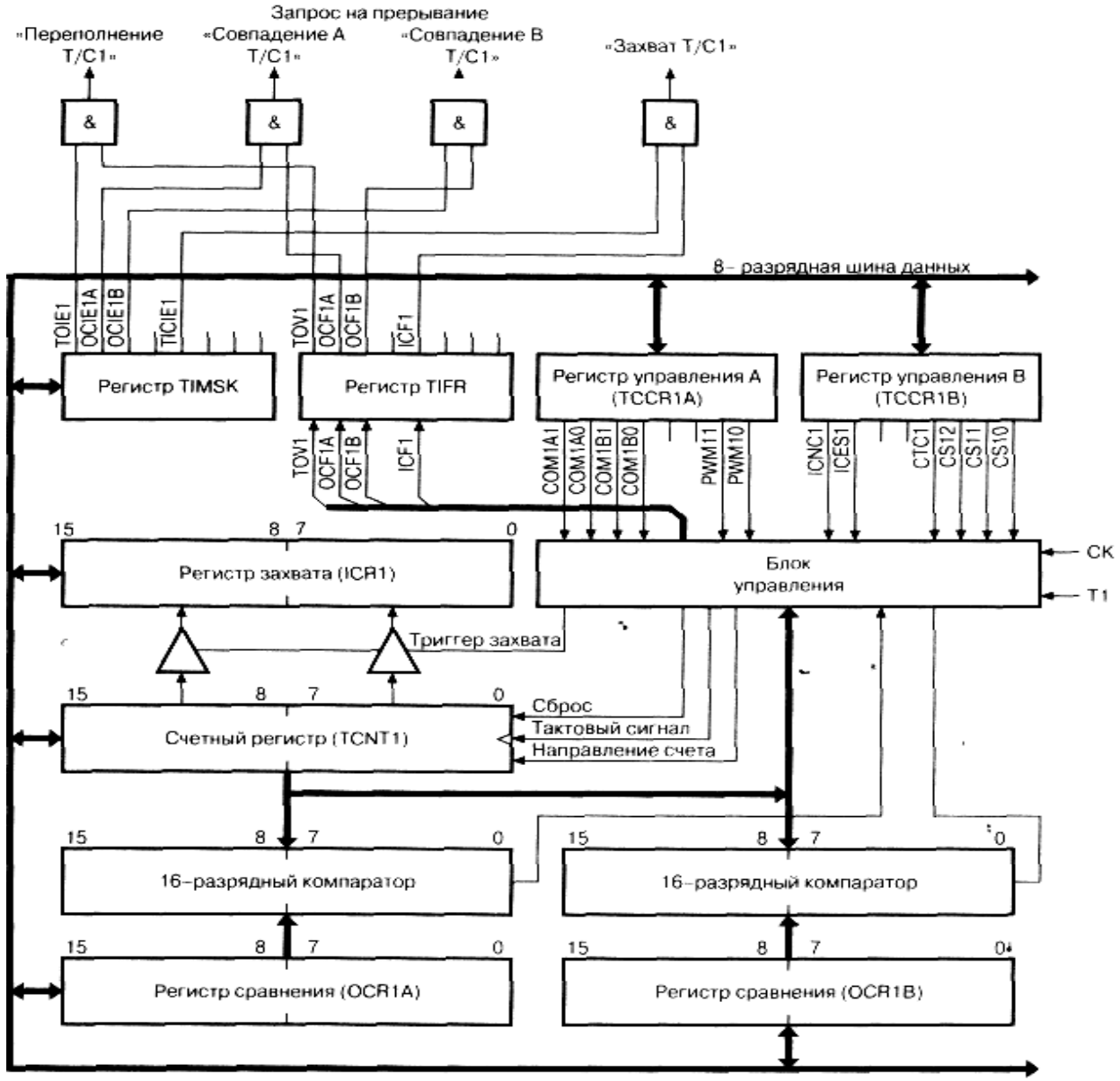
Существует три варианта рассматриваемого таймера/счетчика в зависимости от модели микроконтроллера. Рассмотрим структуру таймера *T1* микроконтроллера *AT90S8515* (рисунок 3.7).

В состав таймера входят четыре 16-разрядных регистра (счетный регистр *TCNT1*, регистр захвата *ICR1*, регистры сравнения *OCR1A* и *OCR1B*), 16-разрядный компаратор, два 8-разрядных управляющих регистра *TCCR1A* и *TCCR1B*, а также блок управления таймером.

Два других варианта имеют сокращенный набор устройств.

Все флаги состояния таймера/счетчика (переполнения, совпадения и захвата) находятся в регистре флагов прерываний от таймеров *TIFR*, а разрешение/запрещение прерываний от таймера осуществляется установкой/сбросом соответствующих флагов регистра *TIMSK*.

Счетный регистр таймера *T1* реализован как суммирующий (в режиме ШИМ — как суммирующий/вычитающий) счетчик и доступен в любой момент времени как для чтения, так и для записи. При записи в регистр *TCNT1* во время работы таймера счет будет продолжен по следующему за операцией записи импульсу тактового сигнала таймера/счетчика. После подачи напряжения питания в регистре *TCNT1* находится нулевое значение.



Примечание: Позиции разрядов регистров TIMSK и TIFR показаны условно

Рисунок 3.7 – Структурная схема таймера *T1*

Физически регистр *TCNT1* размещен в двух регистрам *TCNT1H*:*TCNT1L*, расположенных по адресам *$2D:$2C* (адреса в адресном пространстве ОЗУ, соответственно, *$4D:$4C*). Чтобы при обращении ЦПУ микроконтроллера к этим регистрам запись или чтение обоих байтов содержимого счетного регистра происходило одновременно, обращение производится с использованием специального 8-разрядного регистре *TEMP* (этот регистр используется только процессором и программно недоступен). Этот же регистр используется и при обращении к остальным 16-разрядным регистрам таймера/счетчика *Tl: OCR1* (*OCR1A* и *OCR1B*) и *ICR1*. Прерывания на время обращения к любому из этих регистров должны быть запрещены.

Запись в регистр *TCNT1*.При записи старшего байта значения в регистр *TCNT1H* он помещается в регистр *TEMP*. Далее, при записи младшего байта в регистр *TCNT1L* он объединяется с содержимым регистра *TEMP* и оба байта

записываются в регистр *TCNT1* одновременно. Из сказанного видно, что для выполнения полного цикла записи в 16-разрядный регистр первым должен быть загружен старший байт (регистр *TCNT1H*).

Чтение регистра *TCNT1*. При чтении регистра *TCNT1L* (младший байт) содержимое регистра *TCNT1H* пересылается в регистр *TEMP*. А при последующем чтении регистра *TCNT1H* возвращается значение, сохраненное в регистре *TEMP*. Следовательно, для выполнения полной операции чтения 16-разрядного регистра первым должен быть прочитан младший байт (регистр *TCNT1L*).

Управление таймером/счетчиком *T1* осуществляется с помощью двух регистров управления *TCCR1A* и *TCCR1B* (в модели *AT90C8534* — с помощью одного регистра *TCCR1*). Эти регистры расположены по адресам *$2F ($4F*) (регистр *TCCR1A*) и *$2E* (*$4E*) (регистры *TCCR1B, TCCR1*).

Формат регистра *TCCR1A* приведен на рисунке 3.8, а регистров *TCCR1B* и *TCCR1* — на рисунке 3.9. Неиспользуемые разряды регистров доступны только для чтения.

Разряд	***7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***	***AT90S/LS4434***
Наименование	***COM1A1***	***COM1AO***	***COM1A1***	***COM1AO***	***-***	***-***	***PWM11***	***PWM10***	***AT90S/LS8535***
Чтен.(R)/Зап.(W)	***R/W***	***R/W***	***R/W***	***R/W***	***R***	***R***	***R/W***	***R/W***	***AT90S4414***
Нач. значение	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***AT90S8515***

Рисунок 3.8 – Формат регистра *TCCR1A*

Разряд	***7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***
Наименование	***ICNCI***	***ICES1***	***-***	***-***	***CTC1***	***CS12***	***CS11***	***CS10***
Чтение(R)/Запись(W)	***R/W***	***R/W***	***R***	***R***	***R/W***	***R/W***	***R/W***	***R/W***
Начальное значение	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***

а)

Разряд	***7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***
Наименование	***-***	***-***	***-***	***-***	***CTC1***	***CS12***	***CS11***	***CS10***
Чтение(R)/Запись(W)	***R***	***R***	***R***	***R***	***R/W***	***R/W***	***R/W***	***R/W***
Начальное значение	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***

б)

Рисунок 3.9 –Формат регистров: а — *TCCR1B*; б — *TCCR1*

По отношению к тактовому сигналу таймер/счетчик *T1* может работать в двух режимах:

1. Режим таймера. В этом режиме на вход таймера/счетчика поступают импульсы тактового сигнала микроконтроллера (непосредственно или через предделитель).

2. Режим счетчика событий. В этом режиме инкремент содержимого I счетного регистра производится по активному фронту сигнала на входе *T1* микроконтроллера. Выбор источника тактового сигнала, а также запуск и остановка таймера/счетчика осуществляются с помощью разрядов *CS12...CS10* регистр управления таймером *TCCR IB* (таблица 3.7).

При использовании внешнего тактового сигнала необходимо помнить, что он синхронизируется с частотой тактового генератора микроконтроллера (состояние вывода *T1* считывается по нарастающему фронту внутреннего тактового сигнала). Поэтому для обеспечения корректной работы таймера от внешнего сигнала промежуток времени между соседними импульсами должен быть больше периода тактового сигнала микроконтроллера.

При использовании внешнего тактового сигнала инкремент содержимого счетного регистра таймера/счетчика производится даже в том случае, если вывод *T1* сконфигурирован как выход. Эта особенность дает пользователю возможность программно управлять процессом счета.

Таблица 3.7 – Выбор источника тактового сигнала для таймера T1

Регистр *TCCR1B (TCCR1)*			Источник тактового сигнала
CS12	*CS11*	*CS10*	
0	0	0	Таймер/счетчик остановлен
0	0	1	СК (тактовый сигнал микроконтроллера)
0	1	0	*СК/8*
0	1	1	*СК/64*
1	0	0	*СК/256*
1	0	1	*СК/1024*
1	1	0	Вывод *T1*, инкремент счетчика - по спадающему фронту импульсе*
1	1	1	Вывод *T1*, инкремент счетчика - по нарастающему фронту импульсе*

**В модели AT90C8534 значения «110» и «111» разрядов CS12...CS10 зарезервированы (режим счета внешних событий отсутствует).*

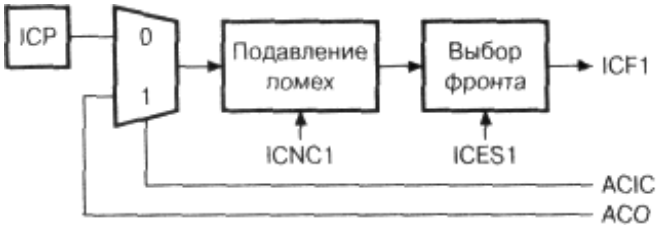
3.4.3.1 Режим таймера

Принцип работы таймера *T1* в этом режиме такой же, как и таймера/счетчика *T0*. По каждому импульсу, поступающему на тактовый вход таймера/счетчика, производится инкремент содержимого счетного регистра *TCNT1*. При переходе таймера/счетчика из состояния "*SFFFF*" в состояние "*$0000*" устанавливается флаг *T0V1* регистра *TIFR* и генерируется запрос на прерывание. Разрешение прерывания осуществляется установкой в "1" разряда *TOIE1* регистра *TIMSK* (разумеется, флаг общего разрешения прерываний I регистра *SREG* также должен быть установлен в "1").

