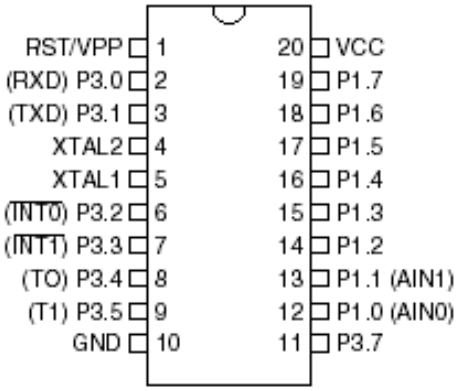
А.В. Шарапов

ОСНОВЫ МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ

Учебное пособие



ТОМСК – 2008

Федеральное агентство по образованию

Томский государственный университет систем
управления и радиоэлектроники

А. В. Шарапов

ОСНОВЫ МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ

Учебное пособие

Рекомендовано Сибирским региональным отделением
учебно-методического объединения высших учебных заведений РФ
по образованию в области радиотехники, электроники, биомедицинской
техники и автоматизации для межвузовского использования в качестве
учебного пособия для студентов радиотехнических специальностей

2008

Рецензенты: зав. кафедрой промышленной и медицинской электроники Томского политехнического университета, д-р техн. наук, проф. Г.С. Евтушенко;
начальник отдела ФГУП «НПЦ «Полюс»,
д-р техн. наук Ю.М. Казанцев

Шарапов А.В.
Основы микропроцессорной техники: Учебное пособие. – Томск: ТУСУР, 2008. – 240 с.

В первой части приведены многочисленные примеры программ обработки данных для микроконтроллеров семейства МК51. Рассмотрены характеристики микроконтроллеров фирмы Atmel с ядром MCS-51.

Во второй части рассмотрены программная модель, система команд и характеристики периферийных устройств микроконтроллеров AVR фирмы Atmel семейств Tiny и Mega. Показано использование отладчика AVR Studio, компилятора CVAVR и симуляторов VMLAB и PROTEUS VSM при отладке программ для AVR на ассемблере и языке Си.

Для студентов вузов радиоэлектронного профиля и инженеров-проектировщиков средств и систем автоматики и промышленной электроники.

ЧАСТЬ 1. Микроконтроллеры МК51

ПРЕДИСЛОВИЕ

Термин «контроллер» образовался от английского слова to control — управлять. Наиболее распространенными на сегодняшний день схемами управления являются схемы, построенные на основе цифровых микросхем.

Основные требования, которые предъявляются к таким устройствам, можно сформулировать следующим образом:

- низкая стоимость;
- высокая надежность;
- высокая степень миниатюризации;
- малое энергопотребление;
- работоспособность в жестких условиях эксплуатации;
- достаточная производительность для выполнения всех требуемых функций.

В отличие от универсальных компьютеров к управляющим контроллерам, как правило, не предъявляются высокие требования к производительности и программной совместимости.

Выполнение всех этих довольно противоречивых условий одновременно затруднительно, поэтому совершенствование и развитие техники пошло по пути специализации и в настоящее время количество различных моделей управляющих контроллеров чрезвычайно велико.

Однако можно выделить некоторые черты архитектуры и системы команд, общие для всех современных микроконтроллеров, это:

- так называемая Гарвардская архитектура — то есть раздельные области памяти для хранения команд (программы) и данных. Они могут иметь различную разрядность, в системе команд для этого предусмотрены различные команды и т.д.;
- интеграция в одном корпусе микросхемы (на одном кристалле) практически всех блоков, характерных для полнофункционального компьютера — процессора, ПЗУ, ОЗУ, устройств ввода/вывода, тактового генератора, контроллера прерываний.

Поэтому в русскоязычной литературе подобные устройства часто называют однокристальными ЭВМ.

Микроконтроллеры (МК) обычно классифицируют по разрядности обрабатываемых чисел:

- четырехразрядные — самые простые и дешевые;
- восьмиразрядные — наиболее многочисленная группа (оптимальное сочетание цены и возможностей), к этой группе относятся микроконтроллеры семейства MCS-51 (Intel) и совместимые с ними AT89 (Atmel) и отечественные МК51 (серии К1816ВЕ51 и К1830ВЕ51), микроконтроллеры AVR (Atmel), PIC (MicroChip), HC08 (Motorola);
- шестнадцатиразрядные — MCS-96 (Intel) и др. — более производительные, но более дорогостоящие;
- тридцатидвухразрядные — обычно являющиеся модификациями универсальных микропроцессоров (i80186 или i386EX).

С точки зрения классификации различают микроконтроллеры двух типов:

- МК с CISC-архитектурой — с полной системой команд (Complicated Instruction Set Computer);
- МК с RISC-архитектурой — с сокращенной системой команд (Reduced Instruction Set Computer), отличающиеся меньшим временем выполнения команд.

8-разрядные CISC-микроконтроллеры архитектуры MCS-51, разработанной фирмой Intel, уже много лет лидируют на мировом рынке, как по количеству разновидностей, так и по количеству выпускающих их фирм. Основными производителями их являются Philips, Atmel, Siemens, Intel, Dallas и ряд других. Пожалуй, не менее популярными являются RISC-микроконтроллеры семейства AVR фирмы Atmel.

Микроконтроллерам семейства MCS-51 уделено основное внимание при изучении первой части данной дисциплины. Изложение лекционного курса сочетается с выполнением двух компьютерных лабораторных работ. Микроконтроллеры семейства AVR изучаются во второй части дисциплины.

1 ПРИНЦИПЫ ПОСТРОЕНИЯ ЦИФРОВЫХ УСТРОЙСТВ УПРАВЛЕНИЯ

При проектировании сложных электронных устройств используется принцип декомпозиции задачи. Он сводится к последовательной разработке структурной, функциональной и принципиальной схемы устройства. Цифровое устройство реализуется аппаратными средствами в виде совокупности интегральных микросхем комбинационного и последовательностного типов.

Схема электрическая структурная (код схемы Э1) определяет основные функциональные части устройства, их назначение и взаимосвязь. Используется для общего ознакомления с изделием.

Схема электрическая функциональная (код схемы Э2) разъясняет процессы, протекающие в отдельных функциональных частях изделия или в изделии в целом. Используется для изучения принципов работы устройства, а также при наладке, контроле, ремонте.

Схема электрическая принципиальная (код схемы Э3) определяет полный состав элементов и связей между ними и дает детальное представление о принципах работы изделия. Она служит исходным документом при разработке других конструкторских документов (печатных плат, сборочных чертежей, схем соединений и т.п.).

Разрешается разрабатывать совмещенные схемы, когда на схемах одного типа изображают фрагменты схем других типов.

Большой практический интерес представляют цифровые устройства, реализующие некоторый алгоритм обработки информации, т.е. выполняющие упорядоченную последовательность определенных операций над поступающими данными. При построении таких устройств целесообразно использовать принцип микропрограммного управления, состоящий в следующем:

- любая операция, реализуемая устройством, рассматривается как сложное действие, которое разделяется на последовательность элементарных действий, называемых микрооперациями;
- для управления порядком следования операций используются оповестительные сигналы — логические условия, прини-

мающие значения 1 или 0 в зависимости от результата выполнения микроопераций;

- процесс выполнения операций в устройстве описывается в виде алгоритма, представленного в терминах микроопераций и логических условий и называемого микропрограммой;
- микропрограмма дает путь к определению структуры устройства, его реализации на выбираемой элементной базе.

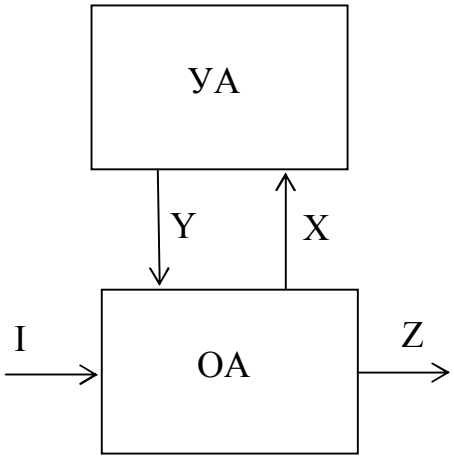


Рис. 1.1 — Структурная схема цифрового автомата

При использовании описанного принципа принято делить цифровое устройство (рис. 1.1) на операционный автомат (ОА) и управляющий автомат (УА).

Операционный автомат предназначен для хранения поступающей информации I, выполнения заданного набора микроопераций, выработки логических условий X и выходных сигналов Z.

Управляющий автомат генерирует последовательность управляющих сигналов Y в соответствии с заданной программой и значениями логических условий X.

Цифровое устройство реализуется аппаратно-программными средствами с использованием микропроцессорных комплектов интегральных схем.

Микропроцессор (МП) — выполненное в виде большой интегральной схемы (БИС) цифровое устройство, предназначенное для обработки информации в соответствии с хранимой в памяти программой. Он реализует принцип микропрограммного управления и содержит на кристалле основные элементы операционного и управляющего автомата. Уровень микрокоманд часто скрыт от пользователя, который разрабатывает программу работы микропроцессора на уровне команд. Но полезно помнить, что каждая команда выполняется за определенное число тактов (микрокоманд). Вместе с памятью и устройствами ввода/вывода информации МП образует микропроцессорную систему.

Микропроцессорные системы можно разделить на микроЭВМ и микроконтроллеры. Микроконтроллеры — специализированные устройства с программой, зашитой в ПЗУ, выполняющие задачи управления в реальном масштабе времени. МикроЭВМ — более универсальные устройства с развитыми средствами диалогового общения с человеком (клавиатура, дисплей и т.п.), легко перестраиваемые на решение новых задач. В изучаемой дисциплине основное внимание уделяется встроенным микропроцессорным системам управления на базе микроконтроллеров. Применение однокристальных микроконтроллеров в устройствах бытовой и медицинской электроники, в устройствах управления технологическим оборудованием, преобразователями электрической энергии, в измерительных приборах обеспечивает достижение исключительно высоких показателей эффективности при низкой стоимости.

Любой микроконтроллер содержит центральный процессор, память и интерфейс ввода/вывода (рис. 1.2). ПЗУ хранит основную программу, подпрограммы, таблицы, константы. ОЗУ используется для хранения результатов промежуточных вычислений, массивов данных, поступающих от датчиков, либо подготовленных к выдаче внешним устройствам. Генератор тактовых импульсов (ГТИ) синхронизирует работу всей микропроцессорной системы. Интерфейс (ИФ) используется для сопряжения с внешними устройствами (ВУ) по временным и электрическим параметрам и представляет собой набор шин (портов), специальных сигналов и алгоритмов обмена информацией.

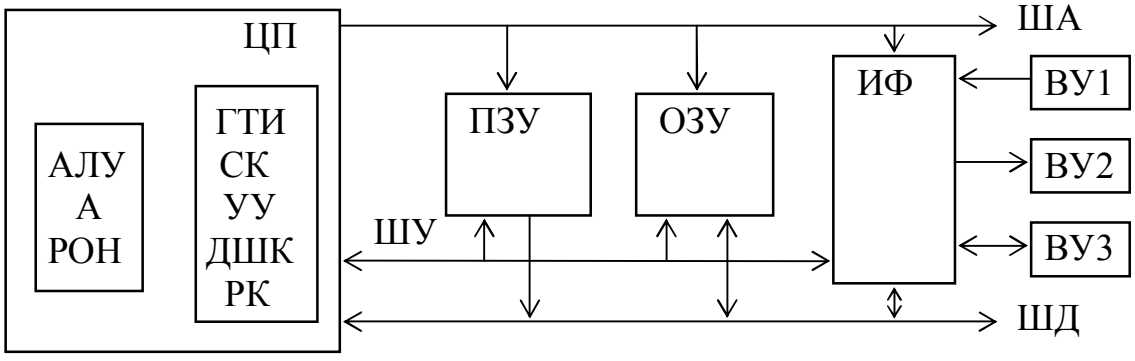


Рис. 1.2 — Структурная схема микропроцессорной системы

Основу центрального процессора (ЦП) составляет арифметико-логическое устройство (АЛУ), позволяющее выполнять арифметические, логические операции и операции сдвига над данными, представленными в двоичном коде. В состав операционной части входят также регистры общего назначения (РОН) и основной рабочий регистр — аккумулятор (А).

Управляющая часть содержит регистр команд (РК), дешифратор команд (ДШК), устройство управления (УУ), счетчик команд (СК) и ГТИ.

Последовательность выполнения команд:

1) содержимое счетчика команд выставляется на шину адреса (ША);

2) выбирается код команды из ПЗУ через шину данных (ШД) в РК;

3) происходит дешифрация кода в ДШК и УУ вырабатывает последовательность сигналов, необходимых для выполнения команды;

4) осуществляется подготовка и размещение операндов;

5) выполняется операция с участием АЛУ, А и РОН.

Во время выполнения команды СК формирует адрес следующей команды. Команды выбираются из ПЗУ последовательно. При выполнении команд условных и безусловных переходов содержимое СК меняется скачком, что позволяет реализовать ветвящиеся или циклические программы. Кроме шины адреса и шины данных системная магистраль включает шину управления (ШУ), в которую входят линии и сигналы, вырабатываемые центральным процессором для управления памятью и устройствами ввода/вывода, и запросы внешних устройств на обмен информацией с ЦП или ОЗУ.

Различают три способа обмена информацией между микропроцессорной системой и внешними устройствами:

1) программный — инициируется программой с помощью команд ввода и вывода. Важной задачей является проверка готовности ВУ. В некоторых системах при отсутствии готовности внешних устройств МП переходит в состояние ожидания. Чаще приходится организовывать специальную команду опроса готовности, которая повторяется многократно до появления сигнала готовности. Достоинство программного обмена — его простота,

недостаток — бесполезная трата времени на ожидание и невозможность обеспечения своевременной реакции МП на внезапно возникшую потребность ВУ в обмене информацией;

2) с прерыванием программы — по специальному сигналу запроса на прерывание МП после выполнения текущей команды переходит к выполнению подпрограммы обслуживания прерываний, затем возвращается к реализации основной программы;

3) с помощью прямого доступа к памяти (ПДП) — на запрос ПДП микропроцессор прекращает работу и отключается от системной магистрали. Обмен информацией между ОЗУ и ВУ осуществляет специальный контроллер прямого доступа к памяти. Используется для ускорения обмена блоками информации.

Наиболее распространёнными в настоящее время являются микроконтроллеры семейства MCS-51. Архитектура семейства MCS-51 в значительной мере предопределяется ее назначением — построение компактных и дешевых цифровых устройств. Все функции микроЭВМ реализуются с помощью единственной микросхемы. В состав семейства MCS-51 входит целый ряд микросхем от самых простых микроконтроллеров до достаточно сложных. Все микросхемы этого семейства работают с одной и той же системой команд, большинство из них выполняется в одинаковых корпусах с совпадающей цоколевкой (нумерация ножек для корпуса). Это позволяет использовать для разработанного устройства микросхемы разных фирм-производителей (таких как Intel, Dallas, Atmel, Philips и т.д.) без переделки принципиальной схемы устройства и программы.

Контроллер MCS-51 состоит из следующих основных функциональных узлов: блока управления, арифметико-логического устройства, блока таймеров/счетчиков, блока последовательного интерфейса и прерываний, программного счетчика, памяти данных и памяти программ. Двусторонний обмен осуществляется с помощью внутренней 8-разрядной магистрали данных. Различные микросхемы этого семейства различаются только регистрами специального назначения (в том числе и количеством портов). В качестве базовых при изучении данного семейства выбраны отечественные микросхемы серий К1816 и К1830, обозначаемые далее для краткости МК51.

2 ОБЩАЯ ХАРАКТЕРИСТИКА МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА МК51

Семейство МК51 включает ряд микросхем, основное различие между которыми состоит в реализации памяти программ и мощности потребления (табл. 2.1). Микросхемы серии К1816 выполнены по n-МОП технологии, микросхемы серии К1830 — по технологии КМОП.

Таблица 2.1

Микро-схемы	Аналог Intel	Объем РПП	Тип РПП	Объем РПД	fmax, МГц	Iп, мА
К1816ВЕ51	8051АН	4Кб	ПЗУ	128 байт	12	150
К1816ВЕ31	8031АН	нет	–	128 байт	12	150
К1830ВЕ51	80С51ВН	4Кб	ПЗУ	128 байт	12	18
К1830ВЕ31	80С31ВН	нет	–	128 байт	12	18

Кроме указанных в таблице резидентной памяти программ и данных (РПП и РПД) микроконтроллер содержит на кристалле:

- 8-разрядный центральный процессор (в АЛУ реализуются сложение, вычитание, умножение и деление);
- четыре программируемых 8-разрядных порта ввода/вывода (P0-P3);
- два 16-битовых многорежимных таймера/счетчика;
- систему прерываний с пятью векторами и двумя уровнями;
- последовательный интерфейс;
- тактовый генератор;
- битовый процессор.

За счет подключения внешних БИС память программ может быть расширена до 64 Кб, память данных — на 64 Кб.

Многие выводы микросхем допускают многофункциональное использование (на рис. 2.1 показаны альтернативные функции порта P3).

Выводы BQ1 и BQ2 служат для подключения кварцевого резонатора, вывод RST — для организации сброса МК. Вывод VPP (выбор памяти программ) для блокировки резидентной памяти программ заземляется. При подключении внешней памяти

через порт P0 передаются данные и младший байт адреса (A0-A7), через порт P2 — старший байт адреса (A8-A15). Для реализации альтернативных функций порта P3 в соответствующие линии порта необходимо вывести единицы.

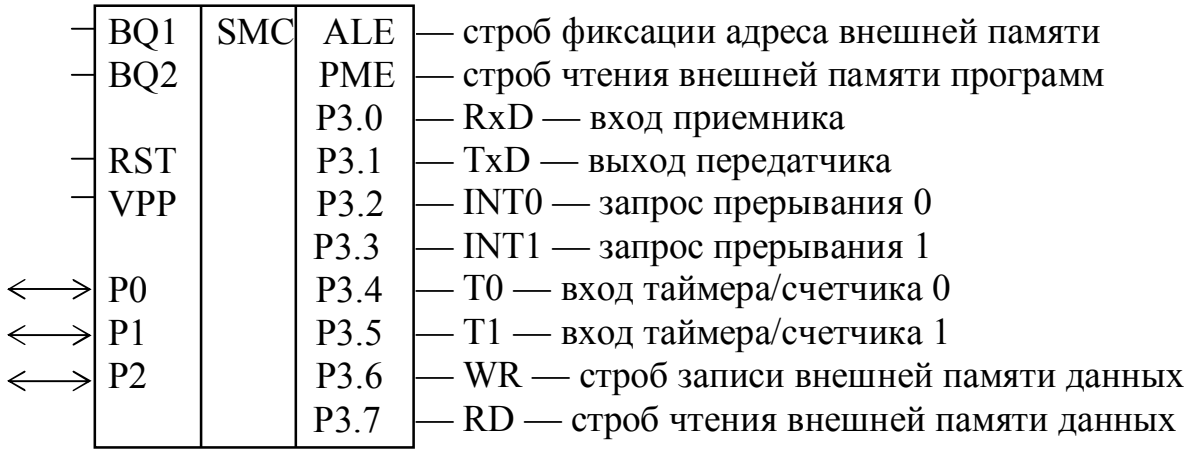


Рис. 2.1 — Обозначение и функциональное назначение выводов

Микроконтроллер выпускается в корпусе (PDIP, SOIC, QFP), имеющем 40 внешних выводов. Для работы МК51 требуется один источник электропитания напряжением +5В. Выводы для подключения источника питания на функциональной схеме рис. 2.1 не показаны. Через четыре программируемых порта ввода/вывода МК51 взаимодействует с внешней средой в стандарте ТТЛ-схем с тремя состояниями выхода. Ниже приводится описание основных функциональных узлов микроконтроллера.

Блок управления и синхронизации предназначен для выработки синхронизирующих и управляющих сигналов, обеспечивающих координацию совместной работы блоков МК51 во всех допустимых режимах работы. В состав блока управления входят: устройство формирования временных интервалов, логика ввода-вывода, регистр команд, регистр управления потреблением электроэнергии, дешифратор команд, логика управления МК.

Устройство формирования временных интервалов предназначено для формирования и выдачи внутренних синхросигналов фаз, тактов и циклов. Количество машинных циклов определяет продолжительность выполнения команд. Практически все команды ОЭВМ выполняются за один или два машинных цикла,

кроме команд умножения и деления, продолжительность выполнения которых составляет четыре машинных цикла. Обозначим частоту задающего генератора через $F_г$. Тогда длительность машинного цикла равна $12/F_г$ или составляет 12 периодов сигнала задающего генератора. Логика ввода-вывода предназначена для приема и выдачи сигналов, обеспечивающих обмен информации с внешними устройствами через порты ввода-вывода P0-P3.

Регистр команд предназначен для записи и хранения 8-ми разрядного кода операции выполняемой команды. Код операции с помощью дешифратора команд и логики управления МК преобразуется в микропрограмму выполнения команды.

Регистр управления потреблением (PCON) позволяет останавливать работу микроконтроллера для уменьшения потребления электроэнергии и уменьшения уровня помех от микроконтроллера. Еще большего уменьшения потребления электроэнергии и уменьшения помех можно добиться, остановив задающий генератор микроконтроллера. Этого можно достичь при помощи переключения бит регистра управления потреблением PCON. Для варианта изготовления по технологии n-МОП (серия 1816) регистр управления потреблением PCON содержит только один бит, управляющий скоростью передачи последовательного порта SMOD, а биты управления потреблением электроэнергией отсутствуют.

Арифметико-логическое устройство (ALU) представляет собой параллельное восьмиразрядное устройство, обеспечивающее выполнение арифметических и логических операций. АЛУ состоит из: регистров временного хранения TMP1 и TMP2, ПЗУ констант, сумматора, дополнительного регистра (регистра B), аккумулятора (ACC), регистра состояния программ (PSW).

Регистры временного хранения — восьмиразрядные регистры, предназначенные для приема и хранения операндов на время выполнения операций над ними. Эти регистры программно не доступны.

ПЗУ констант обеспечивает выработку корректирующего кода при двоично-десятичном представлении данных, кода маски при битовых операциях и кода констант.

Параллельный восьмиразрядный сумматор представляет собой схему комбинационного типа с последовательным переносом, предназначенную для выполнения арифметических операций сложения, вычитания и логических операций сложения, умножения, неравнозначности и тождественности.

Регистр B — восьмиразрядный регистр, используемый во время операций умножения и деления. Для других инструкций он может рассматриваться как дополнительный сверхоперативный регистр.

Аккумулятор — восьмиразрядный регистр, предназначенный для приема и хранения результата, полученного при выполнении арифметико-логических операций или операций сдвига.

Блок последовательного интерфейса и прерываний (ПИП) предназначен для организации ввода-вывода последовательных потоков информации и организации системы прерывания программ. В состав блока входят: буфер ПИП, логика управления, регистр управления, буфер передатчика, буфер приемника, приемопередатчик последовательного порта, регистр приоритетов прерываний, регистр разрешения прерываний, логика обработки флагов прерываний и схема выработки вектора.

Счетчик команд (Program Counter) предназначен для формирования текущего 16-разрядного адреса внутренней памяти программ и 8/16-разрядного адреса внешней памяти программ. В состав счетчика команд входят 16-разрядные буфер PC, регистр PC и схема инкремента (увеличения содержимого на 1).

Память данных предназначена для временного хранения информации, используемой в процессе выполнения программы.

Порты P0, P1, P2, P3 являются квазидвунаправленными портами ввода-вывода и предназначены для обеспечения обмена информацией МК с внешними устройствами, образуя 32 линии ввода-вывода.

Регистр состояния программы (PSW) предназначен для хранения информации о состоянии АЛУ при выполнении программы.

Память программ предназначена для хранения программ и представляет собой постоянное запоминающее устройство (ПЗУ). В разных микросхемах применяются масочные, стираемые ультрафиолетовым излучением или **FLASH** ПЗУ.

Регистр указателя данных (DPTR) предназначен для хранения 16-разрядного адреса внешней памяти данных.

Указатель стека (SP) представляет собой восьмиразрядный регистр, предназначенный для организации особой области памяти данных (стека), в которой можно временно сохранить содержимое любой ячейки памяти. Прежде всего стек предназначен для сохранения адреса возврата при вызове подпрограмм.

В АЛУ реализуется механизм каскадного выполнения микроопераций при выполнении сложных команд. Так, например, при выполнении одной из команд условного перехода по результату сравнения в АЛУ трижды инкрементируется счетчик команд, дважды производится чтение из памяти данных, выполняется арифметическое сравнение двух переменных, формируется 16-битный адрес перехода и принимается решение о том, делать или не делать переход по программе.

Расширенная система команд обеспечивает побайтовую и побитовую адресацию. Отдельные программно доступные биты могут быть установлены, сброшены, могут пересылаться, проверяться и использоваться в логических вычислениях.

Выходные линии портов 1, 2 и 3 могут работать на одну ТТЛ-схему. Линии порта 0 могут быть нагружены на два входа ТТЛ каждая.

Знакомство с программной моделью, системой команд, системой прерываний, таймерами/счетчиками событий и простейшими средствами отладки МК51 предлагается реализовать в ходе выполнения двух лабораторных работ.

3 ПРОГРАММНАЯ МОДЕЛЬ И СИСТЕМА КОМАНД МК51 (ЛАБОРАТОРНАЯ РАБОТА №1)

Цель работы. Целью лабораторной работы является приобретение навыков использования ассемблера, редактора связей и эмулятора при формировании и отладке прикладных программ для однокристальных микроконтроллеров семейства MCS-51.

Программная модель микроконтроллера МК51

С точки зрения программиста микроконтроллер представляет набор регистров и ячеек памяти с конкретными адресами и обозначениями (рис. 3.1). Резидентную память данных (**РПД**) составляют четыре банка регистров общего назначения, выбор которых осуществляется установкой и сбросом битов RS1 и RS0 в PSW, 128 программно управляемых флагов пользователя (регистры с адресами 20H-2FH) и регистры с адресами 30H-7FH, которые можно использовать как ОЗУ пользователя или стек. **Стек** — специально организованная область ОЗУ, предназначенная для временного хранения данных или адресов. Число, записанное в стек последним, извлекается из него первым. Указатель стека **SP** хранит адрес последней ячейки стека, в которой записана информация. При вызове подпрограммы в стеке автоматически сохраняется адрес возврата в основную программу. Как правило, в начале каждой подпрограммы сохраняют в стеке содержимое всех задействованных при ее выполнении регистров, а в конце подпрограммы восстанавливают их из стека. К адресному пространству РПД непосредственно примыкают адреса регистров специальных функций **РСФ** (знаком * отмечены регистры, допускающие адресацию отдельных бит, их адреса делятся на 8).

К полному адресному пространству ячеек внешнего ОЗУ данных обращение производится косвенно через 16-битовый регистр-указатель данных **DPTR**. Для чтения данных из таблиц, зашитых в памяти программ, используется косвенно-регистровая адресация. Любой байт из таблицы может быть выбран по адресу, определяемому суммой содержимого DPTR или **PC** (программный счетчик) и содержимого аккумулятора.

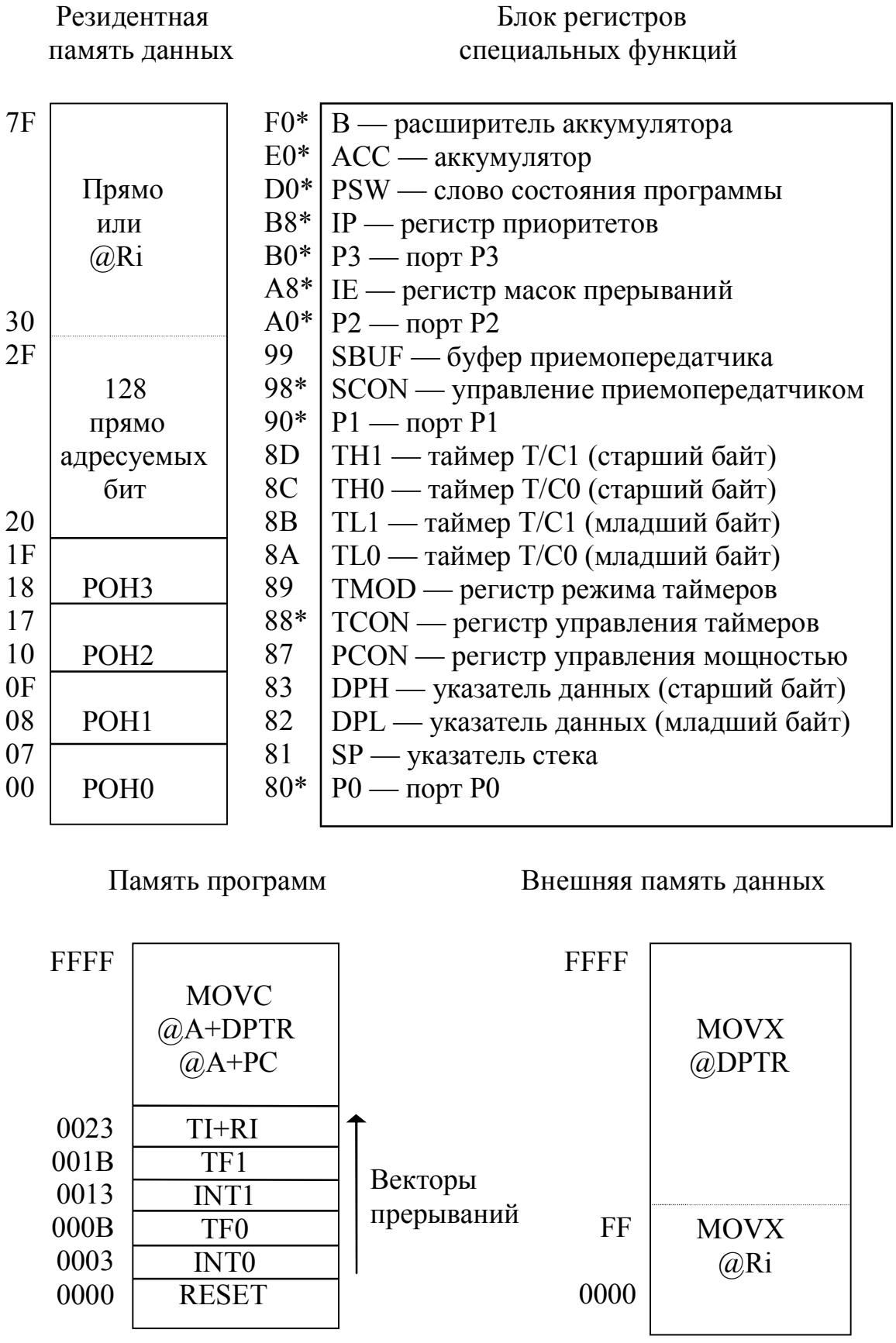


Рис. 3.1 — Программная модель MCS-51

Результатом выполнения некоторых команд является не только изменение содержимого аккумулятора или РОН, но и формирование признаков (флагов) в специальном регистре слова состояния программы **PSW** (рис. 3.2).

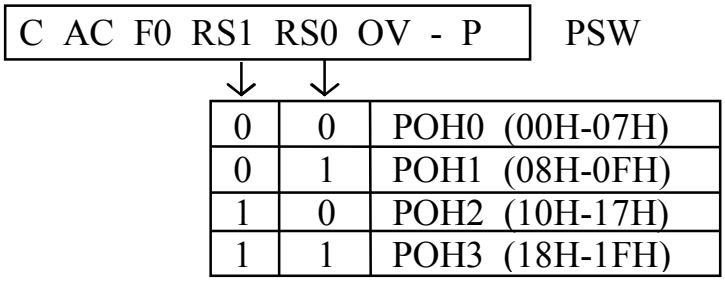


Рис. 3.2. — Слово состояния программы МК51

Слово состояния программы PSW включает в себя четыре флага: **C** — перенос, **AC** — вспомогательный перенос, **OV** — переполнение и **P** — паритет. Флаг OV устанавливается, если результат операции сложения/вычитания не укладывается в семи битах и старший бит результата не может интерпретироваться как знаковый. При выполнении операции деления флаг OV сбрасывается, а в случае деления на нуль — устанавливается. При умножении флаг OV устанавливается, если результат больше 255. Флаг P равен 0, если число единиц аккумулятора четное.

В памяти программ есть пять адресов, которым соответствуют векторы прерываний от пяти источников. Стрелка на рисунке показывает порядок убывания их приоритетов.

Доступ к внешней памяти данных возможен с использованием 16-битного адреса (MOVX A,@DPTR) или 8-битного адреса (MOVX A,@Ri). В любых случаях использования 16-разрядного адреса старший байт адреса фиксируется (и сохраняется неизменным в течение одного цикла записи или чтения) в регистре-защелке порта P2.

Расширенная система команд обеспечивает побайтовую и побитовую адресацию. Отдельные программно доступные биты могут быть установлены, сброшены, могут пересылаться, проверяться и использоваться в логических вычислениях.

Система команд микроконтроллера

Микроконтроллер имеет 255 команд: пересылки данных, арифметических операций, логических операций, передачи управления, операций с битами. По формату команды могут быть одно-, двух- и трехбайтовыми. Из 111 базовых команд 64 выполняются за один машинный цикл (1 мкс при частоте кварца 12 МГц), 45 команд — за два. Команды умножения и деления выполняются за четыре машинных цикла.

Типы команд микроконтроллера МК51:				Пример команды
1)	КОП			SWAP A
2)	КОП	#d		ADDC A,#d
3)	КОП	ad		ANL A,ad
4)	КОП	bit		CLR bit
5)	КОП	rel		SJMP $+2+rel
6)	$A_{10}A_9A_8$КОП	A_7 ... A_0		AJMP ad11
7)	КОП	ad	#d	ORL ad,#d
8)	КОП	ad	rel	DJNZ ad, $+3+rel
9)	КОП	ads	add	MOV add,ads
10)	КОП	#d	rel	CJNE R0,#d, $+3+rel
11)	КОП	bit	rel	JB bit, $+3+rel
12)	КОП	ad16h	ad16l	LCALL ad16
13)	КОП	#d16h	#d16l	MOV DPTR,#d16

Обозначения операндов расшифрованы далее после таблицы команд микроконтроллера. Ниже приведены примеры команд.

Команда ADD A, «байт-источник» [3].

Эта команда (сложение) складывает содержимое аккумулятора A с содержимым байта-источника, оставляя результат в аккумуляторе. При появлении переноса из разрядов 7 и 3 устанавливаются флаги переноса C и дополнительного переноса AC соответственно, в противном случае эти флаги сбрасываются. При сложении целых чисел без знака флаг переноса указывает, что сумма больше 255. Флаг переполнения OV устанавливается, если есть перенос из бита 6 и нет переноса из бита 7, или есть перенос из бита 7 и нет — из бита 6, в противном случае флаг OV сбрасывается. При сложении целых чисел со знаком флаг OV указывает на отрицательную величину, полученную при сложении двух положительных операндов или на положительную сумму для двух отрицательных операндов. Время выполнения команды — один машинный цикл.

Для команды сложения разрешены следующие способы адресации байта-источника:

1) регистровый (в качестве операнда используются регистры R0-R7);

```
                ; (A)=C3H,  (R6)=AAH — до выполнения команды
ADD A,R6        ; (A)=6DH,  (R6)=AAH — после выполнения
                ; AC=0, C=1, OV=1      — команды
```

2) косвенно-регистровый (@ — префикс косвенной адресации, в качестве регистров-указателей памяти могут использоваться R0 и R1);

```
                ; (A)=95H,  (R1)=31H,  ((R1))=4CH
ADD A,@R1       ; (A)=E1H, ((R1))=4CH — после выполнения
                ; AC=1, C=0, OV=0       — команды
```

3) прямой (указывается прямой адрес байта РПД или регистра специальных функций);

```
                ; (A)=77H,  (90H)=FFH — до выполнения команды
ADD A,90H       ; (A)=76H,  (90H)=FFH — после выполнения
                ; AC=1, C=1, OV=0       — команды
```

4) непосредственный (# — префикс непосредственных данных).

```
                    ; (A)=09H   — до выполнения команды
ADD A,#0D3H  ; (A)=DCH — после выполнения команды
                    ; C=0, OV=0, AC=0
```

Команда LCALL «ad16».

Команда «длинный вызов» вызывает подпрограмму, находящуюся по указанному адресу. К счетчику команд PC прибавляется 3 для получения адреса следующей команды и после этого полученный 16-битовый результат помещается в стек (сначала следует младший байт, за ним — старший), а содержимое указателя стека SP увеличивается на 2.

Затем старший и младший байты счетчика команд загружаются соответственно вторым и третьим байтом команды LCALL. Выполнение программы продолжается командой, находящейся по полученному адресу. Подпрограмма может начинаться в любом месте памяти программ объемом до 64 Кбайт. Команда на флаги не влияет.

```
Ассемблер:      LCALL      <метка>
Время:          2 цикла
Алгоритм:       (PC):= (PC)+3
                (SP):= (SP)+1
                ((SP)):= (PC[7-0])
                (SP):= (SP)+1
                ((SP)):= (PC[15-8])
                (PC):= <ad[15-0]>
Пример:                 ; (SP) = 64H
                        ; метке SUBBR соответствует адрес 1234H
                        ; по адресу 0126H находится команда LCALL
        LCALL  SUBBR    ; (SP):= 66H,       (PC)= 1234H
                        ; (ОЗУ[65H])=29H,       (ОЗУ[66H])=01H
```

Команда RET.

Команда «возврат из подпрограммы» последовательно выгружает старший и младший байты счетчика команд из стека, уменьшая указатель стека на 2. Выполнение основной программы продолжается по адресу команды, следующей за ACALL или LCALL. На флаги эта команда не влияет.

Ассемблер: RET
Время: 2 цикла
Алгоритм: (PC[15-8]) := ((SP))
(SP):= (SP)-1
(PC[7-0]) := ((SP))
(SP):= (SP)-1

Пример: ; (SP) = 66H, (ОЗУ[65H])=29H, (ОЗУ[66H])=01H
RET ; (SP) = 64H, (PC)= 0129H

Команда RETI.

Команда «возврат из прерывания» выгружает старший и младший байты счетчика команд из стека, уменьшая указатель стека на 2. Устанавливает логику прерываний, разрешая прием других прерываний с уровнем приоритета, равным уровню приоритета только что обработанного прерывания. Слово состояния программы PSW не восстанавливается автоматически.

Выполнение основной программы продолжается с команды, следующей за командой, на которой произошел переход к обнаружению запроса на прерывание. Если при выполнении команды RETI обнаружено прерывание с таким же или меньшим уровнем приоритета, то одна команда основной программы успевает выполниться до обработки такого прерывания.

Результаты трансляции 13 типов команд микроконтроллера МК51 иллюстрирует приводимая ниже программа (число байт — 1, 2 или 3).

```
#############################################################################
#    Micro Series 8051 Assembler V1.80/MD2                  25/Apr/06  08:47:51  #
#                                                                               #
#    Source  =  test.asm                                                        #
#    List    =  test.lst                                                        #
#   Object          =  test.r03                                                 #
#    Options =                                                                  #
#                                                  (c) Copyright IAR Systems 1985   #
#############################################################################
 1
 2       ; ПРИМЕРЫ 13 ТИПОВ КОМАНД МИКРОКОНТРОЛЛЕРА MCS-51
 3
 4               ; Символические обозначения операндов
 5
 6  0044         DATA    EQU     44H         ; 8-разрядные данные
 7  0077         SMALL   EQU     77H         ; прямой адрес байта
 8  00E0         BIT     EQU     ACC.0       ; прямой адрес бита
 9  FFFB         REL1    EQU     -5          ; смещение назад
10  0005         REL2    EQU     5           ; смещение вперед
11  05FA         MIDL    EQU     5FAH        ; 11-битный адрес
12  0033         PRI     EQU     33H         ; адрес приемника
13  0022         IST     EQU     22H         ; адрес источника
14  ABCD         BIG     EQU     0ABCDH      ; 16-битный операнд
15
16              ; Запись и трансляция команд
17
18  0000 C4             SWAP    A                       ;  1 тип
19  0001 3444           ADDC    A,#DATA                 ;  2 тип
20  0003 5577           ANL     A,SMALL                 ;  3 тип
21  0005 C2E0           CLR     BIT                     ;  4 тип
22  0007 8005           SJMP    $+2+REL2                ;  5 тип
23  0009 A1FA           AJMP    MIDL                    ;  6 тип
24  000B 437744         ORL     SMALL,#DATA             ;  7 тип
25  000E D577FB         DJNZ    SMALL,$+3+REL1          ;  8 тип
26  0011 852233         MOV     PRI,IST                 ;  9 тип
27  0014 B84405         CJNE    R0,#DATA,$+3+REL2       ; 10 тип
28  0017 20E0FB         JB      BIT,$+3+REL1            ; 11 тип
29  001A 12ABCD         LCALL   BIG                     ; 12 тип
30  001D 90ABCD         MOV     DPTR,#BIG               ; 13 тип
31  0020                END

 Errors:  None          ########
 Bytes:   32            #  test  #
 CRC:     3790          ########
```