3.4.3.2 Функция захвата (*Capture*)

Данная функция заключается в сохранении в определенный момент времени состояния таймера/счетчика в регистре захвата *ICR1*. Это действие может производиться либо по активному фронту сигнала на выводе ICP микроконтроллера, либо по сигналу от аналогового компаратора. При этом устанавливается флаг *ICF1* регистра *TIFR* генерируется запрос на прерывание. Разрешение прерывания осуществляется установкой в "1" разряда *TICIE1* регистра *TIMSK*.

Упрощенная структурная схема узла, выполняющего данную функцию, приведена на рисунке 3.10.



ACIC - разрешение захвата по ACO; ACO - выход компаратора

Рисунок 3.10 – Структурная схема узла захвата

Для управления схемой захвата используются два разряда регистра *TCCR1B*: *ICNC1* и *ICES1*. Разряд *ICNC1* управляет схемой подавления помех. Если этот разряд сброшен в "0", схема подавления помех выключена и захват производится по первому же активному фронту на выводе *ICP* микроконтроллера. Если же этот разряд установлен в "1", то при появление активного фронта на выводе *ICP* производится 4 выборки с частотой, равной тактовой частоте микроконтроллера. Захват будет выполнен только в том случае, если все выборки имеют уровень, соответствующий активному фронту сигнала (лог. "1" для нарастающего и лог. "0" для спадающего).

Активный фронт сигнала, т.е. фронт, по которому будет выполнено сохранение содержимого счетного регистра в регистре захвата, определяется состоянием разряда *ICES* 1. Если этот разряд сброшен в "0", то активным является спадающий фронт. Если же этот разряд установлен в "1", то активным является нарастающий фронт. Разумеется, вывод *ICP* должен быть сконфигурирован как входной, т.е. разряд регистра управления портом *DDRx*, соответствующий данному выводу, должен быть сброшен в "0".

Физически регистр захвата *ICR1* размещен в двух регистрах *ICR1H:ICR1L*, расположенных по адресам $27:$26 (адреса в адресной пространстве ОЗУ соответственно $47:$46) и доступных только для чтения. Поскольку регистр захвата является 16-разрядным, при его чтении используется временный регистр *TEMP*. При чтении регистра *ICR1L* (младший байт) содержимое этого регистра пересылается в ЦПУ, а содержимое регистра *ICRH* (старший байт) сохраняется в регистре *TEMP*. При чтении регистра *ICR1H* возвращает значение, сохраненное в регистре *TEMP*. Следовательно, при чтении регистра *ICR1* первым должен быть прочитан регистр *ICR1L*. Прерывания на время обращения к регистру *ICR1* должны быть запрещены.

3.4.3.3 Функция сравнения (*Compare*)

Данная функция заключается в непрерывном (каждый машинный цикл) сравнении содержимого счетного регистра таймера/счетчика с числом, находящемся в регистре сравнения. При совпадении содержимого этих регистров устанавливается флаг соответствующего прерывания, а также могут выполняться другие действия.

Количество регистров сравнения, а также их названия отличаются в различных моделях микроконтроллеров (см. таблицу 3.8). Причем в моделях, имеющих по два регистра сравнения, операция сравнения производится независимо для каждого регистра.

Если состояние таймера становится равным числу, находящемуся в регистре сравнения, то в следующем машинном цикле устанавливается соответствующий этому регистру флаг прерывания в регистре *TIFR* для регистра *OCR1*

– флаг *OCF1*, для регистра *OCR1A* – флаг *OCF1A*, для регистра *OCRIB* – флаг *OCF1B*) и генерируется запрос на прерывание. Разрешение прерываний осуществляется установкой в "1" соответствующих флагов регистра *TIMSK* (*OCIE1*, *OCIE1A* и *OCTE1B* для регистров *OCR1*, *OCR1A* и *OCR1B* соответственно).

Наряду с установкой флага в регистре *TIFR* при равенстве счетного регистра и регистра сравнения могут выполняться и другие действия:

– сброс таймера/счетчика (только для регистров *OCR1* и *OCR1A*);

Таблица 3.8 – Регистры сравнения таймера/счетчика Т1

Модель	Регистр сравнения	
	первый	второй
T90S2313	*OCR1A*	
T90S/LS2333 -T90S/LS4433	*OCR1*	
-T90S/LS4434 -T90S/LS8535	*OCRIA*	*OCR1B*
T90S4414/8515	*OCR1A*	*OCR IB*

– изменение состояния определенного вывода микроконтроллера (для всех регистров).

Поведение микроконтроллера, т.е. выполнение или невыполнение указанных действий, определяется несколькими разрядами регистров управления *TCCR1A* и *TCCR1B*. Названия этих разрядов и их описание приведены в таблице 3.9.

Каждый регистр сравнения размещается в двух регистрах ввода-вывода РВВ:

– *OCR1 – OCR1H:OCR1L*, адреса $2B:$2A ($4B:$4A);
– *OCR1A – OCR1AH:OCR1AL*, адреса $2B:$2A ($4B:$4A);
– *OCR1B – OCR1BH:OCR1BL*, адреса $29:$28 ($49:$48).

Поскольку регистры сравнения являются 16-разрядными, при их чтении и записи используется временный регистр *TEMP*. При записи регистра сравнения первым должен записываться регистр *OCRlxH* (старший 5айт), а при чтении первым должен считываться регистр *OCRlxL* (младший байт). Прерывания на время обращения к регистру сравнения должны быть запрещены.

3.4.3.4 Режим ШИМ

В этом-режиме таймер *T1* представляет собой одинарный или сдвоенный, в зависимости от модели, широтно-импульсный модулятор. Широтно-импульсная модуляция является одним из видов непрерывной импульсной модуляции, при котором ширина импульса пропорциональна значению модулирующего сигнала. Соответственно в данном случае широтно-импульсная модуляция заключается в генерировании сигнала с программируемыми частотой и скважностью.

Для перевода таймера *T1* в режим ШИМ и задания частоты ШИМ-сигнала используются разряды *PWM11*:*PWM10* регистра управления таймером *TCCR1A*. Соответствие между состоянием этих разрядов и режимом работы таймера/счетчика Т1 приведено в таблице 3.10.

Таблица 3.9 – Управление работой схемы сравнения таймера *T1*

Регистр		Название	Модель AT90 …	Описание		
			S2313	**Управление выводом *OC1A* (*OC1*)* микроконтроллера** Состояние этих разрядов определяет поведение вывода *OC14 (OC1)* при совпадении содержимого счетного регистра и регистра сравнения *OCR1A* Изменение состояния этих разрядов соответствующее прерывание от компаратора таймера рекомендуется запретить (во избежание ложной генерации прерывания). Чтобы таймер мог управлять этим выводом, он должен быть сконфигурирован как выходной. Поведение вывода задается следующим образом:		
				COM1A1	*COM1A0*	Описание
				0	0	Таймер/счетчик *T1* отключен от вывода *OC1A*
				0	1	Состояние вывода меняется на противоположное
				1	0	Вывод сбрасывается в «0»
				1	1	Вывод устанавливается в «1»
TCCR1A	5,4	*COM1B1. COM1BO*	*S4434* *LS4434* *S8535* *LS8535* *S4414* *S8515*	**Управление выводом OC1B микроконтроллера** Состояние этих разрядов определяет поведение вывода *OC1B* при совпадении содержимого счетного регистра и регистра сравнения *OCRIB.* При изменении состояния этих разрядов соответствующее прерывание от компаратора таймера рекомендуется запретить (во избежание ложной генерации прерывания). Чтобы таймер мог управлять этим выводом, он должен быть сконфигурирован как выходной. Поведение вывода задается следующим образом:		
				COM1B1	*COM1BO*	Описание
				0	0	Таймер/счетчик *T1* отключен от вывода *OC1B*
				0	1	Состояние вывода меняется на противоположное
				1	0	Вывод сбрасывается в «0»
				1	1	Вывод устанавливается в «1»
TCCR1B	3	*CTC1*	Все	**Сброс таймера/счетчика** Если этот разряд установлен в «1», то при совпадении содержимого счетного регистра и регистра сравнения *OCR A* (*OCR1*) производив сброс таймера в состояние «$0000»**		

Таблица 3.10 – Управление режимом ШИМ таймера T1

Регистр *TCCR1A*		Описание
PWM11	*PWM10*	
0	0	Режим ШИМ таймера/счетчика T1 выключен
0	1	8-разрядный широтно-импульсный модулятор
1	0	9-разрядный широтно-импульсный модулятор
1	1	1 0-разрядный широтно-импульсный модулятор

Таблица 3.11 – Модуль счета счетного регистра и частота ШИМ-сигнала

Разрешение модулятора	Модуль счета (TOP)	Частота ШИМ-сигнала
8 разрядов	*$00FF (255)*	$f_{TCK1}/510$
9 разрядов	*$01FF(511)*	$f_{TCK1}/1022$
10 разрядов	*$03FF (1023)*	$f_{TCK1}/2046$

Примечание: f_{TCK1}— частота тактового сигнала таймера/счетчика T1.

** Для AT90S2313, AT90S/LS2333 и AT90S/LS2343 — OC1,*
для AT90S/LS4434, AT90S/LS8535 и AT90S4414/8515 — OC1 A.
*** Поскольку сброс выполняется в машинном цикле, следующем за тем, во время которого произошло совпадение, поведение таймера зависит от установленного коэффициента деления предделителя таймера (см. табл. 3.7). При коэффициенте деления, равном 1, состояние таймера меняется следующим образом:*

$$... \rightarrow C\text{-}2 \rightarrow C\text{-}1 \rightarrow C \rightarrow 0 \rightarrow 1 \rightarrow ...$$

а при коэффициенте деления, не равном 1:

$$...- \rightarrow (C\text{-}2, ..., C\text{-}2) \rightarrow (C\text{-}1, ..., C\text{-}1) \rightarrow (C, 0, ..., 0) \rightarrow ...,$$

где C — число, находящееся в регистре сравнения.

Для генерации сигнала с ШИМ используется схема сравнения таймера, поэтому в моделях *AT90S2313*, *AT90S/LS2333* и *AT90S/LS4433* модулятор является одинарным (один регистр сравнения), а в моделях *AT90S/LS4434*, *AT90S/LS8535* и *AT90S4414/8515* – сдвоенным (два регистра сравнения).

Сигнал ШИМ снимается с выхода схемы сравнения таймера. В рассматриваемом режиме счетный регистр таймера функционирует как реверсивный счетчик, модуль счета которого (*TOP*) зависит от режима работы модулятора. Значение модуля счета и соответствующая частота ШИМ-сигнала для каждого режима работы модулятора приведены в таблице 3.11.

При работе таймера *T1* в режиме ШИМ состояние счетного регистра изменяется от $000 до значения *TOP*, а затем снова до $000 после чего цикл повторяется. При равенстве состояния счетчика и содержимого регистра сравнения состояние соответствующего этому регистру вывода микроконтроллера изменяется согласно таблице 3.12 (рисунок 3.11). Таким образом, длительность импульса равна удвоенному значению содержимого регистра сравнения.