Команды микроконтроллера семейства МК51

Мнемоника	Описание команды	Байты	Циклы
	Группа команд пересылки данных		
MOV A,Rn	Пересылка в аккумулятор из РОН (n=0-7)	1	1
MOV A,ad	Пересылка в А прямо адресуемого байта	2	1
MOV A,@Ri	Пересылка в аккумулятор байта РПД (i=1,2)	1	1
MOV A,#d	Загрузка в аккумулятор константы	2	1
MOV Rn,A	Пересылка в регистр из аккумулятора	1	1
MOV Rn,ad	Пересылка в РОН прямо адресуемого байта	2	2
MOV Rn,#d	Пересылка в регистр константы	2	1
MOV ad,A	Пересылка по прямому адресу A	2	1
MOV ad,Rn	Пересылка по прямому адресу регистра	2	2
MOV add,ads	Пересылка прямо адресуемого байта по прямому адресу	3	2
MOV ad,@Ri	Пересылка байта РПД по прямому адресу	2	2
MOV ad,#d	Пересылка константы по прямому адресу	3	2
MOV @Ri,A	Пересылка байта в РПД из А	1	1
MOV @Ri,ad	Пересылка прямо адресуемого байта в РПД	2	2
MOV @Ri,#d	Пересылка константы в РПД	2	1
MOV DPTR,#d16	Загрузка указателя данных	3	2
MOVC A,@A+DPTR	Пересылка в А байта из памяти программ	1	2
MOVC A,@A+PC	Пересылка в А байта из памяти программ	1	2
MOVX A,@Ri	Пересылка в А байта из ВПД	1	2
MOVX A,@DPTR	Пересылка в А байта из расширенной ВПД	1	2
MOVX @Ri,A	Пересылка в ВПД из аккумулятора	1	2
MOVX @DPTR,A	Пересылка в расширенную ВПД из А	1	2
PUSH ad	Загрузка в стек	2	2
POP ad	Извлечение из стека	2	2
XCH A, Rn	Обмен аккумулятора с регистром	1	1
XCH A,ad	Обмен А с прямо адресуемым байтом	2	1
XCH A,@Ri	Обмен A с байтом из РПД	1	1
XCHD A,@Ri	Обмен младшими тетрадами А и байта РПД	1	1
	Группа команд арифметических операций		
ADD A,Rn	Сложение А с регистром (n=0-7)	1	1
ADD A,ad	Сложение А с прямо адресуемым байтом	2	1
ADD A,@Ri	Сложение А с байтом из РПД (i=0,1)	1	1
ADD A,#d	Сложение А с константой	2	1
ADDC A,Rn	Сложение А с регистром и переносом	1	1
ADDC A,ad	Сложение А с байтом и переносом	2	1
ADDC A,@Ri	Сложение А с байтом РПД и переносом	1	1
ADDC A,#d	Сложение А с константой и переносом	2	1
DA A	Десятичная коррекция аккумулятора	1	1
SUBB A,Rn	Вычитание из А регистра и заема	1	1
SUBB A,ad	Вычитание из А байта и заема	2	1

Мнемоника	Описание команды	Байты	Циклы
SUBB A,@Ri	Вычитание из А байта РПД и заема	1	1
SUBB A,#d	Вычитание из А константы и заема	2	1
INC A	Инкремент аккумулятора	1	1
INC Rn	Инкремент регистра	1	1
INC ad	Инкремент прямо адресуемого байта	2	1
INC @Ri	Инкремент байта РПД	1	1
INC DPTR	Инкремент указателя данных	1	2
DEC A	Декремент аккумулятора	1	1
DEC Rn	Декремент регистра	1	1
DEC ad	Декремент прямо адресуемого байта	2	1
DEC @Ri	Декремент байта РПД	1	1
MUL AB	Умножение аккумулятора на регистр В	1	4
DIV AB	Деление аккумулятора на регистр В	1	4
	Группа команд логических операций		
ANL A,Rn	Логическое И регистра и А	1	1
ANL A,ad	Логическое И А и прямо адресуемого байта	2	1
ANL A,@Ri	Логическое И байта РПД и А	1	1
ANL A,#d	Логическое И константы и А	2	1
ANL ad,A	Логическое И прямо адресуемого байта и А	2	1
ANL ad,#d	Логическое И байта и константы	3	2
ORL A,Rn	Логическое ИЛИ регистра и А	1	1
ORL A,ad	Логическое ИЛИ А и байта	2	1
ORL A,@Ri	Логическое ИЛИ байта РПД и А	1	1
ORL A,#d	Логическое ИЛИ константы и А	2	1
ORL ad,A	Логическое ИЛИ байта и А	2	1
ORL ad,#d	Логическое ИЛИ байта и константы	3	2
XRL A,Rn	Исключающее ИЛИ регистра и А	1	1
XRL A,ad	Исключающее ИЛИ А и байта	2	1
XRL A,@Ri	Исключающее ИЛИ байта РПД и А	1	1
XRL A,#d	Исключающее ИЛИ константы и А	2	1
XRL ad,A	Исключающее ИЛИ байта и А	2	1
XRL ad,#d	Исключающее ИЛИ байта и константы	3	2
CLR A	Сброс аккумулятора	1	1
CPL A	Инверсия аккумулятора	1	1
RL A	Циклический сдвиг аккумулятора влево	1	1
RLC A	Сдвиг аккумулятора влево через перенос	1	1
RR A	Циклический сдвиг аккумулятора вправо	1	1
RRC A	Сдвиг аккумулятора вправо через перенос	1	1
SWAP A	Обмен местами тетрад в аккумуляторе	1	1
	Группа команд операций с битами		
CLR C	Сброс переноса	1	1
CLR bit	Сброс бита	2	1
SETB C	Установка переноса	1	1
SETB bit	Установка бита	2	1
CPL C	Инверсия переноса	1	1

Мнемоника	Описание команды	Байты	Циклы
CPL bit	Инверсия бита	2	1
ANL C,bit	Логическое И бита и переноса	2	2
ANL C,/bit	Логическое И инверсии бита и переноса	2	2
ORL C,bit	Логическое ИЛИ бита и переноса	2	2
ORL C,/bit	Логическое ИЛИ инверсии бита и переноса	2	2
MOV C,bit	Пересылка бита в перенос	2	1
MOV bit,C	Пересылка переноса в бит	2	2
	Группа команд передачи управления		
LJMP ad16	Длинный переход в полном объеме ПП	3	2
AJMP ad11	Абсолютный переход внутри стр. в 2 Кбайта	2	2
SJMP $+2+rel	Короткий относительный переход	2	2
JMP @A+DPTR	Косвенный относительный переход	1	2
JZ $+2+rel	Переход, если А равен нулю	2	2
JNZ $+2+rel	Переход, если А не равен нулю	2	2
JC $+2+rel	Переход, если перенос равен единице	2	2
JNC $+2+rel	Переход , если перенос равен нулю	2	2
JB $+3+rel	Переход, если бит равен единице	3	2
JNB $+3+rel	Переход, если бит равен нулю	3	2
JBC $+3+rel	Переход, если бит установлен, с последующим сбросом бита	3	2
DJNZ Rn,$+2+rel	Декремент РОН и переход, если не нуль	2	2
DJNZ ad,$+3+rel	Декремент прямо адресуемого байта и переход, если не нуль	3	2
CJNE A,ad,$+3+rel	Сравнение аккумулятора с прямо адресуемым байтом и переход, если не равно	3	2
CJNE A,#d,$+3+rel	Сравнение аккумулятора с константой и переход, если не равно	3	2
CJNE Rn,#d,$+3+rel	Сравнение регистра с константой и переход, если не равно	3	2
CJNE @Ri,#d,$+3+rel	Сравнение байта в РПД с константой и переход, если не равно	3	2
LCALL ad16	Длинный вызов подпрограммы	3	2
ACALL ad11	Абсолютный вызов подпрограммы в пределах страницы в 2 Кбайта	2	2
RET	Возврат из подпрограммы	1	2
RETI	Возврат из подпрограммы обработки прерывания	1	2
NOP	Холостая команда	1	1

Обозначение операндов: ad — прямой 8-битовый адрес байта РПД, порта или РСФ; add — прямой 8-битовый адрес приемника данных; ads — адрес источника данных; ad11 — прямой 11-битовый адрес передачи управления; ad16 — прямой 16-битовый адрес передачи управления; bit — прямой 8-битовый адрес бита; #d — данные 8-разрядные; #d16 — данные 16-разрядные; rel — 8-битовый байт смещения со знаком, который отсчитывается от адреса следующей команды как число со знаком и записывается в дополнительном коде.

Команды, влияющие на флаги результата

Мнемоника	Флаги
Команда ADD A, <байт источника>	AC C OV
Команда ADDC A, <байт источника>	AC C OV
Команда ANL C, <бит источника>	C
Команда ANL C, </бит источника>	C
Команда CJNE <байт назначения>, <байт источника>, <смещение>	C
Команда CLR C	C
Команда CLR <bit>	bit
Команда CPL C	C
Команда CPL <bit>	bit
Команда DA A	AC C
Команда DIV AB	C=0 OV
Команда MOV <бит назначения>, <бит источника>	C bit
Команда MUL AB	C=0 OV
Команда ORL C, <бит источника>	C
Команда ORL C, </бит источника>	C
Команда RLC A	C
Команда RRC A	C
Команда SETB C	C
Команда SETB <bit>	bit
Команда SUBB A, <байт источника>	AC C OV

Сигнал сброса обнуляет содержимое регистров PC, ACC, B, PSW, DPTR, TMOD, TCON, T/C0, T/C1, IE, IP и SCON, в регистре PCON сбрасывается только старший бит, в регистр SP загружается код 07H, а в порты P0-P3 загружаются коды 0FFH (настраивая их на ввод). Сигнал сброса не воздействует на содержимое ячеек РПД.

Запись программы на языке ассемблера и ее трансляция

Язык ассемблера допускает представление всех элементов программы в символической (буквенно-цифровой) форме, отражающей их содержательный смысл. В качестве алфавита допустимых символов принят код ASCII (американский стандартный код для обмена информацией). Каждая строка ассемблера соответствует одной команде или псевдокоманде (директиве) и может содержать поля метки, мнемоники команды, операнда и комментария. При наличии в программе синтаксических ошибок ассемблер в процессе трансляции выдает сообщения об ошибках.

Метка ассоциируется с 16-битовым адресом той ячейки памяти, где будет размещен первый байт отмеченной команды. Использование меток освобождает программиста от необходимости оперировать абсолютными адресами памяти при записи команд передачи управления. Метка должна начинаться с буквы и заканчиваться двоеточием. Не допускается использовать в качестве меток мнемокоды команд, обозначения операндов и директив ассемблера. Символическое обозначение может появиться в поле метки только один раз.

Мнемокод команды может включать до четырех символов и вместе с обозначениями операндов образует группу ключевых слов ассемблера.

Поле операнда содержит числовые и символьные непосредственные данные, обозначения регистров и регистровых пар МК, адреса памяти. Возможно использование выражений, содержащих простейшие арифметические и логические операции, обработку которых ассемблер при трансляции производит в формате 16-разрядных двоичных чисел. Операнд в виде строки символов, заключенной в апострофы, транслируется в последовательность кодов ASCII этих символов.

Поле комментария начинается с точки с запятой и полностью игнорируется ассемблером. В поле комментария фиксируется обычно функция, которую выполняет группа команд в конкретной прикладной программе.

Кроме команд программа может содержать директивы ассемблера:

ORG — начальный адрес массива;
END — прекращение трансляции;
EQU — эквивалентность, присвоение;
DB — определить байт;
DW — определить слово (два байта).

```
        ORG   800H
        END
MASK    EQU   0FH
        DB    89,7FH,'A'
    DW  1234H, 965
```

Запись исходного текста программы осуществляется с помощью любого текстового редактора (например, вход в редактор Shift+F4 из FAR). Рекомендуется поле метки располагать с нулевой позиции, остальные поля отделять друг от друга клавишей табуляции. Имя программы на языке ассемблера должно иметь расширение.**asm**. Программа должна заканчиваться директивой END.

Для трансляции исходной программы с языка мнемокодов в машинные коды команд в данной лабораторной работе используются следующие кросс-средства:

- Ассемблер **a8051**;
- Редактор связей **xlink**.

Ассемблирование производится в диалоговом режиме. Для трансляции программы TABLO.ASM можно вызвать ассемблер командой **a8051**, следующим образом отвечая на его запросы:

```
sourse file [.msa/s03]=TABLO.ASM
list file [.lst]=TABLO
object file=<Enter>
options=<Enter>
```

После ассемблирования в текущей директории будут сформированы файлы TABLO.LST (файл листинга программы) и TABLO.R03 (файл программы в объектном коде).

Ниже приводится пример записи на языке ассемблера текста программы TABLO.ASM:

Метка	Мнемокод	Операнды	Комментарий

```
                ; Основная программа

MAIN:   MOV     R0,#20H         ; начальный адрес массива в РПД
        MOV     DPTR,#20H       ; адрес таблицы символов в ПП
        MOV     R7,#32          ; число элементов массива
        LCALL   TEXT            ; вызов подпрограммы  TEXT
        MOV     R0,#20H         ; начальный адрес массива в РПД
        MOV     DPTR,#40H       ; адрес таблицы символов в ПП
        MOV     R7,#32          ; число элементов массива
        LCALL   TEXT            ; очистка экрана
        SJMP    MAIN            ; зацикливание программы


                ; Подпрограмма заполнения массива ОЗУ

TEXT:   CLR     A               ; очистка аккумулятора
        MOVC    A,@A+DPTR       ; пересылка символа в А
        MOV     @R0,A           ; пересылка символа в РПД
        DJNZ    R6,$            ; временная задержка
        INC     R0              ; увеличение указателей памяти
        INC     DPTR            ; на единицу
        DJNZ    R7,TEXT         ; организация цикла обработки
        RET                     ; массива

                ; Таблица кодов отображаемых символов

        ORG     20H
        DB       '▫▫DEAR▫▫FRIEND!▫WELCOME▫TO▫TOMSK'
       ORG      40H
        DB      '▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫'          ; 32 пробела
        END
```

При наличии синтаксических ошибок ассемблер выдаст строку с ошибкой и кратким комментарием к ней на экран дисплея, а также продублирует эту информацию в листинге программы.

Если ошибок нет, можно вызвать редактор связей:

xlink -c8051 TABLO -o TABLO.HEX

Редактор связей сформирует файл TABLO.HEX, который необходим для работы эмулятора.

Загрузка программы в эмулятор и управление его работой

Эмулятор МК51 разработан фирмой AVOSET SISTEMS INC.

Для вызова эмулятора выполнить команду:

avsim51 -c1 a

и загрузить в него программу:

Load Avoset
Enter filename: TABLO.HEX

Для пользователя эмулятор представлен в виде отдельных окон: окна для размещения отлаживаемой программы (левое окно) и набора окон для программно доступных ресурсов микроконтроллера.

Эмулятор работает в двух режимах: в командном режиме и в режиме окна. Переключение режимов осуществляется клавишей Esc.

В командном режиме пользователю предлагается меню из набора команд, выбор которых осуществляется с помощью курсора. Вот некоторые из них:

- Load — загрузка отлаживаемых файлов в эмулятор (используется подкоманда Avoset);
- Patch — позволяет подключать кросс-ассемблер для записи в командной строке мнемокода команды с автоматической ее трансляцией и помещением в окно программы;

- Dump — выбор ячейки памяти верхнего (1) или нижнего (2) окна Data Space. Позволяет установить начало окна на любую ячейку памяти (подкоманда Abcolute);
- Reset — сброс. Возможно осуществить системный сброс контроллера, счетчика циклов (счетчик циклов позволяет оценить реальное время выполнения программы в машинных циклах, в то время как при работе эмулятора она выполняется в замедленном масштабе по времени);
- Set — установка. Позволяет запустить счетчик циклов;
- Memory — позволяет производить очистку или заполнение любых областей памяти;
- Quit — выход из эмулятора с помощью подкоманды Exit.

Для отказа от выполнения команды и выхода в основное меню нажать **Ctrl+C**.

В режиме окна с помощью курсора возможен выбор любых программно доступных ресурсов с целью изменения их содержимого в двоичном, шестнадцатеричном или ASCII кодах. Также возможно задание кодов команд с целью изменения текста программы (дизассемблирование).

Для редактирования в режиме окна могут применяться клавиши:

- INS — инверсия бита, полубайта;
- +/– — инкремент/декремент бита, полубайта;
- ↑→↓← — движение курсора в окне;
- Ctrl+A — быстрый доступ к аккумулятору;
- Ctrl+B — быстрый доступ к расширителю аккумулятора B;
- Ctrl+T — быстрый доступ к ресурсам таймеров;
- Ctrl+I — быстрый доступ к системе прерываний;
- Ctrl+P — быстрый доступ к программному счетчику;
- Alt+P — быстрый доступ к портам.

Для управления работой эмулятора в режиме отладки служат функциональные клавиши. Их назначение:

- F1 — запуск программы в автоматическом режиме;
- F10 — пошаговое выполнение команд;
- F5 — переключение скорости выполнения программы.

Программа работы

1. Скопировать для выполнения лабораторных работ файлы из директории МК51 в свою рабочую папку. Рассчитав номер варианта (от N=1 до N=10), создать программу TEST.ASM:

```
                ; Программа тестирования ассемблера
                MASC    EQU     N
                        DB      11111111B,377Q,255,0FFH
                        DB      “BEGIN”
                        DB      RS1,PSW.4,0D0H.4,0D4H
                        DW      0,1234H,1000
                        ORG     30H
1LABEL:         SJMP    $
                        ORL     A,#MASC
                M1:     CJNE    A,P1,M1
                        STRT    CNT
                        MOV     TH1,#HIGH(NOT(10000)+1)
                        MOV     TL1,#LOW(NOT(10000)+1)
                        MOV     B,#(15*5-MASC)
                ЦИКЛ:   JMP     ЦИКЛ
                        END
```

Прокомментировать результаты ассемблирования программы и сообщения об ошибках (создать и проанализировать файл TEST.LST). Выполняет ли ассемблер в поле операндов арифметические команды сложения, вычитания, умножения? Может ли ассемблер выполнять логические функции при обработке операндов? Как можно отредактировать текст программы, чтобы ассемблер не выдавал сообщения об ошибках?

2. Составить комментарий к работе следующей программы:

```
                MOV     R7,#16
                MOV     R0,#20H
                MOV     R1,#3FH
M1:             MOV     A,@R0
                MOV     @R1,A
                INC     R0
                DEC     R1
```

```
DJNZ    R7,M1
SJMP    $
END
```

Пояснить результаты трансляции команд, размещенных в 8 и 9 строках программы. Что изменится, если вместо SJMP $ записать команду AJMP $ или LJMP $?

Проверить работу программы на эмуляторе, предварительно заполнив массив ячеек резидентной памяти данных с 20H по 2FH числами 00,11,22,33,44,55,66,77,88,99,AA,BB,CC,DD,EE,FF.

3. Создать файл TABLO.ASM. Проассемблировать программу и проанализировать ее листинг. Сформировать файл TABLO.HEX, загрузить программу в эмулятор и запустить на выполнение. Объяснить изменение содержимого ячеек РПД при выполнении программы. Что отображают команды, расположенные в окне памяти программ по адресам 20H-2FH?

4. В режиме Path Code ввести в эмулятор программу умножения 16-разрядного двоичного числа (DPTR) на 8-разрядное (байт в R0). 24-разрядный результат формируется в регистрах: R1 (старший байт), R2 (средний байт), R3 (младший байт).

```
MOV     A,DPL
MOV     B,R0
MUL     AB
MOV     R3,A
MOV     R7,B
MOV     A,DPH
MOV     B,R0
MUL     AB
ADD     A,R7
MOV     R2,A
CLR     A
ADDC    A,B
MOV     R1,A
```

Проверить работу программы в пошаговом режиме на тестовом примере (например, множимое — 10000=2710H, множитель — 100=64H, произведение — 1000000=0F4240H). Пояснить

алгоритм формирования произведения. Оценить время выполнения программы.

5. Битовый процессор МК51 позволяет установить, очистить или проинвертировать любой программно доступный бит, проводить с битами логические операции И, ИЛИ (один из операндов в этом случае должен находиться в триггере переноса С), осуществлять условные переходы по нулевому или единичному состоянию тестируемого бита.

Разработать и отладить на эмуляторе индивидуальное задание (одно из следующих в соответствии со своим вариантом):

1) бит P2.0 должен обнулиться, если не менее, чем на пяти линиях порта P1 установлены нулевые уровни;

2) на линии P1.7 сформировать бит контроля четности для семиразрядного сообщения, выводимого на младшие линии этого порта из регистра B;

3) бит P0.7 должен обнулиться, если на любых пяти из семи оставшихся линий этого порта установлены нулевые уровни;

4) бит P0.7 должен соответствовать логической функции $F=X \oplus Y$, где X и Y — сигналы, подаваемые на младшие линии этого порта;

5) в ячейки 20H-27H РПД занесена информация о состоянии 64 датчиков. Содержимое порта P0 должно обнулиться, если число датчиков с единичным уровнем сигнала превышает число датчиков с нулевым уровнем;

6) в ячейки 20H-2FH РПД занесена информация о состоянии 128 датчиков. Сформировать на линии P1.0 прямоугольные импульсы, если число датчиков с нулевым уровнем превышает 10;

7) в регистре DPTR сформировать разность двухбайтового числа (содержимое портов P0 и P1) и однобайтового (содержимое порта P2);

8) разработать программу, формирующую в регистре DPTR дополнительный код числа минус 5000 (двухбайтовый формат);

9) получить на линиях порта P1 эффект бегущей единицы со сменой направления;

10) заполнить все четные элементы РПД логическими нулями, а нечетные — единицами.

Контрольные вопросы

- Чему равно содержимое регистров МК51 после системного сброса?
- Чем отличаются друг от друга команды MOV R5,7 и MOV 5,#7?
- Транслировать команду JBC F0,$-7.
- Как выполняется команда MUL AB?
- С помощью каких команд можно прочитать в регистр B информацию с датчиков, подключенных к линиям порта P1?
- Как в режиме окна заполнить исходной информацией массив РПД с адресами 70H…7FH?

Содержание отчета

Отчет должен содержать листинги отлаживаемых программ (в том числе и по индивидуальному заданию), ответы на контрольные вопросы и комментарии по ходу выполнения пунктов программы работы.

Совет. При ответе на контрольные вопросы и составлении программы, реализующей индивидуальное задание, полезно познакомиться с примерами, приведенными в разделах 5 и 6 настоящего пособия. Машинные коды команд (коды операций КОП) приведены в разделе 4.

4 ТАЙМЕРЫ И СИСТЕМА ПРЕРЫВАНИЙ МК51 (ЛАБОРАТОРНАЯ РАБОТА №2)

Цель работы. Целью лабораторной работы является исследование работы таймеров/счетчиков событий в различных режимах и системы прерываний микроконтроллеров семейства MCS-51 с помощью персонального компьютера и программных средств отладки.

Таймеры/счетчики событий MCS-51

При работе в качестве таймера содержимое Т/С инкрементируется в каждом машинном цикле, т.е. через 1 мкс при частоте кварца 12 МГц.

При работе в качестве счетчика событий содержимое Т/С инкрементируется под воздействием перехода из 1 в 0 внешнего входного сигнала, подаваемого на входы Т0 (Р3.4) или Т1 (Р3.5). Для управления режимами работы Т/С и организации взаимодействия таймеров с системой прерывания используются регистры **TMOD** (рис. 4.1) и **TCON** (табл. 4.1). С помощью битов М1 и М0 задаются четыре возможных режима работы Т/С0 и Т/С1.

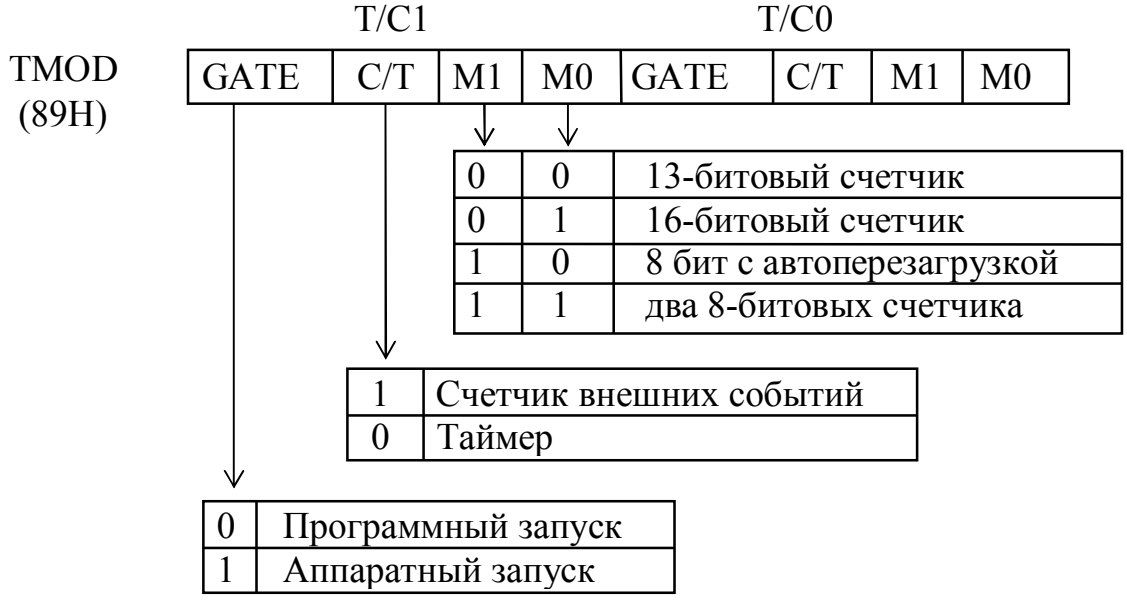
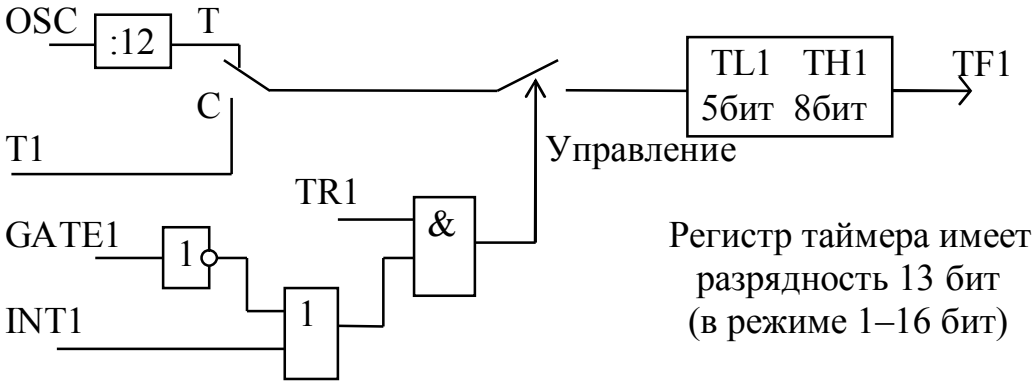
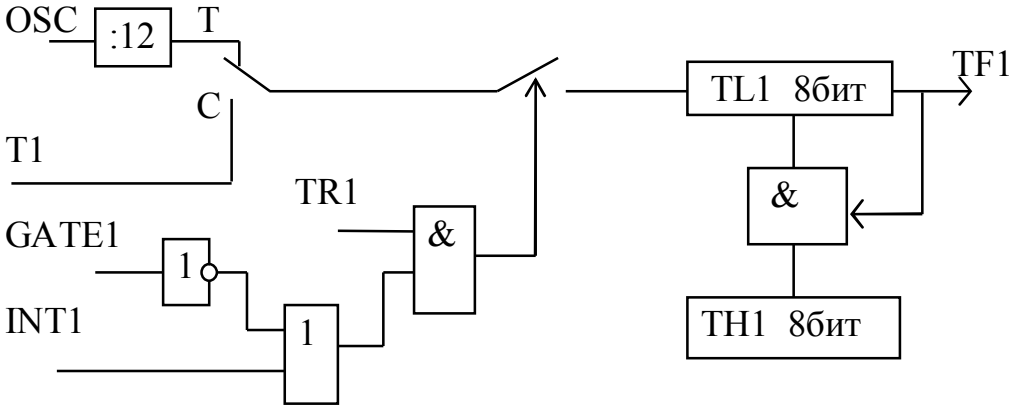


Рис. 4.1 — Формат управляющего слова таймеров

T/C1 в режиме 0



T/C1 в режиме 2



T/C0 в режиме 3

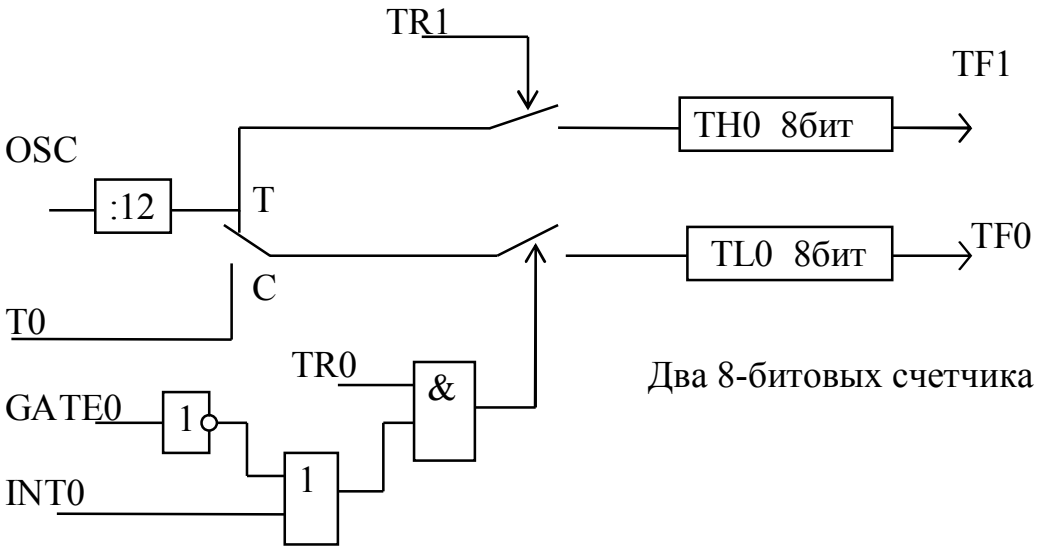


Рис. 4.2 — Режимы работы таймеров

Режим 0. Регистр таймера имеет разрядность 13 бит. При его переходе из состояния «все единицы» в состояние «все нули» устанавливается флаг TF. Работа T/C разрешена, когда TR=1 и либо GATE=0, либо INT=1.

Режим 1. Отличается от режима 0 тем, что регистр таймера имеет разрядность 16 бит.

Режим 2. TL работает как 8-битовый автоперезагружаемый таймер/счетчик. TH хранит значение, которое должно быть перезагружено в TL каждый раз по его переполнению.

Режим 3. TL0 работает как 8-битовый T/C и его режим определяется управляющими битами T/C0. TH0 работает только как 8-битовый таймер, управляемый битом TR1 и использующий флаг TF1. Работа T/C1 постоянно разрешена в режимах 0,1 и 2 без использования прерываний.

Символические обозначения программно доступных битов некоторых регистров приведены в табл. 4.1.

Таблица 4.1

Регистр	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
P3	RD	WR	T1	T0	INT1	INT0	TxD	RxD
PSW	C	AC	F0	RS1	RS0	OV	–	P
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
IE	EA	–	–	ES	ET1	EX1	ET0	EX0
IP	–	–	–	PS	PT1	PX1	PT0	PX0
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Система прерываний МК51

Упрощенная схема прерываний показана на рис. 4.3. В системе прерываний задействованы некоторые биты регистров TCON, IE, IP, SCON.

Внешние прерывания по входам INT0 и INT1 могут быть вызваны либо уровнем (0), либо фронтом (переход из 1 в 0) сигналов на выводах P3.2, P3.3, что определяется программированием битов IT0 и IT1 регистра TCON. При поступлении запроса внешнего прерывания устанавливаются флаги IE0 или IE1. При прерываниях по фронту эти флаги сбрасываются аппаратно при

обращении к подпрограмме обслуживания, при прерываниях по уровню флаги очищаются при снятии запроса внешнего прерывания.

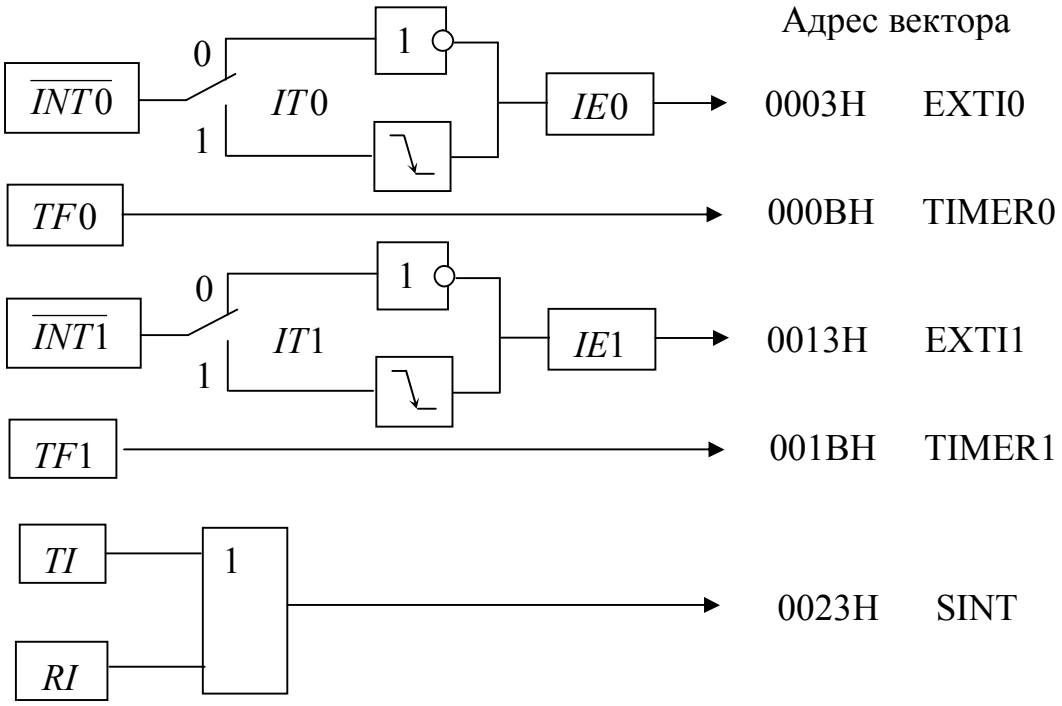


Рис. 4.3 — Система прерываний

В блоке регистров специальных функций есть два регистра, предназначенных для управления режимом прерываний (IE) и уровнями приоритета (IP). Установка бита EA снимает общую блокировку прерываний. При EA=1 прерывания могут быть разрешены индивидуальными разрешениями EX0, ET0, EX1, ET1, ES. Установка одного из битов PX0, PT0, PX1, PT1, PS присваивает соответствующему источнику прерываний высший приоритет.

Прерывания от таймеров/счетчиков вызываются при их переполнении установкой флагов TF0 и TF1. Очистка этих флагов происходит автоматически при обращении к подпрограммам обслуживания. Флаги запросов прерываний приемника и передатчика последовательного порта RI и TI устанавливаются аппаратно, но сбрасываться должны программой.

Флаги IE0, IE1, TF0, TF1, RI и TI устанавливаются независимо от того, разрешено или нет соответствующее прерывание в регистре IE. Выполнение подпрограммы обслуживания с низким уровнем приоритета прерывается при поступлении запроса с высшим уровнем приоритета.

Если прерывания разрешены и сформировался соответствующий флаг, система прерываний помещает в стек содержимое счетчика команд PC и загружает в счетчик команд адрес вектора подпрограммы обслуживания. По адресу вектора обычно располагается команда безусловной передачи управления к начальному адресу подпрограммы обслуживания прерывания. Подпрограмма обслуживания должна в случае необходимости начинаться командами записи в стек (PUSH) PSW, ACC, B, DPTR и заканчиваться командами восстановления их из стека (POP). Подпрограммы обслуживания обязательно заканчиваются командой RETI, снимающей блокировку прерываний. При выполнении подпрограммы обслуживания прерываний действует блокировка запросов прерываний от других источников. Если флаг прерывания был установлен, но не получил обслуживания и к моменту окончания блокировки уже был сброшен, то запрос прерывания теряется и нигде не запоминается.

Отметим, что после системного сброса указатель стека настроен на область первого банка регистров общего назначения. Поэтому, при использовании в программе подпрограмм, в том числе обслуживания прерываний, и банка РОН1 содержимое указателя стека необходимо модифицировать (например, MOV SP,#100).

Программа работы

1. Зафиксировать содержимое регистров, флагов и ячеек памяти микроконтроллера после загрузки эмулятора (**avsim51 –c1 a**). Чему равно содержимое указателя стека? Разрешены ли прерывания? На какой режим настроены таймеры? Какая установлена скорость выполнения программы?

2. Составить комментарий к программе преобразования двоичного числа, задаваемого на линиях порта P1, в двоично-десятичное содержимое регистра DPTR:

```
MOV     A,P1
MOV     B,#100
DIV     AB
MOV     DPH,A
```

```
MOV     A,#10
XCH     A,B
DIV     AB
SWAP    A
ORL     A,B
MOV     DPL,A
```

В режиме Patch Code ввести текст программы в эмулятор и проверить ее работу в пошаговом (F10) режиме. Как выполняется команда деления? Какие флаги PSW изменяются при выполнении программы?

3. Записать в первые две ячейки памяти программ программу, состоящую из одной команды SJMP 0 (80 FE), и запустить ее на выполнение в автоматическом режиме. Почему не работают таймеры T/C0 и T/C1?

Установив TR0=1, проверить работу T/C0 в режиме таймера (скорость счета изменяется клавишей F5) и счетчика событий (TMOD.2=1). Перепады на линии T0 (P3.4) формировать с помощью клавиши Insert. В каком диапазоне изменяется содержимое регистров TL0 и TH0 при работе T/C0 в режиме 0? Когда устанавливается флаг TF0?

Проверить работу T/C1 в режиме 1. Установив TR1=1 и GATE1=1, проверить возможность аппаратного управления работой таймера уровнем сигнала на входе INT1 (P3.3).

Перевести T/C0 в режим 2 (8-битный автоперезагружаемый таймер/счетчик). Установив (TH0)=0D5H, проследить работу T/C0 в режиме таймера и счетчика событий.

Перевести T/C0 в режим 3 (TL0 и TH0 функционируют как два независимых 8-битных счетчика). Возможно ли в этом режиме использование прерываний от T/C1?

4. В режиме Patch Code ввести в эмулятор текст программы, при реализации которой регистры R0, R1, R2, R3, R4 фиксируют число выполнения подпрограмм обслуживания прерываний от различных источников, а аккумулятор работает в режиме двоичного счетчика:

```
ORG   00H
INC   A
SJMP 0
ORG   03H
INC   R0
RETI
ORG   0BH
INC   R1
RETI
ORG   13H
INC   R2
RETI
ORG   1BH
INC   R3
RETI
ORG   23H
INC   R4
CLR   SCON.0
CLR   SCON.1
RETI
```

Разрешить прерывания по входу INT0, установив в режиме окна EA=1 и EX0=1. При работе программы в автоматическом режиме исследовать различие механизма обработки прерывания при IT0=0 и IT0=1 (по уровню и по срезу P3.2).

Разрешить прерывания и по входу INT1. При IT0=IT1=0 установить INT0=INT1=0 и запустить программу. Почему не выполняется подпрограмма обслуживания прерываний по входу INT1? Повторить работу программы, установив в регистре приоритетов прерываний PX1=1.

Разрешить все прерывания. Установить TR0=TR1=IT0= =IT1=1. Запустить программу на выполнение. Убедиться, что периодически выполняются подпрограммы обслуживания прерываний по переполнению таймеров. Что происходит при изменении содержимого буферных регистров приемника и передатчика последовательного порта SBUF? Проимитировать внешние прерывания по входам INT0 и INT1.

Остановить выполнение программы. Установить все флаги прерываний (IE0, IE1, TF0, TF1, RI и TI). Продолжить выполне-

ние программы в пошаговом режиме. Объяснить поведение микроконтроллера. В какой момент сбрасываются флаги IE0, IE1, TF0, TF1(при передаче управления подпрограмме обслуживания или по команде RETI)? Повторить эксперимент, установив в регистре приоритетов PS=1. Объяснить новую последовательность выполнения подпрограмм обслуживания прерываний. Что будет, если при выполнении подпрограммы обслуживания прерываний пришел запрос прерываний с большим приоритетом? Нужно ли сбрасывать программно флаги TF0, TF1, RI и TI?

5. Испытать на эмуляторе работу следующей программы, формирующей в аккумуляторе двоично-десятичный код длительности импульса (единицы и десятые доли мс) на входе INT0:

```
        ORG     00H             ; RESET
        MOV     TH0,#156        ; Загрузка регистров T/C0
        MOV     TL0,#0
        MOV     TMOD,#0AH       ; Настройка T/C0 на режим 2
        SJMP    M1
        ORG     0BH             ; Вектор прерывания от T/C0
        ADD     A,#1            ; Подпрограмма обслуживания
        DA      A               ; прерываний
        RETI                    ; Возврат из подпрограммы
M1:     CLR     A               ; Очистка аккумулятора
        MOV     IE,#82H         ;Разрешение прерываний от T/C0
        SETB    TR0             ; Запуск таймера T/C0
        SJMP    $               ; Зацикливание программы
```

Начиная с адреса 0BH записана подпрограмма обслуживания прерываний по таймеру T/C0. После каждого переполнения таймера (т.е. через каждые 100 мкс при частоте кварца 12 МГц) содержимое двоично-десятичного счетчика, организованного в аккумуляторе, увеличивается на единицу. Основная программа начинается с нулевой ячейки, при выполнении обходит ячейки, занятые подпрограммой, и заканчивается командой SJMP $.

Таймер T/C0 настраивается на режим 8-разрядного счетчика с автоперезагрузкой и возможностью аппаратного запуска логической 1 на входе INT0 (перед запуском программы на этом входе надо зафиксировать логический 0). В регистр TH0 загружается дополнительный код числа минус 100.

Проимитировав на входе INT0 импульс длительностью 10 мс, измерить секундомером реальное время работы программы при наивысшей скорости (HI). Во сколько раз скорость воспроизведения программы с помощью эмулятора отличается от реального масштаба времени?

6. Апробировать программу, реализующую на микроконтроллере К1830ВЕ51 электронные часы с индикацией часов, минут и секунд реального времени. Все операции по решению поставленной задачи выполняет подпрограмма обслуживания прерываний. Остальное время контроллер находится в режиме зацикливания основной программы.