Соответственно если в счетный регистр записать значение $000 или TOP, то при следующем совпадении состояния счетчика и содержимого регистра сравнения выход схемы сравнения переключится в устойчивое состояние согласно таблице 3.13.

Особенностью работы таймера/счетчика T1 в режиме ШИМ является то, что при записи в регистр сравнения младшие 10 разрядов записываемого числа на самом деле сохраняются в специальном временном регистре (не путать с регистром *TEMP*). А изменение содержимого регистра сравнения происходит только в момент достижения счетчиком максимального значения (*TOP*). Благодаря такому решению исключается появление несимметричных выбросов сигнала на выходе модулятора (помех), которые были бы неизбежны при непосредственной записи в регистр сравнения.

Соответственно при чтении регистра сравнения в промежутке между записью в него и его действительным изменением возвращается содержимое временного регистра. То есть всегда возвращается значение, записанное последним.

Таблица 3.12 – Поведение выходов схемы сравнения в режиме ШИМ

Регистр TCCR1A		Поведение вывода *OClx* (*OC1*)
COM1x1 (*COM11*)	*COMlx0* (*COM10*)	
0	0	Таймер/счетчик *T1* отключен от вывода
0	1	Таймер/счетчик *T1* отключен от вывода
1	0	Сбрасывается в «0» при прямом счете и устанавливается в « 1» при обратном счете (неинвертированный ШИМ-сигнал)
1	1	Устанавливается в «1» при прямом счете и сбрасывается в «0» при обратят счете (инвертированный ШИМ-сигнал)

Примечание: x – A или B.

При работе таймера *T1* в режиме ШИМ может генерироваться прерывание по переполнению счетного регистра таймера, а также прерывания от схемы сравнения.

Таблица 3.13 – Устойчивые состояния выхода схемы сравнения

Регистр *TCCR1A*		Регистр *OCRlx* (*OCR1*)	Состояние вывода *OClx* (*OC1*)
COMU1 (*COMИ*)	*COMUO* (*COM10*)		
1	0	$000	0
1	0	*TOP*	1
1	1	$000	1
1	1	*TOP*	0

Примечание: x — A или B.

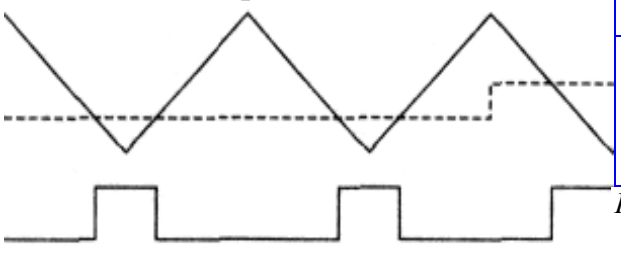


Рисунок 3.11 – Формирование ШИМ-сигнала

Флаги прерываний устанавливаются в «1» при изменении счетчиком направления счета: флаг *TOV1* — в точке $000, а флаги *OCF1* (для регистра *OCR1*), *OCF1A* (для регистра *OCR1A*) и *OCF1B* (для регистра *OCRIB*) – в точке *TOP*. Разрешение и обработка соответствующих прерываний выполняются как обычно.

3.4.4 Сторожевой таймер (*WATCHDOG*)

Основная функция сторожевого таймера – защита устройства от сбоев. Благодаря сторожевому таймеру можно прервать выполнение зациклившейся программы или выйти из других непредвиденных ситуаций, препятствующих нормальному выполнению программы.

Структурная схема сторожевого таймера приведена на рисунке 3.12.

Сторожевой таймер имеет независимый генератор, поэтому он работает даже во время нахождения микроконтроллера в режиме *Power Down*. Частота этого генератора зависит от напряжения питания устройства, температуры, технологического разброса. Типовое значение частоты равно 1 МГц при *fcc* = 5.0 В и 350 кГц при *Vcc* = 3.0 В.

Если сторожевой таймер включен, то через определенные промежутки времени (при наступлении тайм-аута) выполняется сброс микроконтроллера. Чтобы избежать сброса микроконтроллера при нормальном выполнении программы, сторожевой таймер необходимо регулярно сбрасывать через промежутки времени, меньше его периода. Сброс сторожевого таймера осуществляется командой *WDR*.

Для управления сторожевым таймером предназначен регистр *WDTCR*, расположенный по адресу $21 ($41). Формат этого регистра приведен на рисунке 3.13.

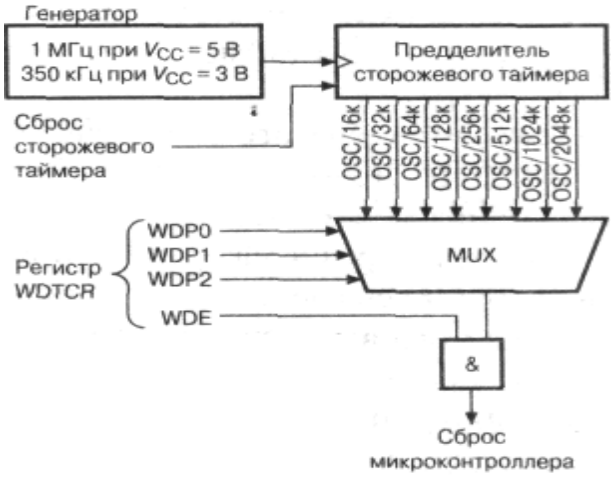


Рисунок 3.12 – Структурная схема сторожевого таймера

Разряд	7	6	5	4	3	2	1	0
Наименование	-	-	-	*WDTOE*	*WDE*	*WDP2*	*WDP1*	*WDP0*
Чтение(*R*)/Запись(*W*)	*R*	*R*	*R*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.13 – Формат регистра *WDTCR*

Для включения/выключения сторожевого таймера используются два разряда регистра *WDTCR* – *WDE* и *WDTOE*. Если разряд *WDE* установлен в "1", сторожевой таймер включен, если сброшен в "0" – выключен. Непосредственно

перед включением таймера рекомендуется также выполнять его сброс командой *WDR*.

Чтобы избежать непреднамеренного выключения таймера счетчика, предназначен разряд *WDTOE*. Дело в том, что выключение сторожевого таймера (сброс разряда *WDE*) можно осуществить только при установленном разряде *WDTOE*. Причем через 4 машинных цикла после установки в "1" этот разряд аппаратно сбрасывается, благодаря чему практически исчезает возможность случайного выключения сторожевого таймера.

Исходя из сказанного, для выключения сторожевого таймера рекомендуется следующая последовательность действий:

– одной командой записать лог. "1" в разряды *WDE* и *WDTOE*;

– в течение следующих четырех машинных циклов записать лог. "0" в разряд *WDE*.

Период наступления тайм-аута сторожевого таймера задается с помощью разрядов *WDP2...WDP0* регистра *WDTCR* согласно таблицы 3.14.

Чтобы избежать непреднамеренного сброса микроконтроллера при изменении периода сторожевого таймера, необходимо перед записью разрядов *WDP2:WDP0* либо запретить работу сторожевого таймера, либо сбросить его.

Таблица 3.14 – Задание периода сторожевого таймера

WDP2	*WDP1*	*WDPO*	Число тактов генератора	Период наступления тайм-аута (типовое значение)	
				Vcc =3.0 В	*Vcc* = 5.0 В
0	0	0	16-1024	47 мс	15 мс
0	0	1	32-1024	91 мс	30 мс
0	1	0	64-1024	0.19с	60 мс
0	1	1	128-1024	0.38с	0.12с
1	0	0	256-1024	0.75с	0.24с
1	0	1	512-1024	1.5с	0.49с
1	1	0	1024-1024	3.0с	0.97с
1	1	1	2048-1024	6.0с	1.9с

3.4.5 Формирование временных интервалов

Для программирования временных интервалов необходимо загрузить в счетчик соответствующее значение и определить коэффициент деления предделителя. Для расчета величины временной задержки τ используется формула (при однократном запуске таймера):

$$\tau = \frac{(2^n - A_{Init}) \cdot k}{F},$$

где n – разрядность счетчика соответствующего таймера; *AInit* - начальная установка таймера; k - коэффициент деления предделителя; F – частота генератора микроконтроллера.

Значение начальной установки таймера, которое необходимо загрузить в счетчик, определяется по формуле:

$$A_{Init} = 2^n - (\tau \cdot F / k).$$

При этом точность формирования временного интервала $\Delta\tau$ определяется отношением $\Delta\tau = k / F$.

Рассмотрим пример программы формирования временного интервала с использованием восьмиразрядного таймера *T0* микроконтроллера *AT90S8515* (*n*=8).

Пусть требуется получить интервал *T*=1.5 мкс. Тогда начальная установка таймера *AInit* будет равна

$$A_{Init} = 2^8 - (1.5 \cdot 10^{-6} \cdot 4 \cdot 10^6 / 1) = 256 - 6 = 250$$

или в шестнадцатеричной системе счисления – *FA*.

3.4.6 Программа исследования таймера в режиме формирования временного интервала

```
;Программа формирования временного интервала
;---------------------------------------------------------------------------
.include "8515def.inc"
      rjmp RESET ;Reset Handle
;---------------------------------------------------------------------------
.ORG $7                          ;вектор прерывания по переполнению таймера T0
    rjmp TIME0
RESET:
.def temp=r16

      ldi   temp,low(RAMEND)  ;инициализируем указатель стека
      out   SPL,temp
      ldi   temp,high(RAMEND)
      out   SPH,temp

      sei                       ;установка глобального флага прерываний
      ldi   r17,256-6           ;запись числа A_init в регистр r17
      out   TCNT0,r17
      ldi   r17,2               ;разрешение прерывания от таймера T0
      out   TIMSK,r17
      sbi   DDRA,0              ;установка вывода PORTA.0 на вывод
      ldi   r17,1               ;пуск счетчика путем записи кода (001)2 в разряды
      out   TCCR0,r17           ;CS02...CS00 регистра управления таймером TCCR0
      sbi   PORTA,0             ;установка вывода PORTA.0 в состояние"1"
      nop                       ;произвольная программа (пустые команды)
      nop
      nop
      nop
      nop
      rjmp RESET1

;Подпрограмма обработки прерывания по переполнению счетчика таймера T0
   TIME0:
      cbi   PORTA,0             ;сброс вывода PORTA.0 в состояние"0"
      cli                       ;глобальное запрещение прерываний
      ldi r17,0
```

```
        out TIMSK,r17           ; запрещение прерываний от таймера T0
        out TCCR0,r17           ;остановка счетчика таймера T0
        reti                    ;выход из подпрограммы
RESET1:
```

3.4.7 Исследование режима сравнения

Структурная схема счетчика-таймера *T2* микроконтроллера *Mega* представлена на рисунке 3.14. Основное отличие этого счетчика от счетчика *T0* семейства *Classic* заключается в наличии аппаратных средств для организации режима сравнения – регистра сравнения, компаратора, схем генерации прерываний.

Управление таймером *T2* осуществляется при помощи регистра управления TCCR2.Формат этого регистра представлен на в таблице 3.9.

Таблица 3.15 – Формат регистра *TCCR2*

Разряд	7	6	5	4	3	2	1	0
Наименование	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
Чтение(R)/ Запись(W)	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Этот регистр содержит три поля разрядов:

– поле *FOC* – единица в этом разряде разрешает изменение вывода *OC2* микроконтроллера в соответствии с логикой работы таймера в режиме *CTC* (сброс при совпадении);

– поле *WGM2* – два разряда поля определяют режим работы таймера-счетчика (таблица 3.10);

– поле *COM2* – два разряда этого поля управляют выводом *OC2* (таблица 3.11);

– поле *CS2* – три разряда этого поля управляют предделителем таймера *T2* (таблица 3.12).