Программа «Часы»

```
; Начальная установка и запуск часов в 00 00 00

        ORG     00H
        MOV     P0,#0           ; Счетчик часов
        MOV     P1,#0           ; Счетчик минут
        MOV     P2,#0           ; Счетчик секунд
        MOV     R0,#100         ; Начальная загрузка
        MOV     R1,#100         ; счетчиков генератора
        MOV     TH1,#9CH        ; секундных импульсов
        MOV     TMOD,#20H       ; T/C1 в режиме 2
        MOV     IE,#88H         ; Разрешение
                                ; прерываний от T/C1
        SETB    TR1             ; Старт таймера T/C1
MAIN:   SJMP    MAIN            ; Основная программа

        ; Подпрограмма обслуживания прерываний

        ORG     1BH             ; Вектор прерывания
        DJNZ    R0,EXIT         ; Задержка в одну
        MOV     R0,#100         ; секунду
        DJNZ    R1,EXIT
        MOV     R1,#100
        JNB     T0,M1           ; Коррекция минут
        JNB     T1,M2           ; Коррекция часов
        MOV     A,P2            ; Счетчик секунд
        ADD     A,#1
        DA      A
        MOV     P2,A
        CJNE    A,#60H,EXIT
        MOV     P2,#0
```

```
        M1:   MOV   A,P1          ; Счетчик минут
        ADD   A,#1
        DA    A
        MOV   P1,A
        CJNE  A,#60H,EXIT
        MOV   P1,#0
   M2:  MOV   A,P0           ; Счетчик часов
        ADD   A,#1
        DA    A
        MOV   P0,A
        CJNE  A,#24H,EXIT
        MOV   P0,#0
 EXIT:  RETI                ; Возврат из п/п прерываний
        END
```

При отладке программы с помощью эмулятора ход часов замедлен. Для ускорения процессов рекомендуется в регистры R0 и R1 загружать число 3, а не 100.

После старта программы производится начальная загрузка регистров секундной задержки, а также счетчиков секунд, минут и часов. Таймер/счетчик T/C1 настраивается на работу в режиме 2, когда TL1 работает как 8-битовый автоперезагружаемый таймер, а TH1 хранит значение, которое перезагружается в TL1 каждый раз по переполнении. Разрешаются прерывания от T/C1, и после его запуска они происходят через каждые 100 машинных циклов (100 мкс при частоте кварца 12 МГц), вызывая выполнение подпрограммы обслуживания с начальным адресом 1BH. Через 10000 прерываний, которые подсчитывают счетчики на регистрах R0 и R1, т.е. ежесекундно, меняется содержимое порта P2, определяющее показания цифрового индикатора секунд.

Двоично-десятичный счетчик минут реализован с помощью порта P1, аналогичный счетчик часов — с помощью P0. При включении контроллера счетчики сбрасываются и на цифровые индикаторы заносятся нули. Установка реального времени производится в определенной последовательности. Сначала держат лог. 0 на входе T1 до тех пор, пока индикаторы покажут требуемое число часов. Затем держат лог. 0 на входе T0 до тех пор, пока не высветятся нужные цифры минут. Коррекция осуществляется подачей секундных импульсов на счетчики часов и минут.

Оценить минимальное и максимальное время выполнения подпрограммы обслуживания прерываний.

Контрольные вопросы

- Разрешены ли прерывания после системного сброса?
- Может ли быть прервано выполнение программы обработки прерывания с высоким уровнем приоритета?
- Транслировать команду JB TF0,$+5.
- Что происходит при выполнении команды CJNE A,#40,M1?
- Какой флаг устанавливается после выполнения команды MOV SBUF,B?
- Какими командами можно загрузить в T/C0 дополнительный код числа 10000?

Содержание отчета

Отчет должен содержать листинги отлаживаемых программ, ответы на контрольные вопросы и комментарии по ходу выполнения пунктов программы работы.

Машинные коды команд МК51

Мнемоника		КОП	Мнемоника		КОП
ACALL	0xxH	11	ANL	C,/bit	B0
ACALL	1xxH	31	AJMP	0xxH	01
ACALL	2xxH	51	AJMP	1xxH	21
ACALL	3xxH	71	AJMP	2xxH	41
ACALL	4xxH	91	AJMP	3xxH	61
ACALL	5xxH	B1	AJMP	4xxH	81
ACALL	6xxH	D1	AJMP	5xxH	A1
ACALL	7xxH	F1	AJMP	6xxH	C1
ADD	A,ad	25	AJMP	7xxH	E1
ADD	A,R0	28	CJNE	A,ad,adr	B5
ADD	A,R1	29	CJNE	A,#d,adr	B4
ADD	A,R2	2A	CJNE	R0,#d,adr	B8
ADD	A,R3	2B	CJNE	R1,#d,adr	B9
ADD	A,R4	2C	CJNE	R2,#d,adr	BA
ADD	A,R5	2D	CJNE	R3,#d,adr	BB
ADD	A,R6	2E	CJNE	R4,#d,adr	BC
ADD	A,R7	2F	CJNE	R5,#d,adr	BD
ADD	A,@R0	26	CJNE	R6,#d,adr	BE
ADD	A,@R1	27	CJNE	R7,#d,adr	BF
ADD	A,#d	24	CJNE	@R0,#d,adr	B6
ADDC	A,ad	35	CJNE	@R1,#d,adr	B7
ADDC	A,R0	38	CLR	A	E4
ADDC	A,R1	39	CLR	bit	C2
ADDC	A,R2	3A	CLR	C	C3

Мнемоника		КОП	Мнемоника		КОП
ADDC	A,R3	3B	CPL	A	F4
ADDC	A,R4	3C	CPL	bit	B2
ADDC	A,R5	3D	CPL	C	B3
ADDC	A,R6	3E	DA	A	D4
ADDC	A,R7	3F	DEC	A	14
ADDC	A,@R0	36	DEC	ad	15
ADDC	A,@R1	37	DEC	R0	18
ADDC	A,#d	34	DEC	R1	19
ANL	A,ad	55	DEC	R2	1A
ANL	A,R0	58	DEC	R3	1B
ANL	A,R1	59	DEC	R4	1C
ANL	A,R2	5A	DEC	R5	1D
ANL	A,R3	5B	DEC	R6	1E
ANL	A,R4	5C	DEC	R7	1F
ANL	A,R5	5D	DEC	@R0	16
ANL	A,R6	5E	DEC	@R1	17
ANL	A,R7	5F	DIV	AB	84
ANL	A,@R0	56	DJNZ	ad,adr	D5
ANL	A,@R1	57	DJNZ	R0,adr	D8
ANL	A,#d	54	DJNZ	R1,adr	D9
ANL	ad,A	52	DJNZ	R2,adr	DA
ANL	ad,#d	53	DJNZ	R3,adr	DB
ANL	C,bit	82	DJNZ	R4,adr	DC
DJNZ	R5,adr	DD	MOV	add,ads	85
DJNZ	R6,adr	DE	MOV	bit,C	92
DJNZ	R7,adr	DF	MOV	C,bit	A2
INC	A	04	MOV	DPTR,#d16	90
INC	ad	05	MOV	R0,ad	A8
INC	DPTR	A3	MOV	R0,A	F8
INC	R0	08	MOV	R0,#d	78
INC	R1	09	MOV	R1,A	F9
INC	R2	0A	MOV	R1,ad	A9
INC	R3	0B	MOV	R1,#d	79
INC	R4	0C	MOV	R2,A	FA
INC	R5	0D	MOV	R2,ad	AA
INC	R6	0E	MOV	R2,#d	7A
INC	R7	0F	MOV	R3,A	FB
INC	@R0	06	MOV	R3,ad	AB
INC	@R1	07	MOV	R3,#d	7B
JB	bit,adr	20	MOV	R4,A	FC
JBC	bit,adr	10	MOV	R4,ad	AC
JC	adr	40	MOV	R4,#d	7C
JMP	@A+DPTR	73	MOV	R5,A	FD
JNB	bit,adr	30	MOV	R5,ad	AD
JNC	adr	50	MOV	R5,#d	7D
JNZ	adr	70	MOV	R6,A	FE
JZ	adr	60	MOV	R6,ad	AE
LCALL	ad16	12	MOV	R6,#d	7E
LJMP	ad16	02	MOV	R7,A	FF

Мнемоника		КОП	Мнемоника		КОП
MOV	A,ad	E5	MOV	R7,ad	AF
MOV	A,R0	E8	MOV	R7,#d	7F
MOV	A,R1	E9	MOV	@R0,A	F6
MOV	A,R2	EA	MOV	@R0,ad	A6
MOV	A,R3	EB	MOV	@R0,#d	76
MOV	A,R4	EC	MOV	@R1,A	F7
MOV	A,R5	ED	MOV	@R1,ad	A7
MOV	A,R6	EE	MOV	@R1,#d	77
MOV	A,R7	EF	MOVC	A,@A+DPTR	93
MOV	A,@R0	E6	MOVC	A,@A+PC	83
MOV	A,@R1	E7	MOVX	A,@DPTR	E0
MOV	A,#d	74	MOVX	A,@R0	E2
MOV	ad,A	F5	MOVX	A,@R1	E3
MOV	ad,R0	88	MOVX	@DPTR,A	F0
MOV	ad,R1	89	MOVX	@R0,A	F2
MOV	ad,R2	8A	MOVX	@R1,A	F3
MOV	ad,R3	8B	MUL	AB	A4
MOV	ad,R4	8C	NOP		00
MOV	ad,R5	8D	ORL	A,ad	45
MOV	ad,R6	8E	ORL	A,R0	48
MOV	ad,R7	8F	ORL	A,R1	49
MOV	ad,@R0	86	ORL	A,R2	4A
MOV	ad,@R1	87	ORL	A,R3	4B
MOV	ad,#d	75	ORL	A,R4	4C
ORL	A,R5	4D	SUBB	A,@R0	96
ORL	A,R6	4E	SUBB	A,@R1	97
ORL	A,R7	4F	SUBB	A,#d	94
ORL	A,@R0	46	XCH	A,ad	C5
ORL	A,@R1	47	XCH	A,R0	C8
ORL	A,#d	44	XCH	A,R1	C9
ORL	ad,A	42	XCH	A,R2	CA
ORL	ad,#d	43	XCH	A,R3	CB
ORL	C,bit	72	XCH	A,R4	CC
ORL	C,/bit	A0	XCH	A,R5	CD
POP	ad	D0	XCH	A,R6	CE
PUSH	ad	C0	XCH	A,R7	CF
RET		22	XCH	A,@R0	C6
RETI		32	XCH	A,@R1	C7
RL	A	23	XCHD	A,@R0	D6
RLC	A	33	XCHD	A,@R1	D7
RR	A	03	XRL	A,ad	65
RRC	A	13	XRL	A,R0	68
SETB	bit	D2	XRL	A,R1	69
SETB	C	D3	XRL	A,R2	6A
SJMP	adr	80	XRL	A,R3	6B
SWAP	A	C4	XRL	A,R4	6C
SUBB	A,ad	95	XRL	A,R5	6D
SUBB	A,R0	98	XRL	A,R6	6E
SUBB	A,R1	99	XRL	A,R7	6F

Мнемоника		КОП	Мнемоника		КОП
SUBB	A,R2	9A	XRL	A,@R0	66
SUBB	A,R3	9B	XRL	A,@R1	67
SUBB	A,R4	9C	XRL	A,#d	64
SUBB	A,R5	9D	XRL	ad,A	62
SUBB	A,R6	9E	XRL	ad,#d	63
SUBB	A,R7	9F			

На рис. 4.4. показан граф возможных пересылок данных, который иллюстрирует структуру информационных связей в микроконтроллере.

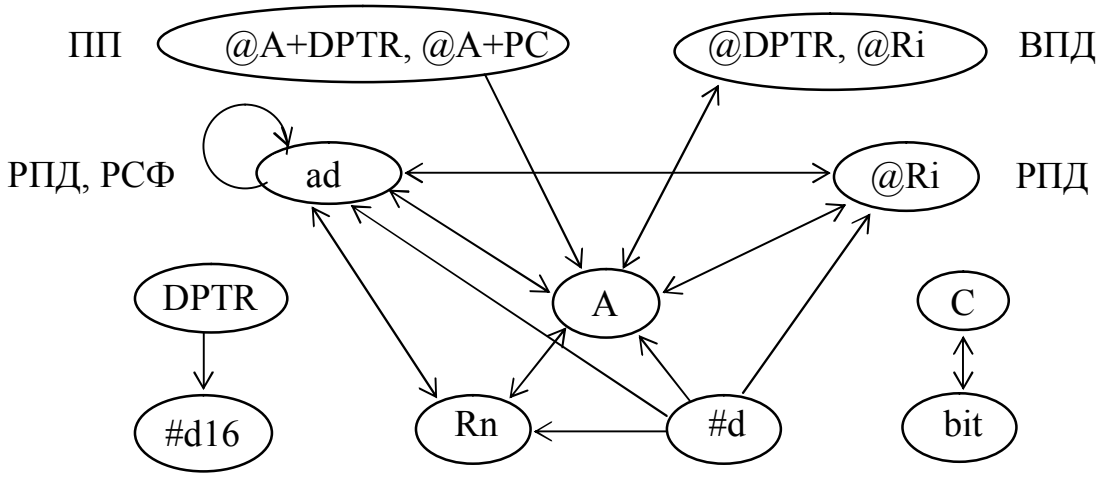


Рис. 4.4 — Пути передачи данных в МК51

Операнды, участвующие в операциях пересылки, различаются по месту расположения и способу адресации.

Взаимодействие с памятью программ ПП (только для чтения) и внешней памятью данных ВПД (для чтения и записи) осуществляется только через аккумулятор А с использованием косвенной адресации.

К ячейкам резидентной памяти данных РПД можно обратиться как с помощью косвенной адресации (в качестве указателей памяти используются регистры R0 или R1), так и с помощью прямой адресации.

Прямая адресация байтов используется при обращении к регистрам специальных функций РСФ.

При работе с битами всегда используется прямая адресация битов. Пересылка битов возможна только через триггер переноса С.

Шестнадцатиразрядный операнд может быть непосредственно загружен в регистр-указатель данных DPTR.

5 УПРАЖНЕНИЯ ПО РЕШЕНИЮ ЗАДАЧ

Задача 1. Какие выводы микроконтроллера используются при выборке очередного байта из внешней памяти программ?

Ответ. Младший байт адреса передается через линии порта P0 и запоминается во внешнем регистре по сигналу, формируемому на выводе ALE. Старший байт адреса передается через линии порта P2. Читается очередной байт команды через линии порта P0 по сигналу, формируемому на выводе PME.

Задача 2. Какие выводы микроконтроллера задействованы при выполнении команды MOVX @R0,A?

Ответ. При выполнении этой команды содержимое аккумулятора записывается в ячейку внешней памяти данных объемом не более 256 байт, косвенно адресуемую содержимым регистра R0. Адрес ячейки защелкивается во внешнем регистре с линий порта P0 по сигналу, формируемому на выводе ALE. Запись осуществляется через линии порта P0 по сигналу WR, формируемому на шестом выводе порта P3.

Задача 3. Какие выводы микроконтроллера задействованы при работе с внутренней памятью программ?

Ответ. Такой режим устанавливается подачей высокого уровня напряжения на вывод VPP. Порты P0 и P2 остаются свободными для других применений, так как адреса и данные передаются по внутренней магистрали микроконтроллера.

Задача 4. Какой из битов PSW программно доступен только по чтению?

Ответ. Это может быть только бит паритета P. Его значение жестко определяется содержимым аккумулятора. В 9-разрядном слове, состоящем из восьми разрядов аккумулятора и бита P, всегда содержится четное число единичных битов. При обнулении аккумулятора флаг P примет нулевое значение.

Задача 5. Можно ли использовать порт P3 для реализации альтернативных функций после выполнения команды MOV P3,#0?

Ответ. Нет, нельзя. Альтернативная функция любой из линий порта P3 реализуется только в том случае, если в соответствующем этой линии триггере-защелке содержится 1. После выполнения данной команды во все триггеры порта будут записаны нули, выходные ключи порта будут открыты и на всех линиях P3 будут уровни логического 0.

Задача 6. Содержимое каких регистров МК51 изменится, если сразу после сброса будет выполнена команда LCALL BEGIN?

Ответ. Трехбайтовая команда LCALL вызывает подпрограмму BEGIN. При сбросе в указатель стека SP загружается число 7. По команде LCALL к содержимому счетчика команд PC прибавляется 3 для получения адреса следующей команды и после этого полученный 16-битовый результат (в нашем случае 0003H) помещается в стек (03H — в восьмую, а 00H — в девятую ячейку резидентной памяти данных). Содержимое указателя стека увеличивается на 2. Затем старший и младший байты счетчика команд загружаются соответственно вторым и третьим байтом команды LCALL, т.е. начальным адресом подпрограммы BEGIN.

Задача 7. Определите прямой адрес пятого бита аккумулятора.

Решение. Прямой адрес младшего бита регистра, допускающего адресацию отдельных бит, совпадает с прямым адресом самого регистра. Добавляя число 5 к прямому адресу ACC, равному 0E0H, получим прямой адрес бита ACC.5, равный 0E5H. Прямая побитовая адресация используется для обращения к отдельно адресуемым 128 битам, расположенным в ячейках резидентной памяти данных с адресами 20H-2FH, и к отдельно адресуемым битам регистров специального назначения.

Задача 8. Можно ли использовать команды работы с битами при обращении к младшему биту регистра PCON, имеющему символическое обозначение IDL (бит холостого хода)?

Ответ. Нельзя, так как регистр PCON не допускает адресацию отдельных бит. Такую адресацию допускают только 11 регистров специальных функций из 21.

Задача 9. Какие регистры используются в качестве указателей данных при косвенно-регистровой адресации?

Ответ. При обращении к ячейкам внутреннего ОЗУ данных для этой цели используются регистры R0, R1 выбранного банка регистров. Они же используются для выборки ячейки из блока в 256 байт внешней памяти данных. При обращении к любой ячейке адресного пространства внешней памяти данных объемом до 64 Кбайт используется 16-разрядный регистр DPTR. Любой байт из таблицы памяти программ может быть выбран по адресу, определяемому суммой содержимого DPTR или PC и содержимого аккумулятора.

Задача 10. Какие методы адресации можно использовать при обращении к ячейке резидентной памяти данных с адресом 18H?

Ответ. Кроме прямой байтовой адресации (как, например, в команде MOV 18H,#100), можно использовать косвенно-регистровую (MOV @R1,#100 если содержимое регистра-указателя R1 равно 18H) или регистровую (MOV R0,#100 если выбран банк РОН3 установкой в 1 битов RS0 и RS1 слова состояния программы PSW).

Задача 11. Определить машинные коды команды ANL A,B.

Ответ. Регистр-расширитель аккумулятора B допускает только прямую байтовую адресацию, следовательно, это двухбайтовая команда типа 3. Первый байт является кодом операции (55H), второй байт — прямым адресом регистра B (F0H).

Задача 12. Транслировать команду CLR F0.

Ответ. F0 — символическое имя бита PSW.5. Следовательно, это двухбайтовая команда типа 4. Код операции — C2H, прямой адрес бита D5H определяем, прибавляя число 5 к прямому адресу регистра PSW D0H.

Задача 13. Определить второй байт команды SJMP $.

Решение. Команда «короткий переход»« выполняет безусловный переход в программе по указанному адресу. Знак $ в поле операндов используется для обозначения текущего содержимого

программного счетчика PC (он равен адресу первого байта рассматриваемой команды). В данном случае речь идет о двухбайтовой команде типа 5. Второй байт команды — относительное смещение со знаком (rel) от начального адреса следующей команды до адреса, указанного в команде. В данном примере rel=–2=FEH (rel записывается в дополнительном коде и находится в диапазоне от –128 до +127).

Задача 14. Транслировать команду AJMP 1000.

Решение. Команда «абсолютный переход» передает управление по адресу внутри текущей страницы памяти программ объемом 2 Кбайт. Это двухбайтовая команда типа 6. Заменив ее эквивалентной командой AJMP 3E8H, записываем второй байт E8H. По таблице машинных кодов для модели AJMP 3xxH находим первый байт команды 61H.

Задача 15. Определить тип команды MOV B,P0.

Решение. Это трехбайтовая команда типа 9 (общий вид команды MOV add,ads). Второй байт команды 80H — прямой адрес порта P0 (адрес источника ads). Третий байт команды F0H — прямой адрес регистра B (адрес приемника add).

Задача 16. Что происходит при выполнении команды LCALL 1AB7H?

Решение. Это трехбайтовая команда типа 12, вызывающая подпрограмму, находящуюся по указанному адресу. По команде LCALL к программному счетчику PC прибавляется 3 для получения адреса следующей команды и после этого полученный 16-разрядный результат помещается в стек (в ячейку ((SP+1)) загружается младший байт, в ячейку ((SP+2)) — старший байт. Затем старший и младший байты PC загружаются соответственно вторым (1AH) и третьим (B7H) байтами команды LCALL. Выполнение программы продолжается командой, находящейся по указанному адресу.

Задача 17. Что происходит при выполнении команды CJNE A,#50,M1?

Решение. Это трехбайтовая команда типа 10. Происходит сравнение содержимого аккумулятора с константой, указанной во втором байте команды (32H) и выполняется переход на метку M1, если операнды не равны. Адрес перехода вычисляется при помощи сложения значения (со знаком), указанного в третьем байте команды (rel), с содержимым счетчика команд PC после увеличения его на три. Флаг переноса C устанавливается в 1, если содержимое аккумулятора меньше константы. В противном случае перенос сбрасывается. Команда не оказывает влияния на операнды. При равенстве операндов выполняется следующая команда программы.

Задача 18. Транслировать команду JBC T1,$+10?

Решение. Это трехбайтовая команда типа 11. Код операции 10H. Второй байт команды B5H соответствует прямому адресу бита T1 (P3.5). Третий байт команды 07H показывает, сколько шагов необходимо сделать от адреса следующей команды до адреса перехода (rel=+7). Команда осуществляет переход по указанному адресу, если бит установлен, с последующим сбросом бита. В противном случае выполняется следующая за JBC команда.

Задача 19. Оценить результат выполнения команды ADD A,R6. До выполнения команды (A)=C3H, (R6)=AAH.

Решение. Команда складывает содержимое аккумулятора и регистра R6, оставляя результат в аккумуляторе: (A)=C3H+AAH= =6DH. При этом устанавливаются флаги: AC=0, C=1, OV=1. При сложении целых без знака флаг переноса C=1 указывает на появление переполнения (сумма больше 255). При сложении целых чисел со знаком флаг OV=1 указывает на отрицательную величину, полученную при суммировании двух положительных операндов или на положительную сумму для двух отрицательных операндов (как в нашем случае).

Задача 20. Можно ли с помощью команды DA A преобразовать двоичное число в аккумуляторе в двоично-десятичный код?

Ответ. Нет, нельзя. Команда «десятичная коррекция аккумулятора для сложения» вместе с предшествующей ей командой

ADD или ADDC используется для сложения чисел, представленных в двоично-десятичном формате.

Например, после выполнения команд

```
MOV     A,#37H
MOV     70,#48H
ADD     A,70
DA      A
```

в аккумуляторе получим число 85H, которое можно трактовать как двоично-десятичный код десятичного числа 85, равного сумме десятичных чисел 37 и 48, предварительно записанных в аккумулятор и семидесятую ячейку резидентной памяти данных в двоично-десятичном коде.

Команда DA A корректирует результат предварительного сложения операндов, добавляя 06H, 60H или 66H в зависимости от начального состояния аккумулятора и слова состояния программы PSW (флагов C и AC).

Задача 21. Как выполняется команда MUL AB?

Ответ. Команда «умножение» умножает 8-битовые целые числа без знака из аккумулятора и регистра B. Старший байт 16-битового произведения помещается в регистр B, а младший — в аккумулятор. Флаг переноса C всегда сбрасывается. Флаг переполнения OV устанавливается, если результат больше 255. Время выполнения команды — четыре машинных цикла.

Пусть (A)=(B)=100=64H. Их произведение 10000=2710H. После выполнения команды MUL AB в регистре B будет 27H, в аккумуляторе — 10H. Флаг C=0, флаг OV=1.

Задача 22. Как выполняется команда DIV AB?

Ответ. Команда «деление» делит 8-битовое целое число без знака из аккумулятора A на 8-битовое целое число без знака в регистре B. Аккумулятору присваивается целая часть частного (старшие разряды), а регистру B — остаток. Флаги C и OV сбрасываются. При делении на 0 флаг OV устанавливается, а содержимое A и B становится неопределенным. Время выполнения команды — четыре машинных цикла.

Пусть аккумулятор содержит число 251 (0FBH или 11111011B), а регистр B — число 18 (12H или 00010010B). После выполнения команды DIV AB в аккумуляторе будет число 13 (0DH или 00001101B), а в регистре B — число17 (11H или 00010001B), т.к. 251=(13*18)+17. Флаги C и OV будут сброшены.

Задача 23. Инвертировать сигналы, выводимые в младшую и старшую линии порта P1.

Ответ. Это можно сделать, последовательно выполнив команды инверсии битов CPL P1.0 и CPL P1.7. Одновременно инвертировать оба бита можно логической командой XRL P1,#10000001B. При выполнении команд изменяется содержимое триггера-защелки порта.

Задача 24. Как выполняется команда SUBB A,R0?

Ответ. Команда «вычитание с заемом» вычитает из аккумулятора содержимое R0 вместе с флагом переноса, засылая результат в аккумулятор. Эта команда устанавливает флаг переноса C (заема), если при вычитании для бита 7 необходим заем, в противном случае флаг переноса сбрасывается. Флаг AC устанавливается, если необходим заем для бита 3. Флаг переполнения OV устанавливается, если заем необходим для бита 6, но его нет для бита 7, или есть для бита 7, но нет для бита 6.

При вычитании целых чисел со знаком OV указывает на отрицательное число, которое получается при вычитании отрицательной величины из положительной, или на положительное число, которое получается при вычитании положительного числа из отрицательного.

Задача 25. Оценить время выполнения записанных с нулевого адреса команд МК51 при частоте кварца 12 МГц.

```
                ORG     0
74 05           MOV     A,#5            ; 1
83              MOVC    A,@A+PC         ; 2
D5 E0 FD        DJNZ    ACC,$           ; 2
D5 F0 F9        DJNZ    B,2             ; 2
```

Решение. Слева от мнемоники приведены результаты трансляции команд, в поле комментария – время выполнения команд в машинных циклах. Первый байт содержит код операции. 05 – константа 5, E0 – прямой адрес аккумулятора, F0 – прямой адрес регистра B. FD – относительное смещение (-3) от адреса первого байта следующей команды до первого байта данной команды. F9 – относительное смещение от адреса следующей команды до ячейки с адресом 2, равное (-6). При выполнении второй команды в аккумулятор пересылается байт из ячейки памяти программ, адрес которой определяется путем суммирования содержимого A и PC. В это время в аккумуляторе находится число 5, а в программном счетчике подготовлен адрес следующей команды, т.е. 3. В восьмой ячейке находится число F9=249. Следующая команда при первом проходе выполнится 249 раз. После сброса в регистре B находится 0. Поэтому последняя команда будет выполняться 256 раз. При этом 255 раз будет осуществляться возврат на вторую команду. При ее выполнении в аккумулятор будет попадать число D5=213. Столько раз в цикле каждый раз будет выполняться предпоследняя команда.

Один машинный цикл равен 1 мкс. В итоге получаем

$$t=1+2*256+(2*249+2*255*213)+2*256=110153 \text{ мкс}.$$

Задача 26. Оценить время выполнения команд после сброса МК51и содержимое аккумулятора при частоте кварца 12 МГц.

```
                ORG     0
0,1             MOV     A,#7
2,3,4           MOV     B,#5
5               MOVC    A,@A+PC   ; (A)=F8=248
6,7,8           DJNZ    ACC,$
9,10            PUSH    ACC
11,12,13        DJNZ    B,6       ; rel=F8
14,15           MOV      A,SP
16,17           ADD     A,10
```

Решение. Слева от мнемоники команд записаны адреса ячеек памяти программ, в которых расположены байты, начиная с нулевой ячейки. При выполнении третьей команды в аккумулятор попадает содержимое 13 ячейки, т.е. число F8=248 (относительное смещение от 14 до 6 ячейки). Столько раз выполняется

при первом проходе следующая команда, после чего содержимое аккумулятора обнуляется. Команда PUSH ACC загружает в стек (в 8 ячейку, т.к. после сброса в указателе стека записано число 7) ноль. Команда DJNZ B,6 выполняется 5 раз. При этом 4 раза осуществляется возврат на 6 ячейку при нулевом содержимом аккумулятора, в результате чего обнуляются еще 4 ячейки стека, в том числе и десятая ячейка. После пятикратного выполнения команды PUSH ACC в указателе стека находится число 12. Именно оно и определяет содержимое аккумулятора после выполнения последней команды.

Ответ: 1+2+2+(2*248+2*4*256)+2*5+2*5+1+1=2571 мкс, A=12=0CH

Задача 27. Оценить время выполнения команд после сброса при частоте кварца 12 МГц.

```
        ORG     0
        MOV     A,#22H      ;1   (A)=34
        PUSH    ACC         ;2
        MOV     B,SP        ;2   (B)=8
        DIV     AB          ;4   (A)=4  (B)=2
        ACALL   1           ;2+2
        MOVC    @A+PC       ;2      (A)=253
        DIV     AB          ;253/2   (A)=126  (B)=1
        MUL     AB          ;(A)=126  (B)=0
M1:     DJNZ    ACC,$       ;2*126+2*255*256
        DJNZ    B,M1        ;2*256
```

Решение. Число 22H находится в первой ячейке и соответствует машинному коду команды RET. Она выполняется при вызове подпрограммы ACALL 1. При выполнении команды MOVC @A+PC в аккумулятор записывается третий байт команды DJNZ ACC,$ (относительное смещение от адреса первого байта следующей команды до метки M1 (-3)=253). Основное время выполнения программы определяют две последние команды. Команда DJNZ B,M1 выполняется 256 раз, так как в регистре B находится 0 при первом ее выполнении. Команда DJNZ ACC,$ первый раз выполнится 126 раз, а затем еще в цикле 255 раз по 256 раз.

Ответ: 1+2+2+4+4+2+4+4+252+130560+512=131347 мкс.

Задача 28. Оценить содержимое аккумулятора и время выполнения команд при частоте кварца 12 МГц

```
                ORG     0
                SJMP    M1              ;2
DELAY:          DJNZ    ACC,$           ;2*192
                MOV     A,SP            ;1          (A)=10
                RET                     ;2
   M1:          MOVC    A,@A+PC         ;2      (A)=0C0H=192
                PUSH    ACC             ;2
                ACALL   DELAY           ;2
                POP     B               ;2      (B)=192
                MUL     AB              ;4      10*192=1920
                                        ;   (B)=7    (A)=128=80H
```

Решение. Время выполнения команд в машинных циклах приведено в поле комментария. После сброса аккумулятор обнулен. Поэтому, при выполнении команды MOVC A,@A+PC в аккумулятор попадает машинный код C0 следующей команды, адрес которой находится в программном счетчике. После выполнения команды PUSH ACC в указателе стека 8, а после вызова подпрограммы DELAY содержимое указателя стека становится равным 10. Именно оно пересылается в аккумулятор при выполнении подпрограммы. В регистр B из стека извлекается число C0H=192. При выполнении команды умножения старший байт определяет содержимое регистра B, а в аккумулятор записывается младший байт произведения. Все команды программы выполняются однократно, кроме второй команды, которая в цикле выполняется 192 раза.

Ответ: 17+2*192=401 мкс, (A)=128=80H

Задача 29. Определить время выполнения программы после сброса МК51 при частоте кварца 12 МГц.

```
                ORG     0
C0 04           PUSH    4               ;2
D5 81 FC        DJNZ    SP,1            ;2
D5 E0 FA        DJNZ    ACC,2           ;2
04              INC     A               ;1
```

Решение. При выполнении первой команды содержимое указателя стека увеличивается на единицу. Поэтому вторая команда выполнится 8 раз. При этом 7 раз реализуется переход к первой ячейке и выполнение команды INC A (КОП=04). По завершению цикла SP=0.

Следующая команда выполняется при A=7. Содержимое аккумулятора декрементируется и реализуется переход ко второй команде (начинающейся со второй ячейки), которая выполнится 256 раз, причем 255 раз в этом цикле выполнится и команда INC A, что приведет к дополнительному декременту аккумулятора (A=5).

Через следующий такой цикл A=3, а еще через цикл A=1, после чего по одному разу выполнятся последние две команды.

Ответ: 2+8*2+7*1+3(2+256*2+255*1)+2+1=2335 мкс

Задача 30. Оценить время выполнения команд после сброса МК51 при частоте кварца 12 МГц.

```
        ORG     0
        MOV     B,SP        ;2
M1:     MOV     A,SP        ;1
        PUSH    ACC         ;2
        DJNZ    B,M1        ;2
        ADD     A,10        ;1
        DJNZ    ACC,$       ;2
```

Решение. Цикл с возвратом на метку M1 реализуется 7 раз. Ячейки резидентной памяти данных, начиная с восьмой, заполняются числами натурального ряда, начиная с 7, т.к. при выполнении команды PUSH ACC содержимое указателя стека инкрементируется. По завершении цикла в аккумуляторе будет число 13, а в десятой ячейке РПД число 9. Поэтому последняя команда выполнится 21 раз.

Ответ: 2+(1+2+2)7+1+2(13+9)=82 мкс

6 ПРИМЕРЫ ПРОГРАММ ОБРАБОТКИ ДАННЫХ

Пример 1. Заполнить массив 1100H-110FH внешнего ОЗУ данных константой со входов порта P1. Транслировать программу, начиная с адреса 100H.

```
1  0100                ORG     100H
2  0100  901100        MOV     DPTR,#1100H
3  0103  7910          MOV     R1,#16
4  0105  E590          MOV     A,P1
5  0107  F0      M1:   MOVX    @DPTR,A
6  0108  A3            INC     DPTR
7  0109  D9FC          DJNZ    R1,M1
8  010B                END
```

Для обращения к ВПД используется регистр указатель данных DPTR. Счетчик числа элементов массива выполнен на регистре R1. В четвертой строке программы второй байт берется равным прямому адресу порта P1. В седьмой строке второй байт равен относительному смещению от адреса следующей команды до адреса, соответствующего метке M1 (дополнительный код числа минус 4).

Пример 2. Произведение П цифр двухразрядного десятичного числа, находящегося в аккумуляторе в двоично-десятичном коде, возвратить в аккумулятор также в двоично-десятичном коде. Транслировать программу, начиная с нулевой ячейки.

```
1  0000  75F010    MOV     B,#10H      ; Распаковка цифр числа
2  0003  84        DIV     AB          ;  в регистры   A и  B
3  0004  A4        MUL     AB          ; Двоичный код  П  в  A
4  0005  75F00A    MOV     B,#10       ; П делится на 10. A содержит
5  0008  84        DIV     AB          ; цифру десятков,  B — остаток
6  0009  C4        SWAP    A           ; Цифры произведения в
7  000A  45F0      ORL     A,B         ; упакованном формате
8  000C            END
```

Сначала исходное число делится на 16 (процессор при выполнении команды деления считает, что содержимое аккумуля-

тора соответствует двоичному числу). Старшая цифра числа попадает в A, младшая — в B. Затем в аккумуляторе формируется двоичный код их произведения. Далее делением на 10 реализуется преобразование произведения в двоично-десятичный формат. Второй байт команды в седьмой строке соответствует прямому адресу регистра B.

Пример 3. Скопировать массив РПД 20H-2FH на новое место, начиная с ячейки 40H.

```
        MOV     R0,#20H     ; Начальный адрес первого массива
        MOV     R1,#40H     ; Начальный адрес второго массива
        MOV     R2,#16      ; Число элементов массива
M1:     MOV     A,@R0       ; Пересылка очередного элемента
        MOV     @R1,A       ; массива
        INC     R0          ; Организация цикла
        INC     R1          ; копирования
        DJNZ    R2,M1       ; элементов массива
```

Для обработки элементов массива в цикле всегда удобно использовать косвенную адресацию, которая в данном примере реализуется с помощью регистров R0 и R1. Другие регистры общего назначения для этой цели использовать нельзя. Директива END в данном и последующих примерах опущена.

Пример 4. Наибольшее число массива 8-разрядных чисел без знака в РПД (20H-2FH) поместить в ячейку 30H.

```
MAX     EQU     30H         ; Директива ассемблера
        MOV     R0,#20H     ; Указатель памяти
        MOV     R1,#16      ; Счетчик числа элементов
        MOV     MAX,#0      ; Обнуление ячейки результата
M1:     MOV     A,@R0       ; Сравнение очередного элемента
        CJNE    A,MAX,$+3   ; массива с ячейкой результата
        JC      M2          ; Переход, если меньше или равно
        MOV     MAX,A       ; Замена, если больше
M2:     INC     R0          ; Организация цикла
        DJNZ    R1,M1       ; просмотра элементов массива
```

С помощью директивы EQU ячейке резидентной памяти данных с адресом 30H присвоено символическое имя MAX, которое неоднократно используется в тексте программы, улучшая ее «читаемость». Присвоение уникальных имен всем переменным, используемым при выполнении задачи — прием, широко используемый в практике программирования на языке ассемблера.

В данном примере результатом выполнения команды сравнения является установка или сброс флага переноса C. Команда тестирования этого флага выполняется не зависимо от того, равно содержимое аккумулятора содержимому ячейки MAX или нет.

Пример 5. Сравнить содержимое аккумулятора с константой 100 и выполнить следующие действия:

если A=100, то перейти на метку M1;

если A<100, то перейти на метку M2;

если A>100, то перейти на метку M3.

```
        CJNE    A,#100,$+6
        LJMP    M1          ; Переход, если A=100
        JC      M2          ; Переход, если A<100
M3:     ……..                ; Выполнение условия A>100
```

Если содержимое аккумулятора не равно 100, то дополнительно тестируется флаг переноса C. Он устанавливается в единицу при выполнении условия A<100 и в ноль при A>100.

Пример 6. Преобразовать двоичное число без знака, находящееся в аккумуляторе, в двоично-десятичное и поместить его в DPTR.

```
MOV     B,#100      ; Делим на 100 для определения
DIV     AB          ; числа сотен
MOV     DPH,A       ;
MOV     A,#10       ; Делим на 10 для определения
XCH     A,B         ; числа десятков
DIV     AB          ;
SWAP    A           ; Обмен полубайтов аккумулятора
ADD     A,B         ; Добавляем единицы
MOV     DPL,A       ;
```

Пример 7. Показать структуру построения программы, использующей аппаратное прерывание по фронту INT0.

```
        ORG     0       ; Начало программы
        SJMP    MAIN    ; Переход к основной программе
        ORG     0003H   ; Вектор прерывания
        AJMP    SUBR    ; Переход к п/п обслуживания
MAIN:   MOV     SP,#100 ; Настройка указателя стека
        SETB    EA      ; Сброс блокировки прерываний
        SETB    EX0     ; Разрешение прерывания INT0
        SETB    IT0     ; Бит прерывания по фронту
                ....... ; Текст основной программы
        ORG     800H    ; Начало п/п прерывания
SUBR:   PUSH    PSW     ; Сохранение в стеке
        PUSH    ACC     ; содержимого
        PUSH    B       ; регистров
        PUSH    DPL     ;
        PUSH    DPH     ;
        SETB    RS0     ; Выбор первого банка РОН
        CLR     RS1
                ....... ; Текст подпрограммы
        POP     DPH     ; Восстановление из стека
        POP     DPL     ; содержимого
        POP     B       ; регистров
        POP     ACC     ;
        POP     PSW     ;
        RETI            ; Возврат из п/п прерывания
```

Предполагается регистры первого банка РОН использовать только в подпрограмме обслуживания аппаратного прерывания. Для этого необходимо модифицировать содержимое указателя стека, так как после сброса он настроен на область РПД, занимаемую банком РОН1.

Прерывание происходит всегда неожиданно для основной программы, поэтому при его обслуживании необходимо сохранить содержимое PSW и всех регистров, используемых подпрограммой. В данном примере предполагается использовать в подпрограмме регистры A, B и DPTR.

Текст подпрограммы обслуживания прерываний можно располагать в любом удобном месте программы. Так как при наличии запроса на прерывание управление передается ячейке с адре-

сом 0003H, в этой ячейке записывается команда безусловного перехода к выбранному адресу подпрограммы обслуживания. Если после выполнения команды RETI на входе INT0 сохраняется 0, подпрограмма обслуживания не будет выполняться повторно, так как установлен бит прерывания по фронту IT0.

Пример 8. На линии P1.0 — P1.3 поступают сигналы *X*, *Y*, *Z* и *V* от датчиков. Выдать на линию P1.4 этого порта сигнал в соответствии с логическим выражением $F=X(Y+Z)+\overline{V}$.

```
 1  0090              X     EQU   P1.0
 2  0091              Y     EQU   P1.1
 3  0092              Z     EQU   P1.2
 4  0093              V     EQU   P1.3
 5  0094              F     EQU   P1.4
 6  0000  A291              MOV   C,Y
 7  0002  7292              ORL   C,Z
 8  0004  8290              ANL   C,X
 9  0006  A093              ORL   C,/V
10  0008  9294              MOV   F,C
11  000A                    END
```

Контроллер МК51 имеет битовый процессор и позволяет проводить логические операции и операции тестирования с отдельными битами. Прямую адресацию имеют 128 флагов пользователя в РПД и биты 11 регистров специальных функций. В РПД можно организовать карту опроса 128 датчиков и эффективно обрабатывать эту информацию с использованием команд битового процессора.

В контроллерах, не имеющих битового процессора (К580, МК48), каждая команда логической обработки бита требует загрузки байта в аккумулятор, выполнения команд логической обработки байтов, маскирования и команд условных переходов. Поэтому реализация булевых функций микроконтроллером МК51 осуществляется значительно проще и быстрее.

На языке ассемблера битовый операнд можно записывать, используя символические имена бита или регистра, либо прямые

адреса бита или регистра. Вот примеры записи команды выбора первого банка РОН:

```
SETB   RS0        ; Используется символическое имя бита
SETB   PSW.3      ; Используется символическое имя регистра
SETB   0D0H.3     ; Используется прямой адрес регистра
SETB   0D3H       ; Используется прямой адрес бита
```

Независимо от формы записи команды второй байт при трансляции соответствует прямому адресу бита (D3).

Логические операции обработки битов реализуются с участием триггера переноса C, который для битовых операндов выполняет такую же роль, как аккумулятор в командах арифметических и логических операций с байтами.