В нормальном режиме формируется прерывание "переполнение" при переходе счетчика из состояния *FF* в состояние 0 и прерывание "сравнение" при достижении счетчиком значения, записанного в регистр сравнения *OCR2*.

Таблица 3.16 – Режим работы таймера-счетчика T2

Номер режима	Код поля		режим работы
	WGM21	WGM20	
1	0	0	Нормальный
2	0	1	ШИМ с точной фазой
3	1	0	СТС (сброс при совпадении)
4	1	1	Быстрый ШИМ

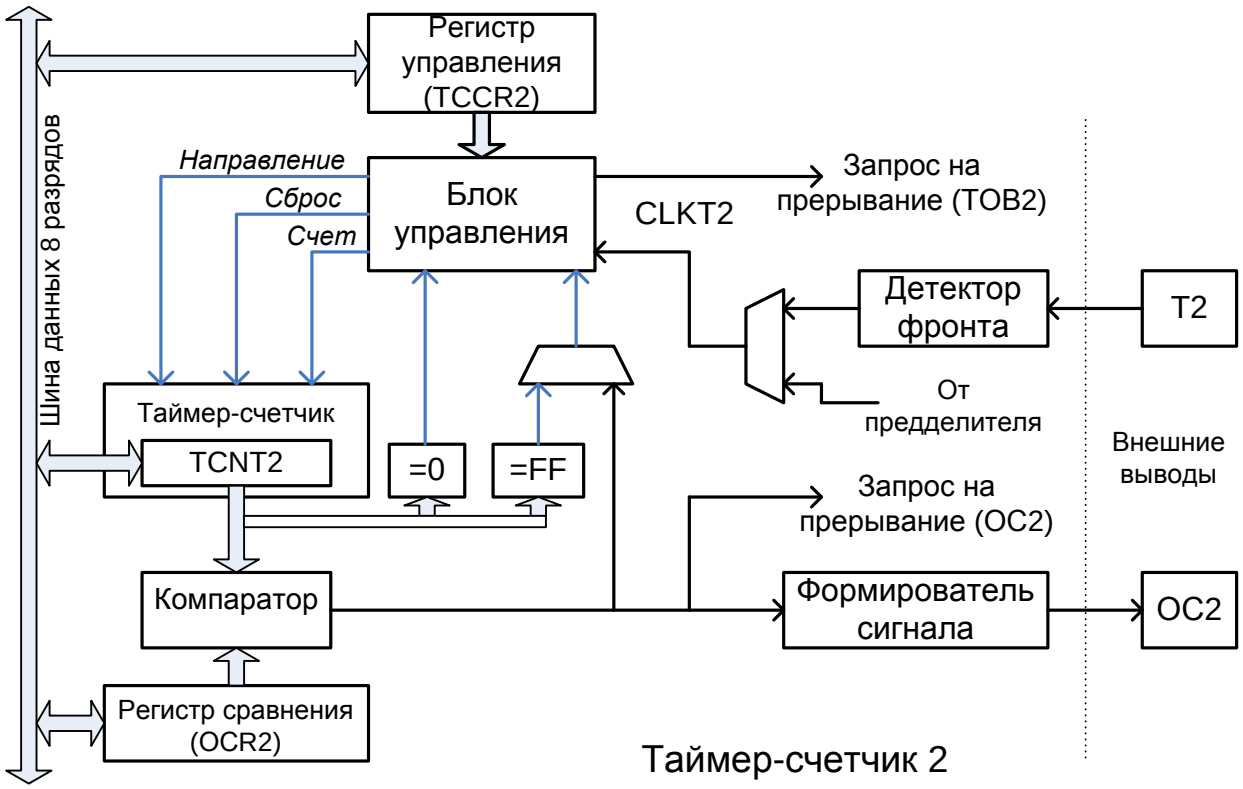


Рисунок 3.14 – Структурная схема таймера/счетчика T2

Режим "ШИМ с точной фазой" и режим "Быстрый ШИМ" предназначены для генерации сигналов с широтно-импульсной модуляцией.

Таблица 3.17 – Управление выводом OC2 в режиме CTC

COM21	COM20	Описание
0	0	Таймер-счетчик T2 отключен от вывода OC2
0	1	Состояние вывода OC2 меняется на противоположное
1	0	Вывод сбрасывается в "0"
1	1	Вывод устанавливается в "1"

Таблица 3.18 – Управление выводом OC2 в режиме CTC

CS22	CS21	CS20	Источник тактового сигнала
0	0	0	Таймер-счетчик остановлен
0	0	1	f_{CLLK}
0	1	0	$f_{CLLK}/8$
0	1	1	$f_{CLLK}/64$
1	0	0	$f_{CLLK}/256$
1	0	1	$f_{CLLK}/1024$
1	1	0	Спадающий фронт импульса со входа T2
1	1	1	Нарастающий фронт импульса со входа T2

В режиме CTC счетчик работает, так же как и в нормальном режиме, однако после достижения счетчиком значения, записанного в регистр сравнения

OCR2, счетчик сбрасывается. В таблице 3.11 представлены функции вывода *OC2* в режиме *CTC*.

Для исследования режима сравнения разработаем алгоритм и составим программу генерацию периодического сигнала заданной частоты на выводе *OC2*.

В режиме *CTC* (сброс при совпадении) частота генерации сигнала счетчиком *T2* f_{OC2} составляет

$$f_{OC2} = \frac{f_{CLK}}{2 \cdot N \cdot (1 + A_{INIT})}, \qquad (3.1)$$

где f_{CLK}=1МГц – тактовая частота микроконтроллера;

N – коэффициент деления предделителя счетчика *T2*;

A_{INT} – константа, записанная в регистр сравнения *OCP2* таймера-счетчика *T2*.

По заданным значениям f_{CLK}, f_{OC2}, *N* можно определить константу A_{INIT} из следующего выражения

$$A_{INIT} = \frac{f_{CLK}}{2 \cdot N \cdot f_{OC2}} - 1. \qquad (3.2)$$

Эмпирическим путем (провел экспериментальные исследования в программе *AVR Studio*) выбираем значение установки A_{INIT} =19 и значение предделителя *N* =1. Из выражения (3.1) определим значение частоты генерации сигнала на выводе О*С*2 – для коэффициента предделителя *N*=1, для тактовой частоты микроконтроллера *f = 1* МГц и для числа в регистре сравнения A_{INIT} =19 точное значение частоты генерации сигнала таймером-счетчиком *T2* на выводе *OC2* составит f_{OC2}=25 кГц.

3.4.8 Алгоритм программы исследования режима сравнения

Алгоритм исследования режима сравнения состоит из алгоритма основной программы и алгоритма подпрограммы прерывания по сравнению таймера-счетчика *T2*. Алгоритм основной программы представлен на рисунке 1.3.

В основной программе происходит инициализация микроконтроллера, настройка и пуск таймера *T2*.

Настройка и пуск таймера заключается в разрешении прерывания и в записи в поля регистра управления следующих значений:

– *WGM2* – [1 0] (режим *CTC*);

– *COM2* – [0 1] (состояние вывода *OC2* меняется на противоположное);

– *CS2* – [0 0 1] (источник тактовых сигналов – импульсы тактового генератора).

Рисунок 3.15 – Блок-схема алгоритма основной программы исследования режима сравнения

В регистр управления таймером *TCCR2* должно быть записано следующее число, двоичный код которого представлен в таблице 3.13.

Таблица 3.19 – Управляющий код регистра *TCCR2*

Разряд	7	6	5	4	3	2	1	0
Наименование	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
установленное значение	0	0	0	1	1	0	0	1

В шестнадцатеричном коде – 0x19.

Прерывание по сигналу сравнения разрешается путем установки разряда *OCIE2* (7-й разряд регистра масок прерывания от таймеров *TIMSK*) и в установке глобального разрешения прерывания (7-й разряд регистра состояния микроконтроллера *SREG*).

Алгоритм основной программы включает следующие блоки:

– блок инициализации параметров микроконтроллера (настройка стека, конфигурирование портов *A* и *D*, запуск сторожевого таймера (1);

– пуск сторожевого таймера (2);

– посредством записи в регистры временного хранения индицируемой информации *IND0* и *IND1* формируем два числа для поочередного их вывода через порт *A*, этот оператор может быть использован для исследования процедуры динамической индикации на два элемента индикации (3).

– настройка таймера *T2* – запись константы *AINIT*=19в регистр сравнения OCP2 таймера-счетчика *T2* (4);

– установка режима *CTC* – "сброс при совпадении", занесение константы сравнения и пуск таймера, осуществляется занесением кода 0x19 в регистр управления счетчиком *TCCR2* (5);

– разрешение глобального прерывания и прерывания от сигнала сравнения счетчика-таймера *T2*, генерируемого при достижении счетчиком значения 19, на этом настройка счетчика заканчивается (6);

– тело основной программы – бесконечный цикл со сбросом сторожевого таймера установки 0 – имитирует выполнение некоторой управляющей программы микроконтроллером (7).

Алгоритм подпрограммы обработки прерывания по сигналу сравнения таймера *T2* включает следующие блоки:

– сохранение регистра состояния в стеке (1);

– занесение в регистр вывода информации *RG_out* младшего индицируемого разряда (*RG_ind0*) для младшего семисегментного индикатора *HL2* (2)

– проверка состояния вывода *PIND*.7 порта *D* (вывод *OC2*), если он равен нулю, то в регистр вывода заносится информация для семисегментного индикатора *HL1* (оператор алгоритма 4), иначе в регистре вывода информации *RG_out* остается информация для младшего семисегментного индикатора *HL2*;

– вывод через порт *A* индицируемой информации из *RG*_out (5);

– восстановление регистра состояния из стека (6)

– возврат из подпрограммы (7).

Рисунок 3.16 – Блок-схема алгоритма подпрограммы обработки прерывания по сигналу сравнения таймера *T2*

3.4.9 Программа исследования таймера в режиме сравнения

Ниже представлен листинг программы динамической индикации, в которой таймер-счетчика *T2* работает в режиме сравнения.

```
;***** Программа исследования режима сравнения таймера T2 **********
.include "m16def.inc"
        rjmp RESET ;

.def ind_0 = r20        ;младший разряд индикации HL1
.def ind_1 = r21        ;старший разряд индикации HL2
.equ nul = 0x7E         ;семисегментный код нуля
.equ one = 0x0C         ;семисегментный код единицы
.ORG $6                 ;вектор прерывания по совпадению таймера T2
        rjmp TIMER2

RESET:
        ldi ind_0,nul  ;выводимая цифра "0"
        ldi ind_1,one  ;выводимая цифра "1"
```

```
        wdr                     ;сброс сторожевого таймера
        ldi ind_0,0x1A          ;пуск сторожевого таймера
        out WDTCR,ind_0

        ldi r31, low(ramend)    ;формирование стека
        out SPL, r31        ;в верхней части ОЗУ
        ldi r31, high(ramend)   ;формирование стека
        out SPH, r31        ;в верхней части ОЗУ

        ldi ind_0,nul       ;вводимая цифра 0
        ldi ind_1,one       ;вводимая цифра 1

        ldi r17,$FE             ;Конфигурирование порта A
        out DDRA,r17
        sbi     DDRD,7          ;и вывода D.7
        cbi PORTD,7

        ldi r17,19              ;запись Ainit=19 в r17 и OCR2
        out OCR2,r17

        ldi r17,$80             ;разрешение прерывания от таймера T0
        out TIMSK,r17
        ldi r17,$19             ;
        out     TCCR2,r17   ;пуск таймера в режиме CTC с предделителем 1
        sei                 ;установка глобального разрешения прерывания

;*************Бесконечный цикл*******************************
tcikl:
        WDR
        nop
         rjmp tcikl

;*************Подпрограмма прерывания************************

TIMER2:
        WDR
        in r16,SREG
        push r16        ;сохранение регистра состояния в стеке
        mov r17,ind_0
        in r16,PIND
        bst r16,7
        brts HL
        mov r17,ind_1
HL:
        out PORTA,r17
        pop r16
        out SREG,r16;восстановление регистра состояния из стека
```

reti
.EXIT

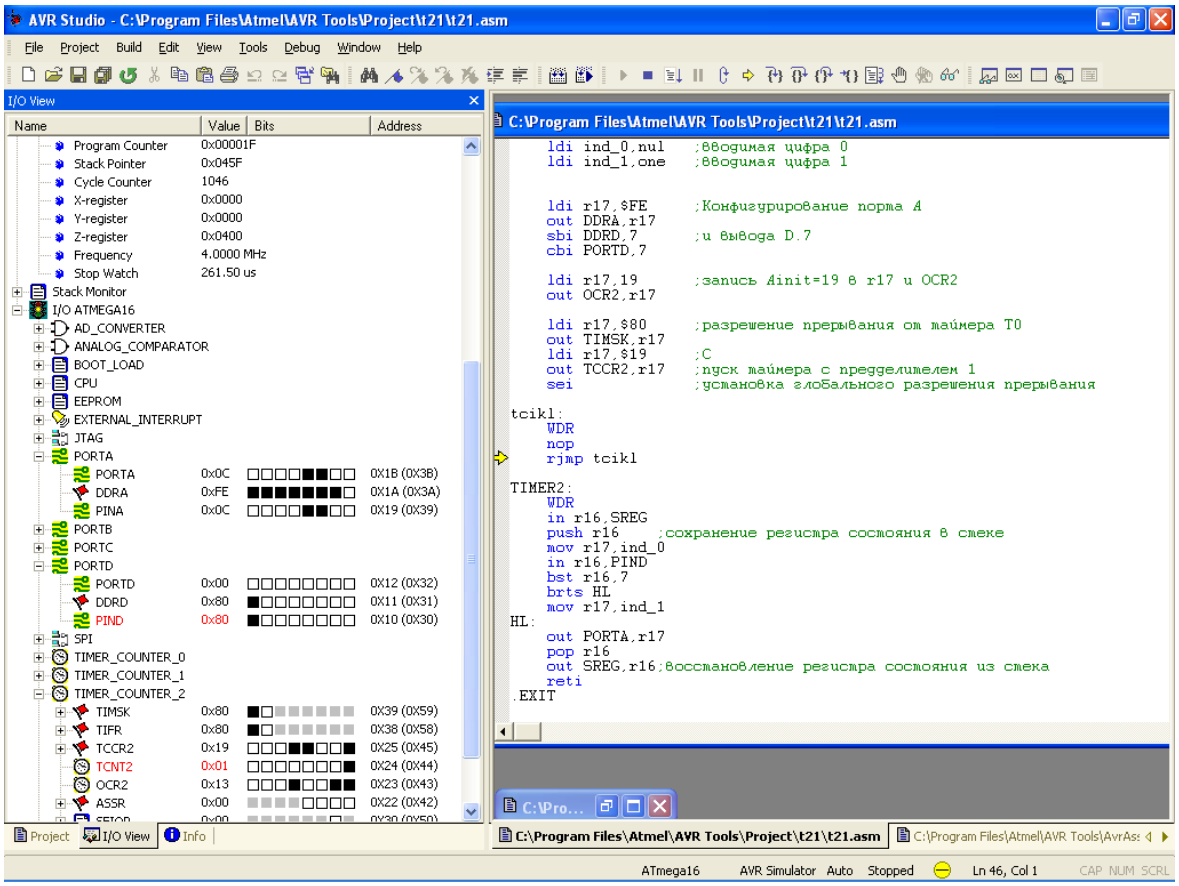


Рисунок 3.17 – Состояние узлов МК: при выводе "1"

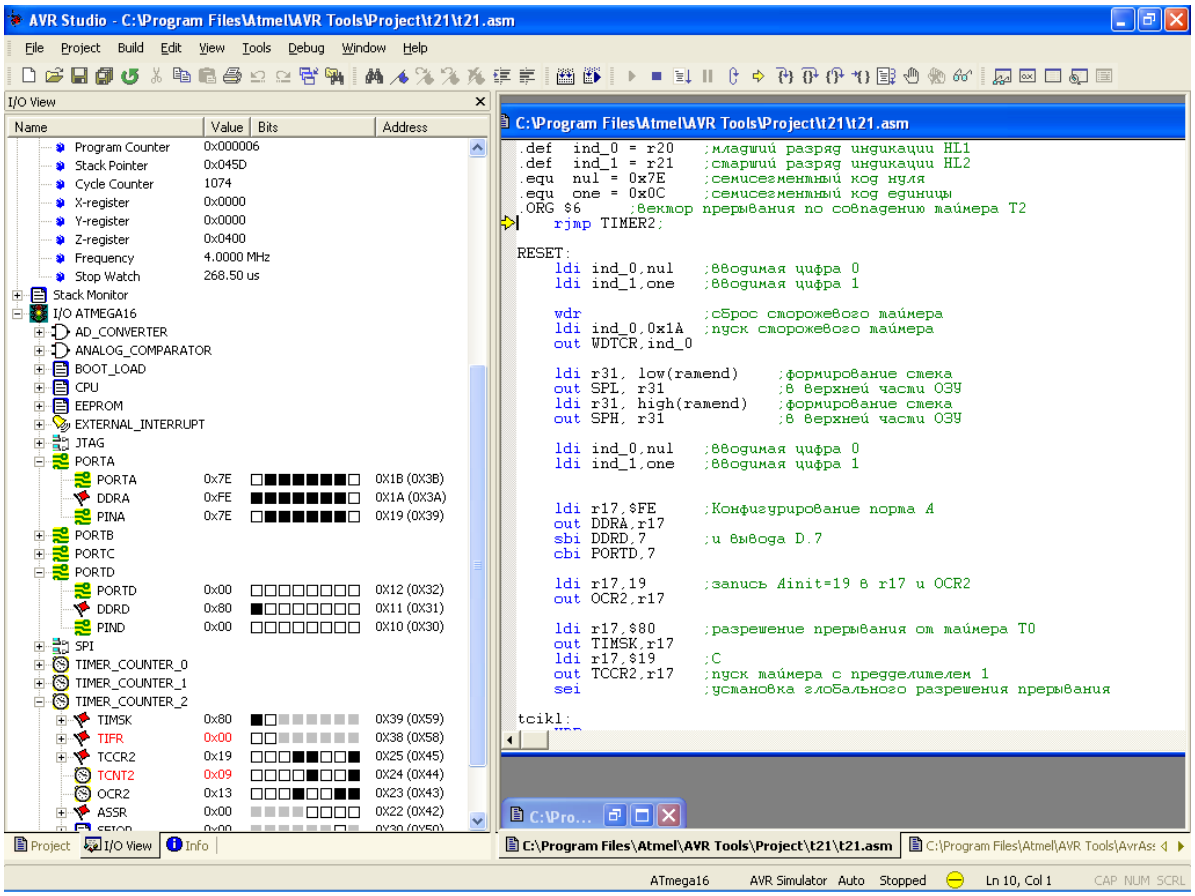


Рисунок 3.18 – Состояние узлов МК: при выводе "0"

4 ЗАДАНИЕ НА ПРОВЕДЕНИЕ ИССЛЕДОВАНИЙ

Исследование команд и отдельных узлов микроконтроллера выполняется по следующему алгоритму:

1. Изучите теоретический материал по соответствующей теме и программ исследования, а также получите задание на проведение исследования.
2. Составьте программу на ассемблере для исследования.
3. Откройте среду разработки *AVR Studio.*
4. Создайте новый проект для исследования.
5. Наберите текст программы в окне ассемблера или откройте файл с текстом программы в *AVR Studio*.
6. Откомпилируйте программу.
7. Запустите выполнение программы и пронаблюдайте за изменением содержимого регистров и портов.
8. Выведите на экран окна просмотра содержимого регистров и портов.
9. Остановите выполнение программы и сохраните проект.
10. Оформите отчет, в котором должны быть:
    - 10.1 название; цель работы;
    - 10.2 краткие теоретические сведения об объекте исследования (схемы, таблицы, логика работы);
    - 10.3 листинг программы с подробными комментариями к каждой строке программы, а также состояние задействованных регистров на каждом шаге выполнения программы;
    - 10.4 выводы по работе.

ЛИТЕРАТУРА

1. Микропроцессорные системы: Учебное пособие для вузов / Александров Е.К., Грушвицкий Р.И., Куприянов М.С. и др.; Под общ. ред . Пузанкова Д.В. – СПб.: Политехника, 2002. – 935 с.: ил.

2. Основы микропроцессорной техники / Новиков Ю.В., Скоробогатов П.К. / М.: ИНТУИТ.РУ. "Интернет-Университет Информационных технологий", 2003. – 440 с.: ил.

3. Кривченко И. AVR - микроконтроллеры: очередной этап на пути развития

4. http://www.atmel.com/atmel/products/prod23.htm

5. Ефстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы "Atmel". М.: Издательский дом "Додека – XXI", 2002. – 288 с.: ил.

6. Голубцов М.С. Микроконтроллеры AVR: от простого к сложному, М. 2003. – 456 с.