Пример 9. Организовать задержку длительностью 50 мс на микроконтроллере К1830ВЕ51 с использованием таймера, прерываний и режима холостого хода.

```
 1  000B                    ORG     000BH    ; Вектор прерывания
 2  000B  C28C              CLR     TR0      ; Останов T/C0
 3  000D  32                RETI             ; Возврат из п/п
 4  0100                    ORG     100H     ; Начало программы
 5  0100  D2AF     MAIN:    SETB    EA       ; Снятие блокировки
 6  0102  758901            MOV     TMOD,#01H    ; Режим 1 T/C0
 7  0105  758AB0            MOV     TL0,#LOW(NOT(50000)+1)
 8  0108  758C3C            MOV     TH0,#HIGH(NOT(50000)+1)
 9  010B   D28C             SETB    TR0        ; Старт T/C0
10  010D  D2A9              SETB    IE.1   ; Разрешение прерываний
11  010F  758701            MOV     PCON,#01H        ; Режим XX
12  0112          NEXT:     .....          ; Продолжение программы
```

При использовании таймера в режиме 1 (кварц на 12 МГц) можно получать задержки до 65536 мкс. Включение и выключение таймера осуществляется установкой и сбросом бита TR0 (TCON.4). Чтобы переполнение таймера произошло через 50 мс, в его регистры необходимо загрузить дополнительный код числа 50000. Формирование дополнительного кода, загрузку младшего байта в TL0, а старшего в TH0, выполняет ассемблер.

Микроконтроллеры серии К1830, выполненные по КМОП-технологии, можно перевести в режим холостого хода установкой нулевого бита регистра PCON (IDL). В этом режиме блокируются функциональные узлы центрального процессора, что уменьшает энергопотребление до 4 мА. Сохраняется содержимое SP, PC, PSW, A и других регистров и РПД. Активизация любого разрешенного прерывания (а также аппаратный сброс микроконтроллера) заканчивает режим ХХ. После выполнения команды RETI будет исполнена команда, которая следует за командой, переведшей МК в режим холостого хода.

Установкой первого бита регистра PCON (PD) можно перевести МК в режим микропотребления (напряжение питания может быть уменьшено до 2 В, потребляемый ток — 50 мкА). В этом режиме прекращается работа всех узлов микроконтроллера, но сохраняется содержимое ОЗУ. Вывести МК из режима микропотребления можно только аппаратным сбросом (RST=1).

У микроконтроллеров серии К1816 используется только старший бит регистра PCON (SMOD). Установка этого бита удваивает скорость передачи при работе последовательного порта.

Пример 10. Подсчитать число импульсов, поступающих на вход T1 (P3.5) за заданный промежуток времени (10 мс). Текст программы разместить с адреса 1000H. Результат сформировать в DPTR.

```
 1  D8F0            TIME    EQU   NOT(10000)+1
 2  1000                    ORG   1000H
 3  1000  758951            MOV   TMOD,#01010001B
 4  1003  E4                CLR   A
 5  1004  F58D              MOV   TH1,A
 6  1006  F58B              MOV   TL1,A
 7  1008  758CD8            MOV   TH0,#HIGH(TIME)
 8  100B  758AF0            MOV   TL0,#LOW(TIME)
 9  100E  438850            ORL   TCON,#50H
10  1011  108D02    M1:     JBC   TF0,M2
11  1014  80FB              SJMP  M1
12  1016  858D83    M2:     MOV   DPH,TH1
13  1019  858B82            MOV   DPL,TL1
14  101C                    END
```

Управляющее слово в третьей строке программы настраивает T/C0 на работу в режиме 16-битового таймера, T/C1 — в режиме 16-битового счетчика событий. В регистры таймера перед пуском загружается дополнительный код числа 10000, регистры счетчика событий обнуляются. Команда в девятой строке одновременно осуществляет старт T/C0 и T/C1. Счет импульсов происходит до переполнения T/C0, после чего содержимое T/C1 переписывается в DPTR. Команда JBC сбрасывает флаг TF0.

Пример 11. Спроектировать устройство для измерения длительности одиночного импульса, поступающего на вход INT0 (P3.2) микроконтроллера К1830ВЕ31 (рис. 6.1).

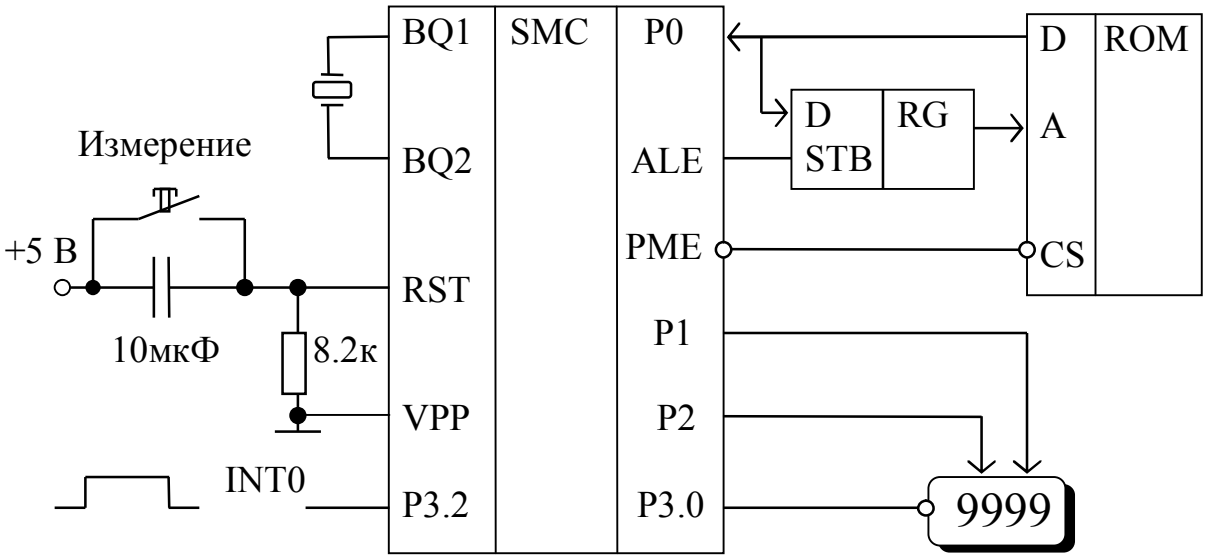


Рис. 6.1 — Функциональная схема измерителя длительности одиночного импульса

Информация о длительности импульса (число до 9999 мкс) выводится на линии портов P2 (старший байт) и P1 (младший байт) в двоично-десятичном коде. Сигнал управления работой внешнего индикатора сформирован на линии P3.0. По сигналу ALE адрес очередного байта команд фиксируется с порта P0 в регистре RG (микросхема К580ИР82), а по сигналу PME этот байт поступает в микроконтроллер из внешнего ПЗУ (микросхема К556РТ5) через тот же порт P0. Запуск устройства на измерение импульса производится нажатием кнопки «Измерение», вызы-

вающим сброс микроконтроллера и повторный старт программы, начиная с нулевого адреса. При этом автоматически происходит обнуление регистров таймера T/C0.

К микроконтроллеру, используемому для решения поставленной задачи, должен подключаться кварцевый резонатор на 12 МГц, при котором на вход таймера при положительном уровне на входе INT0 поступают импульсы с частотой 1 МГц. По окончании измеряемого импульса его длительность (она не должна превышать 10 мс) в микросекундах фиксируется в двоичном коде в регистрах TH0 и TL0.

```
        ; Прикладная программа измерителя
        ; длительности одиночного импульса

        CLR     P3.0            ; Гашение индикаторов
        MOV     TMOD,#1001B     ; Настройка T/C0 на режим 1
        MOV     P1,#0           ; Сброс суммирующего
        MOV     P2,#0           ; двоично-десятичного счетчика
        SETB    TR0             ; Разрешение работы таймера
        JNB     INT0,$          ; Ожидание начала импульса
        JB      INT0,$          ; Счет в T/C0 до окончания импульса
        CLR     TR0             ; Блокировка работы таймера
M1:     MOV     A,P1            ; Инкремент двоично-десятичного
        ADD     A,#1            ; счетчика
        DA      A
        MOV     P1,A
        CLR     A
        ADDC    A,P2
        DA      A
        MOV     P2,A
        MOV     A,TL0           ; Декремент двоичного счетчика
        CLR     C
        SUBB    A,#1
        MOV     TL0,A
        MOV     A,TH0
        SUBB    A,#0
        MOV     TH0,A
        ORL     A,TL0           ; Тестирование на нуль
        JNZ     M1              ; Продолжить преобразование
        SETB    P3.0            ; Зажечь индикаторы
        SJMP    $               ; Останов
```

Преобразование двухбайтового двоичного в двухбайтовое двоично-десятичное число осуществляется методом двух счетчиков. Суммирующий двоично-десятичный счетчик реализован в портах P1, P2, вычитающий двоичный — на регистрах таймера TL0, TH0. Процесс преобразования заканчивается при обнулении двоичного счетчика.

Пример 12. Построить на микроконтроллере К1830ВЕ51 электронные часы с индикацией часов, минут и секунд реального времени (рис. 6.2). Прикладная программа приведена в программе выполнения лабораторной работы №2. Информация с двоично-десятичных счетчиков часов, минут и секунд, реализованных в портах P0, P1 и P2, поступает на цифровые семисегментные индикаторы через преобразователи кода.

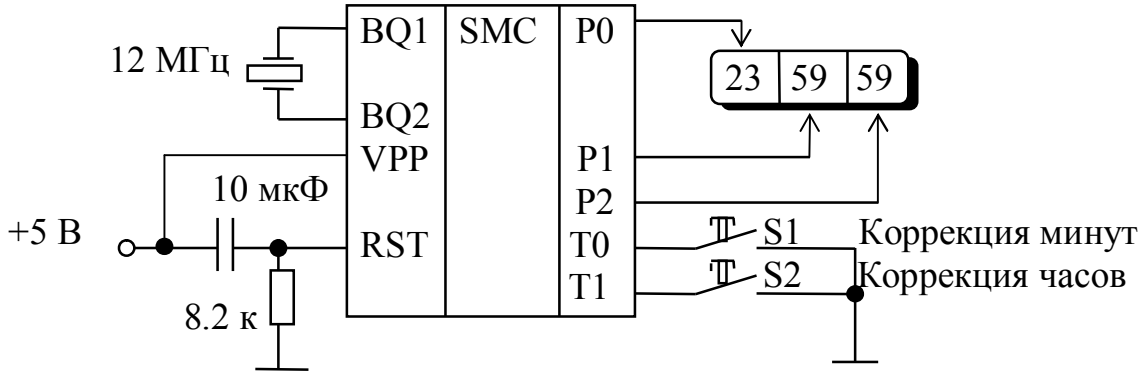


Рис. 6.2 — Функциональная схема электронных часов

Пример 13. Сформировать на выходе WR импульс логического нуля длительностью 50 мкс.

Ниже приводится фрагмент программы, реализующий эту функцию, с указанием времени выполнения отдельных команд (при частоте кварца 12 МГц один машинный цикл составляет 1 мкс):

```
CLR     WR      ; 1 машинный цикл
MOV     R7,#24  ; 1 машинный цикл
DJNZ    R7,$    ; 2 машинных цикла
SETB    WR      ; 1 машинный цикл
```

Время между сбросом и установкой бита WR (P3.6) составит примерно 50 мкс, так как команда DJNZ выполнится 24 раза.

Пример 14. Разработать программу преобразования кода 10 элементов массива, находящегося в резидентной памяти микроконтроллера К1816ВЕ51, из двоично-десятичного в двоичный (байтовая информация).

Считаем, что массив занимает ячейки памяти данных на кристалле с 50 по 59. Используем команду умножения байтов. Подробный комментарий помещен в текст программы.

```
        MOV     R0,#50      ; Начальный адрес массива
        MOV     R7,#10      ; Число элементов массива
M1:     MOV     A,@R0       ; Выделение числа
        SWAP    A           ;  десятков
        ANL     A,#0FH
        MOV     B,#10       ; 10 в расширитель аккумулятора
        MUL     AB
        MOV     B,A
        MOV     A,@R0       ; Выделение единиц
        ANL     A,#0FH
        ADD     A,B         ; Формирование двоичного числа
        MOV     @R0,A       ; Замена элемента массива
        INC     R0          ; Наращивание адреса
        DJNZ    R7,M1       ; Цикл обработки массива
```

Время между сбросом и установкой бита WR (P3.6) составит примерно 50 мкс, так как команда DJNZ выполнится 24 раза.

Пример 15. Разрешить внешние прерывания по фронту сигналов на входах INT0 и INT1 и прерывание по переполнению таймера/счетчика T/C1.

Прерыванию по T/C1 присвоить высший приоритет. Стек организовать начиная с сотой ячейки РПД.

Инициализацию системы прерываний выполним установкой в 1 соответствующих битов регистров IE, IP, TCON. После системного сброса эти регистры обнулены.

```
MOV     IE,#10001101B   ; Установка битов EA, ET1, EX1, EX0
                        ; регистра масок прерываний
SETB    PT1             ; Высший приоритет T/C1 установкой
                        ; бита IP.3 регистра приоритетов
SETB    IT0             ; Прерывания по фронту INT0
SETB    IT1             ; Прерывания по фронту INT1
MOV     SP,#99          ; Модификация указателя стека
```

Установка бита EA снимает общую блокировку прерываний, которая действует после системного сброса. После выполнения первой команды разрешены три прерывания (в порядке убывания приоритетов): внешнее аппаратное по входу INT0 с вектором 03H, внешнее аппаратное по входу INT1 с вектором 13H и прерывание по таймеру/счетчику T/C1 с вектором 1BH. После выполнения второй команды прерыванию от T/C1 присваивается высокий уровень приоритета. Подпрограмма обслуживания этого прерывания не может быть прервана другим источником прерываний. В то же время установка флага TF1 вызовет переход к вектору 1BH даже при выполнении подпрограмм обслуживания внешних прерываний. Все подпрограммы прерываний должны заканчиваться командой RETI (при обслуживании прерываний действует блокировка прерываний такого же уровня, которую команда RET не снимает). После системного сброса в указатель стека заносится число 7H. Последняя команда модифицирует его так, что при вызове подпрограммы адрес возврата запишется в сотую (младший байт) и сто первую ячейки РПД (старший байт).

Пример 16. Получить на линейке светодиодов, подключенных к линиям порта P1, световой эффект бегущего огонька.

Нагрузочной способности порта P1 недостаточно для непосредственного включения светодиодов (выходной ток высокого уровня сигналов — 80 мкА, выходной ток низкого уровня сигналов — 1,6 мА, в то время как нормальное свечение светодиода обеспечивается при токе порядка 5–10 мА). В качестве усилителя тока можно использовать преобразователи уровня или логические элементы НЕ (см. рис. 8.2).

Алгоритм получения эффекта «бегущая единица» состоит в организации цикла последовательных операций: вывод 1 в одну из линий порта, организация задержки, смена адреса активной линии порта. Смену адреса проще всего получить путем сдвига кодовой единицы в аккумуляторе. Задержка должна составлять десятые доли секунды. При частоте переключений больше 24 Гц глазу человека будет казаться, что все светодиоды горят непрерывно. С учетом этих замечаний построена приводимая ниже программа.

```
        MOV     A,#1            ; 1
M1:     MOV     P1,A            ; 2
DELAY:  DJNZ    R0,$            ; 2
        DJNZ    R1,DELAY        ; 2
        RL      A               ; 1
        SJMP    M1              ; 2
```

В поле примечания приведено время выполнения команд программы в машинных циклах (при частоте кварцевого резонатора 12 МГц один машинный цикл равен 1 мкс). После окончания временной задержки регистры R0 и R1 обнулены. Таким образом, каждый раз при реализации временной задержки команда DJNZ R1,DELAY выполняется 256 раз, а команда DJNZ R0,$ — 256^2 раз.

Суммарное время задержки составляет $5+2(256+256^2) = 131589$ мкс.

Пример 17. Подсчитать количество символов в первом предложении текста, размещенного в РПД начиная с ячейки 20H (включая пробелы). Результат сформировать в регистре R7 в двоично–десятичном коде.

Решение. Ниже приведен текст программы на языке ассемблера с подробным комментарием:

```
        MOV     R0,#20H         ; начало массива
        CLR     A               ; очистка счетчика
M1:     CJNE    @R0,#'.',M2     ; сравнение с кодом точки 2EH
        SJMP    EXIT            ; закончить обработку массива
M2:     INC     R0              ; наращивание указателя памяти
```

```
        ADD     A,#1            ; двоично–десятичный
        DA      A               ; счетчик в аккумуляторе
        SJMP    M1              ; организация цикла
EXIT:   MOV     R7,A            ; результат в R7
```

Используется косвенная адресация элементов массива. Двоично-десятичный счетчик числа символов организован в аккумуляторе с использованием команды десятичной коррекции.

После ассемблирования программы сформирован файл листинга прикладной программы massiv.lst, приведенный ниже.

```
##############################################################
#                                                            #
# Micro Series 8051 Assembler V1.80/MD2   11/Oct/07  11:21:50  #
#                                                            #
#      Source   =  massiv.asm                                #
#      List     =  massiv.lst                                #
#      Object   =  massiv.r03                                #
#      Options  =                                            #
#                                                            #
#                                (c) Copyright IAR Systems 1985      #
##############################################################


   1  0000 7820              mov       r0,#20H
   2  0002 E4                clr       a
   3  0003 B62E02    m1:     cjne      @r0,#'.',m2
   4  0006 8006              sjmp      exit
   5  0008 08        m2:     inc       r0
   6  0009 2401              add       a,#1
   7  000B D4                da        a
   8  000C 80F5              sjmp       m1
   9  000E FF        exit:   mov       r7,a
  10  000F                   end

   Errors:  None          ##########
   Bytes:   15            #  massiv  #
   CRC:     F3A2          ##########
```

7 ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ МК51

Программный доступ к регистрам приемника и передатчика последовательного интерфейса МК51 осуществляется обращением к регистру **SBUF**. При записи в SBUF байт загружается в регистр передатчика, при чтении SBUF байт читается из регистра приемника.

Управление работой последовательного порта осуществляется с помощью регистра **SCON**, все разряды которого программно доступны по записи и чтению (рис. 7.1):

SCON

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SM1	
0	0	Режим 0
0	1	Режим 1
1	0	Режим 2
1	1	Режим 3

Рис. 7.1 — Управляющее слово приемопередатчика

Порт может работать в следующих четырех режимах.

Режим 0. Информация передается (младшими битами вперед) и принимается через вход приемника RxD (P3.0). Через выход передатчика TxD (P3.1) выдаются импульсы синхронизации, стробирующие каждый передаваемый или принимаемый бит информации. Формат посылки — 8 бит. Частота приема и передачи — $f_{BQ}/12$. Передача начинается любой командой, которая использует регистр SBUF в качестве регистра назначения, прием — при сбросе флага RI (если REN=1).

Режим 1. Информация передается через выход передатчика, а принимается через вход приемника. Формат посылки — 10 бит: старт-бит (ноль), восемь бит данных и стоп-бит (единица). Частота приема и передачи задается таймером-счетчиком T/C1.

Режим 2. Формат посылки — 11 бит (рис. 7.2): старт-бит, восемь бит данных, программируемый девятый бит и стоп-бит. Передаваемый девятый бит данных принимает значение бита **TB8**. Бит TB8 регистра SCON может быть программно установлен в 1 или 0, или в него можно поместить значение бита P из регистра

PSW для повышения достоверности принимаемой информации (контроль по паритету). При приеме девятый бит принятой посылки поступает в бит **RB8** регистра SCON. Частота приема и передачи задается программно и может быть равна $f_{BQ}/32$ (SMOD=1) или $f_{BQ}/64$ (SMOD=0). Бит **SMOD** регистра PCON можно установить в 1 командой MOV PCON,#80H.