7. http://www.atmel.ru/Disks/AVR%20Technical%20Library/index.html

ПРИЛОЖЕНИЕ

Таблица П.1 – РВВ моделей *AT90S1200, AT90S2313, AT90S/LS2323 и AT90S/LS2343*

Название	Функция	Адрес		
		AT90S1200	AT90S2313	AT90S/LS2323 AT90S/LS2343
ACSR	Регистр управления и состояния аналогового компаратора	*$08*	*$08 ($28)*	
DDRB	Регистр направления данных порта B	*$17*	*$17 ($37)*	*$17 ($37)*
DDRD	Регистр направления данных порта D	*$11*	*$11 ($31)*	—
EEAR	Регистр адреса EEPROM	*$1E*	*$1E($3E)*	*$1E($3E)*
EECR	Регистр управления EEPROM	*$1C*	*$1C ДОС)*	*$1C ($30*
EEDR	Регистр данных EEPROM	*$10*	*$1D ($3D)*	*$10 ($30)*
GIFR	Общий регистр флагов прерываний	—	*$3A ($5A)*	*$3A ($5A)*
GIMSK	Общий регистр маски прерываний	*$3B*	*$3B ($5B)*	*$3B ($5B)*
ICR1H	Регистр захвата таймера/счетчика 1 (старший байт)	—	*$25 ($45)*	—
ICR1L	Регистр захвата таймера/счетчика 1 (младший байт)	—	*$24 ($44)*	—
MCUCR	Общий регистр управления микроконтроллером	*$35*	*$35 ($55)*	*$35 ($55)*
MCUSR	Регистр состояния микроконтроллера	—	—	*$34 ($54)*
OCR1AH	Регистр совпадения выхода 1 (старший байт)	—	*$2B (4B)*	—
OCRIAL	Регистр совпадения выхода 1 (младший байт)	—	*$2A (4A)*	—
PINB	Выводы порта *B*	*$15*	*$16 ($36)*	*$16 ($36)*
PIND	Выводы порта *D*	*$10*	*$10 ($30)-*	
PORTB	Регистр данных порта *B*	*$18*	*$18 ($38)*	*$18 ($38)*
PORTD	Регистр данных порта *D*	*$12*	*$12 ($32)*	—
SPL	Указатель стека	—	*$30 ($50)*	*$30 ($50)*
SREG	Регистр состояния	*$3F*	*$3F ($5F)*	*$3F ($5F)*
TCCR0	Регистр управления таймером/счетчиком 0	*$33*	*$33 ($53)*	*$33 ($53)*
TCCR1A	Регистр управления *A* таймером/счетчиком 1	—	*$2F ($4F)*	—
TCCR1B	Регистр управления *B* таймером/счетчиком 1	—	*$2E ($4E)*	—
TCNT0	Счетный регистр таймера/счетчика 0 (8-разрядный)	*$32*	*$32 ($52)*	*$32 ($52)*
TCNT1H	Счетный регистр таймера/счетчика 1	—	*$20 ($40)*	—
TCNT1L	Счетный регистр таймера/счетчика 1 (млад. байт)	—	*$2C ($4C)*	—
TIFR	Регистр флагов прерываний от таймера/счетчика	*$38*	*$38 ($58)*	*$38 ($58)*
TIMSK	Регистр маски прерываний от таймера/счетчика	*$39*	*$39 ($59)*	*$39 ($59)*
UBRR	Регистр скорости передачи *UART*	—	*$09 ($29)*	—
UCR	Регистр управления *UART*	—	*$OA ($2A)*	—
UDR	Регистр данных *UART*	—	*$OC ДОС)*	—
USR	Регистр состояния *UART*	—	*$OB ($2B)*	—
WDTCR	Регистр управления сторожевым таймером	*$21*	*$21 ($41)*	

Таблица П.2 – РВВ моделей *AT90S/LS2333* и *AT90S/LS4433*

Название	Функция	Адрес
		AT90S/LS2333 *AT90S/LS4433*
ACSR	Регистр управления и состояния аналогового компаратора	*$08 ($28)*
ADCH	Регистр данных АЦП (старший байт)	*$05 ($25)*
ADCL	Регистр данных АЦП (младший байт)	*$04 ($24)*
ADCSR	Регистр управления и состояния АЦП	*$06 ($26)*
ADMUX	Регистр управления мультиплексором АЦП	*$07 ($27)*
DDRB	Регистр направления данных порта *B*	*$17 ($37)*
DDRC	Регистр направления данных порта *C*	*$14 ($37)*
ODRD	Регистр направления данных порта *D*	*$11 ($31)*
EEAR	Регистр адреса *EEPROM*	*$1E($3E)*
EECR	Регистр управления *EEPROM*	*$1C($3C)*
EEDR	Регистр данных *EEPROM*	*$10 ($30)*
GIFR	Общий регистр флагов прерываний	*$3A (S5A)*
GIMSK	Общий регистр маски прерываний	*$3B ($5B)*
ICR1H	Регистр захвата таймера/счетчика 1 (старший байт)	*$27 ($47)*
1CR1L	Регистр гахвата таймера/счетчика 1 (младший байт)	*$26 ($46)*
MCUCR	Общий регистр управления микроконтроллером	*$35 ($55)*
OCR1H	Регистр совпадения выхода 1 (старший байт)	*$2B ($4B)*
OCR1L	Регистр совпадения выхода 1 (младший байт)	*$2A ($4A)*
P1NB	Выводы порта *B*	*$16 ($36)*
P1NC	Выводы порта *C*	*$13 ($36)*
PIND	Выводы порта *D*	*$10 ($30)*
PORTB	Регистр данных порта *B*	*$18 ($38)*
PORTC	Регистр данных порта *C*	*$15 ($38)*
PORTD	Регистр данных порта *D* 0	*$12 ($32)*
SP	Указатель стека	*$30 ($50)*
SPCR	Регистр управления *SPI*	*$00 ($20)*
SPDR	Регистр данных *SPI*	*$OF ($2F)*
SPSR	Регистр состояния *SPI*	*$OE ($2E)*
SREG	Регистр состояния	*$3F ($5F)*
TCCRO	Регистр управления таймером/счетчиком 0	*$33 ($53)*
TCCR1A	Регистр управления *A* таймером/счетчиком 1	*$2F ($4F)*
TCCR1B	Регистр управления *B* таймером/счетчиком 1	*$2E ($4E)*
TCNTO	Счетный регистр таймера/счетчика 0 (8-разрядный)	*$32 ($52)*
TCNT1H	Счетный регистр таймера/счетчика 1	*$20 ($40)*
TCNT1L	Счетный регистр таймера/счетчика 1 (младший байт)	*$2C ($4C)*
T1FR	Регистр флагов прерываний от таймера/счетчика	*$38 ($58)*
TIMSK	Регистр маски прерываний от таймера/счетчика	*$39 ($59)*
UBRR	Регистр скорости передачи *UART* (младший байт)	*$09 ($29)*
UBRRHI	Регистр скорости передачи *UART* (старший байт)	*$03 ($23)*
UCR	Регистр управления *UART*	*$OA ($2A)*
UDR	Регистр данных *UART*	*$OC ($20)*
USR	Регистр состояния *UART*	*$OB ($2B)*
WDTR	Регистр управления сторожевым таймером	*$21 ($41)*

Таблица П.3 – РВВ моделей *AT90S/LS4434, AT90S/LS8535, AT90S4414* и *AT90S8515*

Назва-ние	Функция	Адрес	
		AT90S/LS4434 AT90S/LS8535	*AT90S4414 AT90S8515*
1	2	*3*	*4*
ACSR	Регистр управления и состояния аналог. компаратора	*$08 ($28)*	*$08 ($28)*
ADCH	Регистр данных АЦП (старший байт)	*$05 ($25)*	—
ADCL	Регистр данных АЦП (младший байт)	*$04 ($24)*	—
ADCSR	Регистр управления и состояния АЦП	*$06 ($26)*	—
ADMUX	Регистр управления мультиплексором АЦП	*$07 ($27)*	—
ASSR	Регистр состояния асинхронного режима	*$22 ($42)*	—
DDRA	Регистр направления данных порта *A*	*$1A($3A)*	*$1A($3A)*
DDRB	Регистр направления данных порта *B*	*$17 ($37)*	*$17 ($37)*
DDRC	Регистр направления данных порта *C*	*$14 ($37)*	*$14 ($37)*
ODRD	Регистр направления данных порта *D*	*$11 ($31)*	*$11 ($31)*
EEARH	Регистр адреса *EEPROM* (старший байт)	*$1F($3F)*	*$1F($3F)*
EEARL	Регистр адреса *EEPROM* (младший байт)	*$1E($3E)*	*$1E($3E)*
EECR	Регистр управления *EEPROM*	*$1C($3C)*	*$1C($3C)*
EEDR	Регистр данных *EEPROM*	*$1D ($3D)*	*$1D ($3D)*
GIFR	Общий регистр флагов прерываний	*$3A ($5A)*	*$3A ($5A)*
GIMSK	Общий регистр маски прерываний	*$3B ($5B)*	*$3B ($5B)*
ICR1H	Регистр захвата таймера/счетчика 1 (стар. байт)	*$27 ($47)*	*$25 ($45)*
1CR1L	Регистр захвата таймера/счетчика 1 (млад. байт)	*$26 ($46)*	*$24 ($44);*
MCUCR	Общий регистр управления микроконтроллером	*$35 ($55)*	*$35 ($55)*
MCUSR	Регистр состояния микроконтроллера	*$34 ($54)*	
OCRIAH	Регистр совпадения выхода *A* (старший байт)	*$2B ($4B)*	*$2B ($4B)*
OCR1AL	Регистр совпадения выхода *A* (младший байт)	*$2A ($4A)*	*$2A ($4A)*
OCRIBH	Регистр совпадения выхода *B* (старший байт)	*$29 ($49)*	*$29 ($49)*
OCRIBL	Регистр совпадения выхода *B* (младший байт)	*$28 ($48)*	*$28 ($48)*
OCR2	Регистр совпадения выхода таймера/счетчика 2	*$23 ($43)*	—
P1NA	Выводы порта *A*	*$19 ($39)*	*$19 ($39)*
PINB	Выводы порта *B*	*$16 ($36)*	*$16 ($36)*
PINC	Выводы порта *C*	*$13 ($36)*	*$13 ($36)*
PIND	Выводы порта *D*	*$10 ($30)*	*$10 ($30)*
PORTA	Регистр данных порта *A*	*$1B ($3B)*	*$1B ($3B)*
PORTB	Регистр данных порта *B*	*$18 ($38)*	*$18 ($38)*
PORTC	Регистр данных порта *C*	*$15 ($38)*	*$15 ($38)*
PORTD	Регистр данных порта *D*	*$12 ($32)*	*$12 ($32)*
SPCR	Регистр управления *SPI*	*$0D ($2D)*	*$0D ($2D)*
SPDR	Регистр данных *SPI*	*$0F ($2F)*	*$0F ($2F)*
SPH	Указатель стека (старший байт)	*$3E ($5E)*	*$3E ($5E)*
SPL	Указатель стека (младший байт)	*$3D ($5D)*	*$3D ($5D)*
SPSR	Регистр состояния *SPI*	*$0E ($2E)*	*$0E ($2E)*
SREG	Регистр состояния	*$3F ($5F)*	*$3F ($5F)*
TCCRO	Регистр управления таймером/счетчиком 0	*$33 ($53)*	*$33 ($53)*
TCCR1A	Регистр управления *A* таймером/счетчиком 1	*$2F ($4F)*	*$2F ($4F)*
TCCR1B	Регистр управления *B* таймером/счетчиком 1	*$2E ($4E)*	*$2E ($4E)*
TCCR2	Счетный регистр таймера/счетчика 2	*$25 ($45)*	—
TCNTO	Счетный регистр таймера/счетчика 0 (8-разрядный)	*$32 ($52)*	*$32 ($52)*

Продолжение таблицы П.3

1	2	*3*	*4*
TCNT1H	Счетный регистр таймера/счетчика 1	*$2D (S4D)*	*$2D ($4D)*
TCNT1L	Счетный регистр таймера/счетчика 1 (младший байт)	*$2C (S4C)*	*$2C ($4C)*
TCNT2	Счетный регистр таймера/счетчика 2 (8-разрядный)	*$24 ($44)*	—
TIFR	Регистр флагов прерываний от таймера/счетчика	*$38 ($58)*	*$38 ($58)*
TIMSK	Регистр маски прерываний от таймера/счетчика	*$39 ($59)*	*$39 ($59)*
UBRR	Регистр скорости передачи *UART*	*$09 ($29)*	*$09 ($29)*
UCR	Регистр управления UART	*$0A ($2A)*	*$0A ($2A)*
UDR	Регистр данных *UART*	*$0C ($2C)*	*S0C ($2C)*
USR	Регистр состояния *UART*	*$0B ($2B)*	*S0B ($2B)*
WDTCR	Регистр управления сторожевым таймером	*$21 ($41)*	*$21 ($41)*

Таблица П.4 – РВВ модели *AT90C8534*

Назва-ние	Функция	Адрес
		AT90C8534
ADCH	Регистр данных АЦП (старший байт)	*$05 ($25)*
ADCL	Регистр данных АЦП (младший байт)	*$04 ($24)*
ADCSR	Регистр управления и состояния АЦП	*$06 ($26)*
ADMUX	Регистр управления мультиплексором АЦП	*$07 ($27)*
DORA	Регистр направления данных порта *A*	*$1A ($3A)*
EEARH	Регистр адреса *EEPROM* (старший байт)	*$1F($3F)*
EEARL	Регистр адреса *EEPROM* (младший байт)	*$1E($3E)*
EECR	Регистр управления *EEPROM*	*$1C($3C)*
EEDR	Регистр данных *EEPROM*	*$10 ($30)*
GIFR	Общий регистр флагов прерываний	*$3A ($5A)*
G1MSK	Общий регистр маски прерываний	*$3B ($5B)*
G1PR	Регистр входов внешних прерываний	*$10 ($30)*
MCUCR	Общий регистр управления микроконтроллером	*$35 ($55)*
PORTA	Регистр данных порта *A*	*$1B($3B)*
SPH	Указатель стека (старший байт)	*$3E ($5E)*
SPL	Указатель стека (младший байт)	*$30 ($50)*
SPEG	Регистр состояния	*$3F ($5F)*
TCCRO	Регистр управления таймером/счетчиком 0	*$33 ($53)*
TCCR1	Регистр управления таймером/счетчиком 1	*$2E ($4E)*
TCNTO	Счетный регистр таймера/счетчика 0 (8-разрядный)	*$32 ($52)*
TCNT1H	Счетный регистр таймера/счетчика 1	*$20 ($40)*
TCNT1L	Счетный регистр таймера/счетчика 1 (младший байт)	*S2C ($4C)*
TIFR	Регистр флагов прерываний от таймера/счетчика	*S38($58)*
TIMSK	Регистр маски прерываний от таймера/счетчика	*$39 ($59)*

Таблица П.5 – Регистр *MCUCR* моделей *AT90S1200, AT90S2313, AT90S/LS2323, AT90S/LS2343, AT90S/LS2333, AT90S/LS4433, AT90S4414* и *AT90S8515*

<table>
<tr><th>Разр</th><th>Назв.</th><th>Описание</th><th>Модель</th></tr>
<tr><td>7</td><td>SRE</td><td>Разрешение внешнего ОЗУ. Если этот разряд установлен в "1", использование внешнего ОЗУ разрешено. В противном случае использование внешнего ОЗУ запрещено, и выводы, используемые для подключения внешнего ОЗУ, функционируют как обычные линии ввода/вывода.</td><td>AT90S4414
AT90S8515</td></tr>
<tr><td></td><td>—</td><td>Не используется, читается как "0".</td><td>Прочие</td></tr>
<tr><td>6</td><td>SRW</td><td>Режим обращения к внешнему ОЗУ. Если этот разряд установлен в "1", обращение к внешнему ОЗУ выполняется за 4 машинных цикла (с одним циклом ожидания). Если этот разряд сброшен, обращение к внешнему ОЗУ выполняется за 3 машинных цикла.</td><td>AT90S4414
AT90S8515</td></tr>
<tr><td></td><td>—</td><td>Не используется, читается как "0"</td><td>Прочие</td></tr>
<tr><td>5</td><td>SE</td><td>Разрешение перехода в режим пониженного энергопотребления. Если этот разряд установлен в "1", то по команде "SLEEP" микроконтроллер переходит в "спящий" режим</td><td>Все модели</td></tr>
<tr><td>4</td><td>SM</td><td>Выбор режима пониженного энергопотребления. Состояние этого разряда определяет, в какой режим перейдет микроконтроллер после выполнения команды "SLEEP". Если этот разряд установлен в "1", "спящим" режимом является режим "Power Down". Если этот разряд сброшен — режим "Idle".</td><td>Все модели</td></tr>
<tr><td>3,2</td><td>—</td><td>Не используются, читаются как "0"</td><td>AT90S1200
AT90S2323
AT90S2343</td></tr>
<tr><td></td><td>ISC11, ISC10</td><td>Определяют условие генерации внешнего прерывания INT1 следующим образом:
ISC11 | ISC10 | Условие
0 | 0 | по НИЗКОМУ уровню на выводе 1NT1
0 | 1 | при любом изменении уровня на выводе 1NT1 (для AT90S2333/4433); зарезервировано для остальных
1 | 0 | по спадающему фронту сигнала на выводе INT1
1 | 1 | по нарастающему фронту сигнала на выводе INT!</td><td>Прочие</td></tr>
<tr><td>1,0</td><td>ISC01ISC00</td><td>Определяют условие генерации внешнего прерывания INTO следующим образом:
ISC01 | ISC00 | Условие
0 | 0 | по НИЗКОМУ уровню на выводе INTO
0 | 1 | при любом изменении уровня на выводе INTO (для AT90S2333/4433); зарезервировано для остальных
1 | 0 | по спадающему фронту сигнала на выводе INTO
1 | 1 | по нарастающему фронту сигнала на выводе INTO</td><td>Все модели</td></tr>
</table>

Таблица П.6 – Регистр *MCUCR* моделей *AT90S/LS4434* и *AT90S/LS8535*

Разр.	Назв.	Описание
7	—	Не используется, читается как "0"
6	*SE*	Разрешение перехода в режим пониженного энергопотребления. Если этот разряд установлен в "1", то по команде "*SLEEP*" микроконтроллер переходит в "спящий" режим.

Продолжение таблицы П.6.

Разр.	Назв.	Описание		
5,4	*SM1, SM0*	Выбор режима пониженного энергопотребления. Состояние этих разрядов определяет, в какой режим перейдет микроконтроллер после выполнения команды "*SLEEP*".		
		SM1	*SMO*	Режим
		0	0	*Idle*
		0	1	зарезервировано
		1	0	*Power Down*
		1	1	*Power Save*
3,2	*ISC11, ISC10*	Определяют условие генерации внешнего прерывания *INT1* следующим образом:		
		ISC11	*ISC10*	Условие
		0	0	по НИЗКОМУ уровню на выводе *INT1*
		0	1	зарезервировано
		1	0	по спадающему фронту сигнала на выводе *INT1*
		1	1	по нарастающему фронту сигнала на выводе *INT1*
1,0	*ISCOI, ISCOO*			
		ISC01	ISC00	Условие
		0	0	по НИЗКОМУ уровню на выводе *INT0*
		0	1	зарезервировано
		1	0	по спадающему фронту сигнала на выводе *INT0*
		1	1	по нарастающему фронту сигнала на выводе *INT0*

Таблица П.7 – Регистр MCUCR модели AT90C8534

Разр.	Назв.	Описание
	—	Не используется, читается как "0"
6	SE	Разрешение перехода в режим пониженного энергопотребления. Если этот разряд установлен в "1", то по команде "*SLEEP*" микроконтроллер переходит в "спящий" режим
5	SM	Выбор режима пониженного энергопотребления. Состояние этого разряда определяет, в какой режим перейдет микроконтроллер после выполнения команды "*SLEEP*". Если этот разряд установлен в "*I*", "спящим" режимом является режим "*Power Down*". Если этот разряд сброшен — режим "*Idle*".
4,3	—	Не используются, читаются как "0"
2	ISC1	Определяет условие генерации внешнего прерывания *INT1*. Если этот разряд установлен в "1", прерывание генерируется по нарастающему фронту сигнала на выводе *INT1*. Если этот разряд сброшен, прерывание генерируется по спадающему фронту сигнала на выводе *INT1* . Генерация прерывания гарантируется для импульсов длительностью не менее 40 нс
1	—	-1е используется, читается как "0"
0	1SCO	Определяет условие генерации внешнего прерывания *INTO*. Если этот разряд установлен в "1", прерывание генерируется по нарастающему фронту сигнала на выводе *INTO*. Если лог разряд сброшен, прерывание генерируется по спадающему фронту сигнала на выводе *INTO*. Генерация прерывания гарантируется для импульсов длительностью не менее 40 нс.

При изменении состояния разрядов ISC1 и ISCO возможна ложная генерация соответствующего прерывания. Чтобы этого избежать, рекомендуется следующая последовательность действий:

- *запретить прерывание, соответствующее изменяемому разряду;*
- *изменить состояние разряда;*
- *сбросить флаг прерывания;*
- *разрешить прерывание.*

Таблица П.8 – Команды МК *AVR* семейства *Classic*

Мнемоника	Описание	Операция	Циклы	Флаги
1	2	3	4	5
Группа команд логических операций				
AND Rd, Rr	"Логическое И" двух РОН	*Rd = Rd & Rr*	1	*Z,N,V*
ANDI Rd, K	"Логическое И" РОН и константы	*Rd = Rd & K*	1	*Z,N,V*
EOR Rd, Rr	"Исключающее ИЛИ" двух РОН	*Rd = Rd ⊕ Rr*	1	*Z,N,V*
OR Rd, Rr	"Логическое ИЛИ" двух РОН	*Rd = Rd ∨ Rr*	1	*Z,N,V*
ORI Rd, K	"Логическое ИЛИ" РОН и константы	*Rd = Rd ∨ K*	1	*Z,N,V*
COM Rd	Перевод в обратный код	*Rd = $FF - Rd*	1	*Z,C,N,V*
NEG Rd	Перевод в дополнительный код	*Rd = $00-Rd*	1	*Z,C,N,V,H*
CLR Rd	Сброс всех разрядов РОН	*Rd = Rd ⊕ Rd*	1	*Z,N,V*
SER Rd	Установка всех разрядов РОН	*Rd = $FF*	1	—
TST Rd	Проверка РОН на отрицательное или нулевое значение	*Rd&Rd*	1	*Z,N,V*
SWAP Rd	Обмен местами тетрад в РОН	*Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)*	1	—
Группа команд арифметических операций				
ADD Rd, Rr	Сложение двух РОН	*Rd = Rd + Rr*	1	*Z,C,N,V,H*
ADC Rd, Rr	Сложение двух РОН с переносом	*Rd = Rd + Rr + C*	1	*Z,C,N,V,H*
ADIW Rd, K	Сложение регистровой пары с константой	*Rdh:Rdi=Rdh:Rdl+K*	2	*Z,C,N,V,S*
SUB Rd, Rr	Вычитание двух РОН	*Rd = Rd - Rr*	1	*Z,C,N,V,H*
SUBI Rd, K	Вычитание константы из РОН	*Rd = Rd - K*	1	*Z,C,N,V,H*
SBC Rd, Rr	Вычитание двух РОН с заемом	*Rd = Rd – Rr - C*	]	*Z,C,N,V,H*
SBCI Rd, K	Вычитание константы из РОН с заемом	*Rd = Rd – K - C*	1	*Z,C,N,V,H*
SBIW Rd,	Вычитание константы из регистровой пары	*Rdh:Rdl=Rdh:Rdl-K*	2	*Z,C,N,V,S*
DEC Rd	Декремент РОН	*Rd = Rd - l*	1	*Z,N,V*
INC Rd	Инкремент РОН	*Rd = Rd + 1*	1	*Z,N,V*
ASR Rd	Арифметический сдвиг вправо	*Rd(n) = Rd(n+1), n=0..6*	1	*Z,C,N,V*
LSL Rd	Логический сдвиг влево	*Rd(n+l) = Rd(n), Rd(0) = 0*	1	*Z,C,N,V*
LSR Rd	Логический сдвиг вправо	*Rd(n) = Rd(n+l), Rd(7) = 0*	1	*Z,C,N,V*
ROL Rd	Сдвиг влево через перенос	*Rd(0) = C, Rd(n+l)= = Rd(n), C = Rd(7)*	1	*Z,C,N,V*
ROR Rd	Сдвиг вправо через перенос	*Rd(7) = C, Rd(n) = =Rd(n+l), C = Rd(0)*	1	*Z,C,N,V*
Группа команд операций с разрядами				
CBR Rd, K	Сброс разряда(ов) РОН	*Rd = Rd & ($FF - K)*	1	*Z,N,V*
SBR Rd, K	Установка разряда(ов) РОН	*Rd = Rd ∨ K*	1	*Z,N,V*
CBI A,b	Сброс разряда РВВ	*A.b = 0*	2	—
SBI A,b	Установка разряда РВВ	*A.b=1*	2	—
BCLR s	Сброс флага	*SREG.s = 0*	1	*SREG.s*
BLD Rd, b	Загрузка разряда РОН из флага T (SREG)	*Rd.b = T*	1	—
BSET s	Установка флага	*SREG.s = 1*	1	*SREG.s*
BST Rr, b	Запись разряда РОН в флаг T (SREG)	*T = Rr.b*	1	*T*

Продолжение таблицы П.8

1	2	3	4	5
CLC	Сброс флага переноса	*C = 0*	1	*C*
SEC	Установка флага переноса	*C=l*	1	*C*
CLN	Сброс флага отр. числа	*N = 0*	1	*N*
SEN	Установка флага отр. числа	*N=1*	1	*N*
CLZ	Сброс флага нуля	*Z = 0*	1	*Z*
SEZ	Установка флага нуля	*Z=l*	1	*Z*
CLI	Общее запрещение прерываний	*I = 0*	1	*I*
SEI	Общее разрешение прерываний	*I = 1*	1	*I*
CLS	Сброс флага знака	*S = 0*	1	*S*
SES	Установка флага знака	*S=l*	1	*S*
CLV	Сброс флага переполнения доп. кода	*V = 0*	1	*V*
SEV	Установка флага переполнения доп. кода	*V= 1*	1	*V*
CLT	Сброс флага Т	*T = 0*	1	*T*
SET	Установка флага Т	*T= 1*	1	*T*
CLH	Сброс флага половинного переноса	*H = 0*	1	*H*
SEH	Установка флага половинного переноса	*H=1*	1	*H*
Группа команд пересылки данных				
MOV Rd. Rr	Пересылка между РОН	*Rd = Rr*	1	
LDI Rd, K	Загрузка константы в РН	*Rd = K*	1	—
LD Rd, X	Косвенное чтение	*Rd = [X]*	2	—
LD Rd, X+	Косвенное чтение с постинкрементом	*Rd = [X], X = X + 1*	2	—
LD Rd, -X	Косвенное чтение с преддекрементом	*X = X-l, Rd = [X]*	2	—
LD Rd, Y	Косвенное чтение	*Rd=[Y]*	2	—
LD Rd, Y+	Косвенное чтение с постинкрементом	*Rd = [Y], Y = Y + 1*	2	—
LD Rd, -Y	Косвенное чтение с преддекрементом	*Y = Y-l, Rd = [Y]*	2	—
LDD Rd, Y+q	Косвенное относительное чтение	*Rd = [Y+q]*	2	—
LD Rd, Z	Косвенное чтение	*Rd = [Z]*	2	—
LD Rd, Z +	Косвенное чтение с постинкрементом	*Rd = [Z], Z = Z + 1*	2	—
LD Rd,-Z	Косвенное чтение с преддекрементом	*Z = Z-l, Rd = [Z]*	2	—
LDD Rd, Z+q	Косвенное относительное чтение	*Rd = [Z+q]*	2	—
LDS Rd, k	Непосредственное чтение из ОЗУ	*Rd = [k]*	2	—
ST X, Rr	Косвенная запись	*[X] = Rr*	2	—
ST X+, Rr	Косвенная запись с постинкрементом	*[X] = Rr, X = X +1*	2	—
ST -X, Rr	Косвенная запись с преддекрементом	*X = X-l, [X] = Rr*	2	
ST Y, Rr	Косвенная запись	*[Y] = Rr*	2	—
ST Y+, Rr	Косвенная запись с постинкрементом	*[Y] = Rr, Y = Y + 1*	2	—
ST -Y, Rr	Косвенная запись с преддекрементом	*Y = Y- l,[Y] = Rr*	2	—
STD Y+q, Rr	Косвенная относительная запись	*[Y+q] = Rr*	2	—
ST Z, Rr	Косвенная запись	*[Z] = Rr*	2	—
ST Z+, Rr	Косвенная запись с постинкрементом	*[Z] = Rr, Z = Z + 1*	2	—
ST -Z, Rr	Косвенная запись с преддекрементом	*Z = Z-l,[Z] = Rr*	2	—
STS k, Rr	Непосредственная запись в ОЗУ	*[k] = Rr*	2	
STD Z+q, Rr	Косвенная относительная запись	*[Z+q] = Rr*	2	—
LPM	Загрузка данных из памяти программ	*R0={Z}*	3	—
IN Rd, A	Пересылка из РВВ в РОН	*Rd = A*	1	—
OUT A, Rr	Пересылка из РОН в РВВ	*A = Rr*	1	—
PUSH Rr	Сохранение байта в стеке	*STACK = Rr*	2	—
POP Rd	Извлечение байта из стека	*Rd = STACK*	2	

Продолжение таблицы П.8

1	2	3	4	5
Группа команд передачи управления				
RJMP k	Относительный безусловный переход	*PC = PC + k +1*	2	—
IJMP	Косвенный безусловный переход	*PC = Z*	2	—
RCALL	Относительный вызов подпрограммы	*PC = PC + k +1*	3	—
ICALL	Косвенный вызов подпрограммы	*PC = Z*	3	—
RET	Возврат из подпрограммы	*PC = STACK*	4	—
RETI	Возврат из подпрограммы обработки прерывания	*PC = STACK*	4	*I*
CP Rd, Rr	Сравнение РОН	*Rd-Rr*	1	*Z,N,V,C,H*
CPC Rd, Rr	Сравнение РОН с учетом переноса	*Rd-Rr-C*	1	*Z,N,V,C,H*
CPI Rd, K	Сравнение РОН с константой	*Rd-K*	1	*Z,N,V,C,H*
CPSE Rd, Rr	Сравнение и пропуск следующей команды при равенстве	Если *Rd = Rr*, то *PC= PC + 2 (3)*	1/2/3	—
SBRC Rr, b	Пропуск след, команды, если разряд РОН сброшен	Если *Rr.b = 0*, то *PC = PC + 2 (3)*	1/2/3	—
SBRS Rr, b	Пропуск след, команды, если разряд РОН установлен	Если *Rr.b = 1*, то *PC = PC + 2 (3)*	1/2/3	—
SBIC A, b	Пропуск след, команды, если разряд РВВ сброшен	Если *A.b = 0*. то *PC = PC + 2 (3)*	1/2/3	*I*
SBIS A, b	Пропуск след, команды, если разряд РВВ установлен	Если *A.b = 1*, то *PC = PC + 2 (3)*	1/2/3	—
BRBC s, k	Переход, если флаг s регистра SREG сброшен	Если *SREG.s = 0*, то *PC = PC + k + l*	1/2	—
BRBS s, k	Переход, если флаг s регистра SREG установлен	Если *SREG.s =1*, то *PC = PC + k + l*	1/2	—
BRCS k	Переход по переносу	Если *C =1*, то *PC = PC + k +1*	1/2	—
BRCC k	Переход, если нет переноса	Если *C = 0*, то *PC = PC + k + 1*	1/2	—
BREQ k	Переход по условию "равно"	Если *Z = 1*, то *PC = PC + k +1*	1/2	—
BRNE k	Переход по условию "не равно"	Если *Z = 0*, то *PC = PC + k + 1*	1/2	—
BRSH k	Переход по условию "выше или равно"	Если *C = 0*, то *PC = PC + k + 1*	1/2	—
BRLO k	Переход по условию "меньше"	Если *C = 1*, то *PC = PC + k + 1*	1/2	—
BRMI	Переход по условию "отрицательное значение"	Если *N = 1*, то *PC = PC + k + 1*	1/2	—
BRPL	Переход по условию "положительное значение"	Если *N = 0*, то *PC = PC + k + 1*	1/2	—
BRGE	Переход по условию "больше или равно" (числа со знаком)	Если (*N ⊕V*) *= 0*, то *PC = PC + k + 1*	1/2	—
BPLT	Переход по условию "меньше нуля" (числа со знаком)	Если (*N⊕V*)=*l*, то *PC = PC + k + 1*	1/2	—
BRHS	Переход по половинному переносу	Если *H=1*, то *PC = PC + k + 1*	1/2	—
BRHC	Переход, если нет половинного переноса	Если *H = 0*, то *PC = PC + k + 1*	1/2	—

Продолжение таблицы П.8

1	2	3	4	5
BRTS	Переход, если флаг T установлен	Если *T= 1*, то *PC = PC + k + 1*	1/2	—
BRTC	Переход, если флаг T сброшен	Если *T = 0*, то *PC = PC + k + 1*	1/2	—
BRVS	Переход по переполнению доп. кода	Если *V=1*, то *PC = PC + k + 1*	1/2	—
BRVC	Переход, если нет переполнения доп. кода	Если *V = 0*, то *PC = PC + k + 1*	1/2	—
BRID	Переход, если прерывания запрещены,	Если *I = 0*, то *PC = PC + k + 1*	1/2	—
BRIE	Переход, если прерывания разрешены	Если *I= 1*, то *PC = PC + k + 1*	1/2	
NOP	Нет операции		1	
SLEEP	Переход в "спящий" режим		3	—
WDR	Сброс сторожевого таймера		1	

1 Моисей Гельман, Вадим Михневич**.** Российская авионика нуждается в земной опоре// Промышленные ведомости, №2(3) 11-20 марта 2000 г.
2 Чурилин Л. «Кот», который живёт на палубе. Морской тяжёлый истребитель-перехватчик F-14A (рус.) // **Крылья Родины**. — М.: 1998. — № 12. — С. 17-21.
3 Микропроцессорные системы: Учебное пособие для вузов / Александров Е.К., Грушвицкий Р.И., Куприянов М.С. и др.; Под общ. ред . Пузанкова Д.В. – СПб.: Политехника, 2002. – 935 с.: ил.
4 Кривченко И. AVR - микроконтроллеры: очередной этап на пути развития
5 http://www.atmel.com/atmel/products/prod23.htm
6 Ефстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы "Atmel". М.: Издательский дом "Додека – XXI", 2002. – 288 с.: ил.
7 Голубцов М.С. Микроконтроллеры AVR: от простого к сложному, М. 2003. – 456 с.
8 http://www.atmel.ru/Disks/AVR%20Technical%20Library/index.html