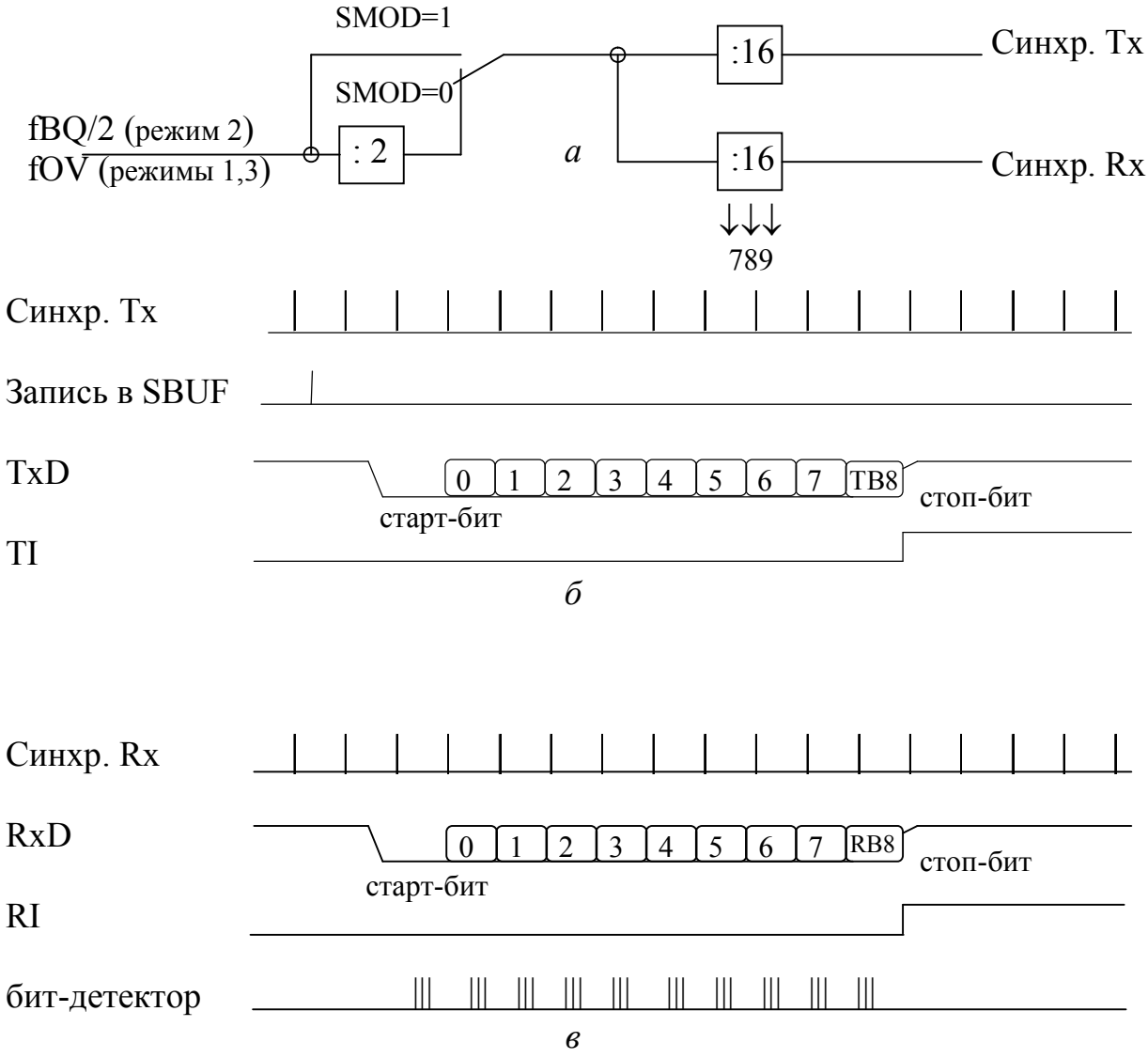


Рис. 7.2 — Временные диаграммы, иллюстрирующие работу последовательного порта в режимах 2 и 3:
а — схема синхронизации; *б* — передача; *в* — прием

Режим 3. Полностью идентичен режиму 2, только частота приема и передачи задается (как и в режиме 1) таймером T/C1 и равна $(2^{SMOD}/32)f_{OV}$, где f_{OV} — частота переполнения T/C1. Обычно

для синхронизации последовательного порта T/C1 включается в режим перезагрузки (режим 2). В этом случае $f_{OV}=f_{BQ}/\{12\ [256 - (TH1)]\}$. Прерывания от T/C1 запрещаются.

Передача инициируется любой командой, использующей регистр SBUF в качестве регистра назначения. На выход TxD выдается девять бит данных: D0-D7 и TB8. По окончании передачи устанавливается флаг прерывания передатчика TI. Прием начинается при обнаружении перехода сигнала на входе RxD из 1 в 0. В фазах 7, 8 и 9 специальное устройство МК бит-детектор считывает со входа RxD значения принимаемого бита и по мажоритарному принципу выбирает одно из них. По окончании приема устанавливается флаг прерывания приемника RI.

Настройка таймера 1 для управления частотой работы порта

Частота приема/передачи	Частота резонатора, МГц	SMOD	C/T	Режим работы T/C1	Перезагружаемое число
Режим 0, макс: 1 МГц	12	X	X	X	X
Режим 2, макс: 375 кГц	12	1	X	X	X
Режимы 1,3: 62,5 кГц	12	1	0	2	0FFH
19,2 кГц	11,059	1	0	2	0FDH
9,6 кГц	11,059	0	0	2	0FDH
4,8 кГц	11,059	0	0	2	0FAH
2,4 кГц	11,059	0	0	2	0F4H
1,2 кГц	11,059	0	0	2	0E8H
137,5 Гц	11,059	0	0	2	1DH
110 Гц	6	0	0	2	72H
110 Гц	12	0	0	1	0FEEBH

Назначение остальных битов регистра SCON:

REN — разрешение приема последовательных данных;

TI — флаг прерывания передатчика. Устанавливается аппаратно в конце выдачи 8-го бита в режиме 0 или в начале стоп-бита в других режимах. Сбрасывается программно;

RI — флаг прерывания приемника. Устанавливается аппаратно в конце времени приема 8-го бита в режиме 0 или через половину интервала стоп-бита в других режимах при SM2=0;

SM2 в режиме 0 должен быть равен 0. При SM2=1 в режиме 1 флаг RI не активизируется, если не принят стоп-бит, а в режимах 2 и 3 — если 9-й принятый бит данных равен 0.

Пример 18. Разработать программу инициализации последовательного порта для работы со скоростью 110 бод (бит/с) при частоте кварца 6 МГц, программы приема и выдачи символа.

```
; Программирование режимов таймера и последовательного порта

CLR     TR1             ; Останов таймера 1
MOV     TH1,#72H        ; Установка скорости пересылки
MOV     SCON,#11011100B ; Режим 9-разрядной посылки
MOV     TMOD,#00100000B ; Режим автозагрузки T/C1
SETB    TR1             ; Запуск таймера 1
; Прием символа из внешнего устройства в аккумулятор

JNB     RI,$    ; Ожидание завершения приема
MOV     A,SBUF  ; Чтение символа в аккумулятор
CLR     RI      ; Очистка флага приема

; Передача символа  из аккумулятора внешнему устройству

JNB     TI,$    ; Ожидание окончания передачи
CLR     TI      ; Очистка флага передачи
MOV     SBUF,A  ; Выдача символа
```

Пример 19. Организовать асинхронный программный обмен данными в последовательном формате (строки символов) между двумя микроконтроллерами со скоростью 2400 бод (бит в секунду). Каждый символ передается последовательностью семи информационных бит и бита контроля четности. Разработать подпрограммы передачи и приема символа. Режим прерываний не использовать.

Последовательный порт настраиваем на режим 1 (SM0=0, SM1=1) с возможностью принимать все сообщения (SM2=0, REN=1). Флаг TI искусственно установим в единицу, обеспечивая начальную готовность регистра SBUF для вывода. Таким образом, в регистр SCON следует загрузить управляющее слово 01010010B.

Таймер 1 будем использовать в режиме 2 (8 бит с автоперезагрузкой) в качестве генератора скорости передачи данных. Для получения частоты 2400 бод в старший регистр таймера необходимо загрузить число 0F4H (дополнительный код числа минус 12) при частоте кварца 11059 кГц.

Требуемый режим работы таймера задаем записью в регистр TMOD управляющего слова 00100000B. Программный запуск таймера осуществим командой SETB TR1.

Таким образом, инициализацию (выбор необходимой конфигурации) таймера и последовательного порта реализуем с помощью следующей последовательности команд:

```
CLR     TR1
MOV     TMOD,#00100000B
MOV     SCON,#01010010B
MOV     TH1,#0F4H
SETB    TR1
```

Передачу символа реализует подпрограмма TBYTE. Она добавляет разряд четности к находящемуся в аккумуляторе 7-разрядному коду символа (код ASCII) и осуществляет передачу байта при наличии готовности последовательного порта.

```
TBYTE:  MOV     C,P         ; Добавление разряда
        CPL     C           ; четности в старший
        MOV     ACC.7,C     ; бит аккумулятора
        JNB     TI,$        ; Ожидание разрешения
        CLR     TI          ; передачи очередного байта
        MOV     SBUF,A      ; Передача байта
        RET
```

Прием символа реализует подпрограмма RBYTE. Она вводит очередной символ в аккумулятор из последовательного порта и устанавливает перенос в случае ошибки четности.

```
RBYTE:  JNB     RI,$        ; Ожидание готовности
        CLR     RI          ; приемника
        MOV     A,SBUF      ; Прием байта в аккумулятор
        MOV     C,P         ; Установка C=1 в случае
        CPL     C           ; ошибки передачи
        ANL     A,#7FH      ; Выделение символа
        RET
```

8 ОРГАНИЗАЦИЯ ЛИНИЙ ПОРТОВ МК51. ПОДКЛЮЧЕНИЕ ВНЕШНИХ УСТРОЙСТВ

Покажем организацию одной из линий порта P1 (рис. 8.1). При записи информации в порт формируется тактовый сигнал «запись в защелку» и информация записывается в D-триггер, который управляет выходным ключевым каскадом VT1–VT3. VT1 открывается единицей и закрывается нулем. VT3 всегда открыт и работает в режиме стабилизатора тока, фиксируя уровень 1 при закрытых VT1 и VT2. Транзистор VT2 открывается на два такта только при запирании VT1, увеличивая скорость заряда выходной емкости в 100 раз.

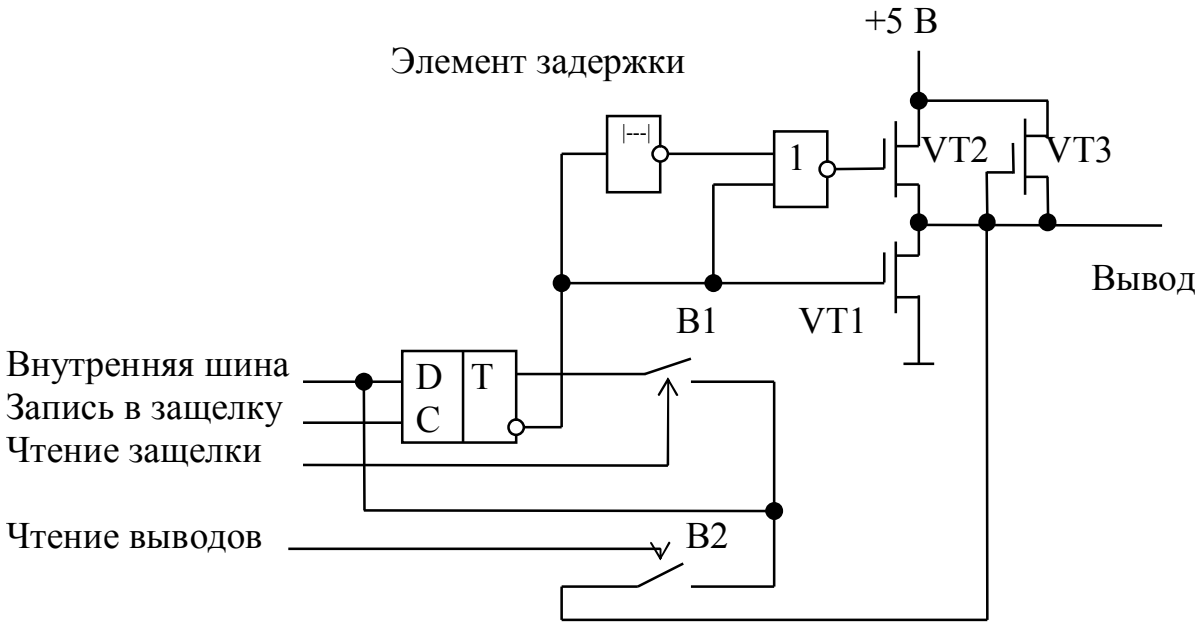


Рис. 8.1 — Организация линии порта P1

Часть команд МК при чтении порта активизируют сигнал «чтение защелки» и через буфер B1 читают содержимое фиксатора. Часть команд формируют сигнал «чтение выводов» и читают информацию на внешних выводах порта через буфер B2. В последнем случае необходимо, чтобы транзистор VT1 был закрыт, т.е. в триггер записана 1.

Когда регистром назначения является порт, реализуется цикл «чтение, модификация, запись» и читается защелка (ANL

P1,A; ORL P2,#d; JBC P1.1,M1). Когда регистром назначения является не сам порт, читается потенциал вывода (MOV A,P1).

Сопряжение линий порта P1 с датчиками и исполнительными элементами цепи электрооборудования автомобиля (аккумулятор на 12 В) показано на рис. 8.2. Выходной ток порта в состоянии логической 1 равен 80 мкА. Согласующие транзисторы должны обладать достаточным усилением по току для работы в ключевом режиме на лампу накаливания или обмотку реле.

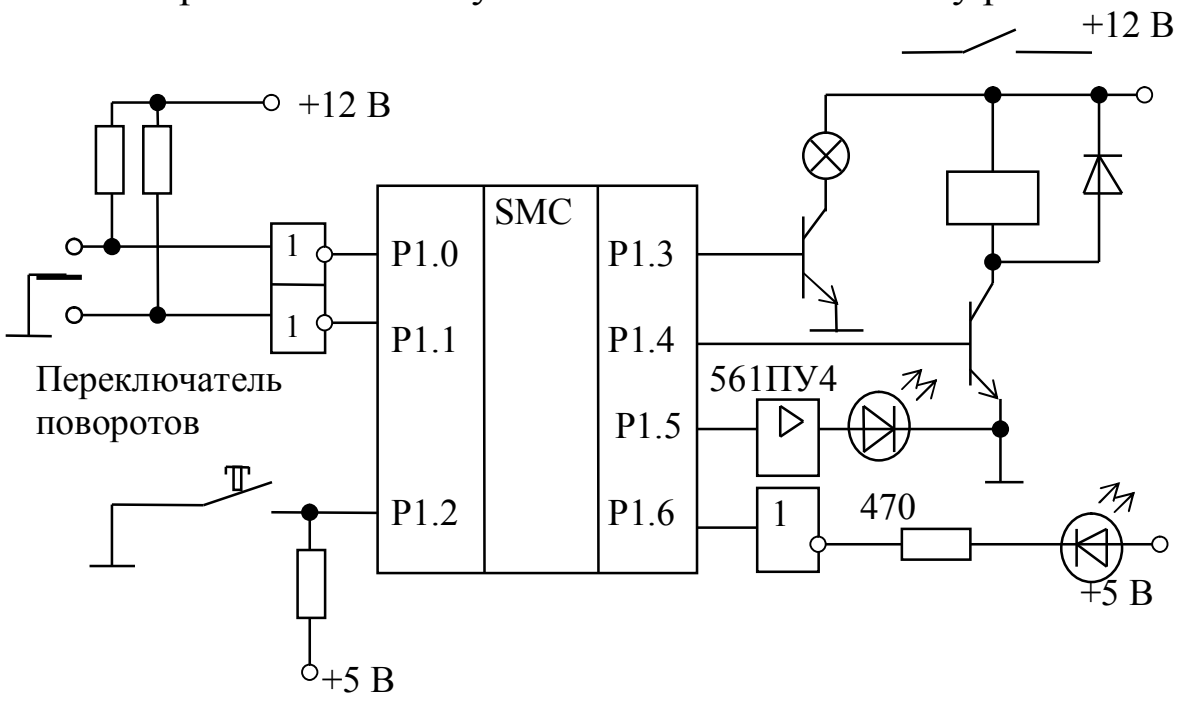


Рис. 8.2 — Подключение внешних устройств

Для увеличения числа каналов ввода информации через порт P1 можно использовать мультиплексоры или шинные формирователи (рис. 8.3), управляя ими программно с помощью линий порта P2.

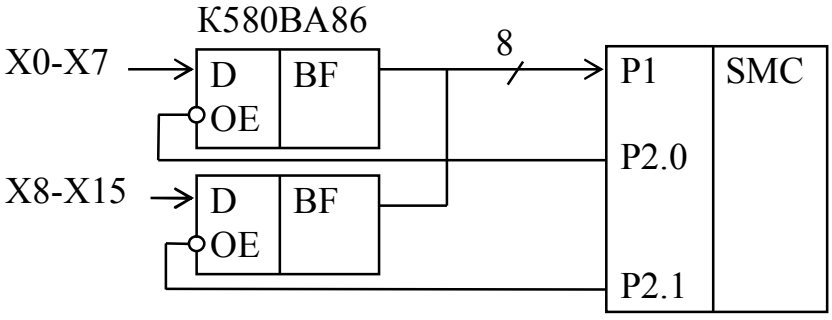


Рис. 8.3 — Мультиплексирование входных данных

Например, на МК51 можно построить контроллер со следующими характеристиками (рис. 8.4):

− 64 датчика битовой переменной;
− часы реального времени;
− 10 выходных сигналов;
− 2 внешних прерывания (с приоритетом).

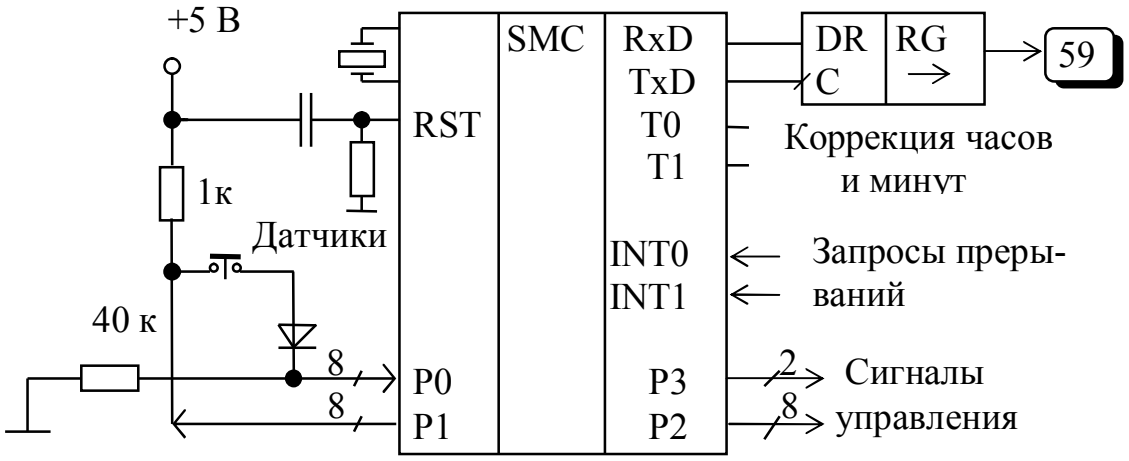


Рис. 8.4 — Вариант построения контроллера на МК51

Передача информации на цифровые индикаторы часов осуществляется записью байта в регистр SBUF последовательного порта, который после сброса работает в режиме сдвигового регистра (режим 0).

Сканирование матрицы датчиков реализуется с помощью линий портов P1 и P0. После завершения цикла сканирования информация с восьми столбцов матрицы датчиков запоминается в 8-байтовом блоке ОЗУ с побитовой адресацией, и создается внутренняя карта текущего состояния всех датчиков, используемая программой. Диод разрешает несколько срабатываний датчиков на одной линии возврата.

Пример 20. Разработать подпрограмму чтения текущего состояния 64 датчиков в блок ОЗУ 20H-27H с сохранением предыдущего состояния в блоке 28H-2FH.

```
SCAN:   MOV     R0,#20H     ; Установка указателей
        MOV     R1,#28H     ; памяти
        MOV     A,#80H      ; Установка старшего бита A
M1:     MOV     P1,A        ; Возбуждение линии сканирования
        RR      A           ; Адрес следующей линии
```

```
MOV     R2,A      ; сканирования в R2
MOV     A,P0      ; Чтение линии возврата
XCH     A,@R0     ; Запись текущего состояния
MOV     @R1,A     ; Сохранение предыдущего
INC     R0        ; Смещение указателей
INC     R1        ; памяти
MOV     A,R2      ; Выбор следующей линии опроса
JNB     ACC.7,M1  ; Цикл считывания 8 столбцов
RET
```

Пример 21. С помощью микросхемы К580ВД79 обеспечить взаимодействие МК51 с цифровой клавиатурой (кнопки S0-S9) и 8-местным дисплеем на светодиодных семисегментных индикаторах (рис. 8.5). Разработать подпрограмму ввода 8-разрядного десятичного числа с клавиатуры в РПД микроконтроллера (ячейки 20H-27H) с дублированием информации на восьмисимвольном дисплее (HG0-HG7).

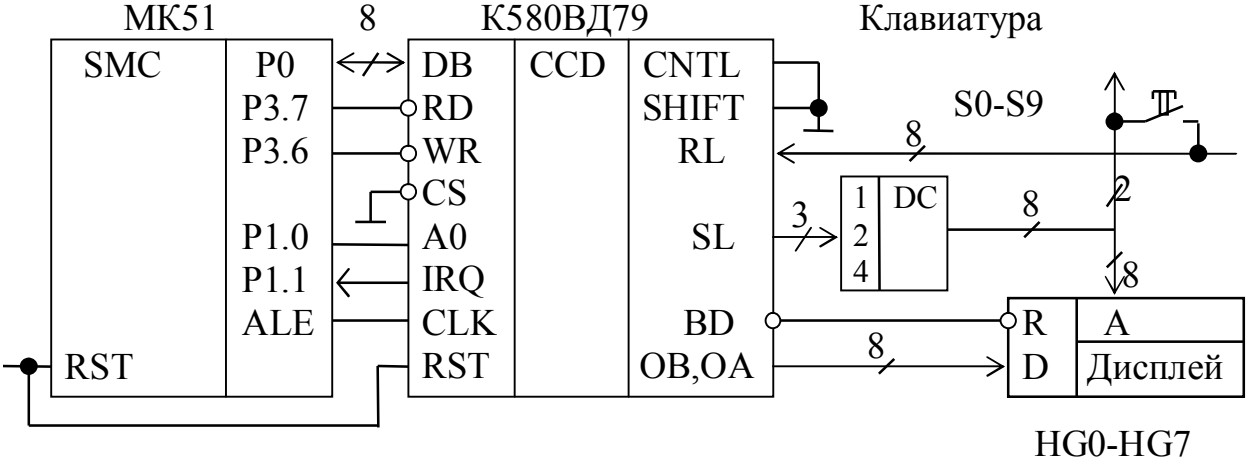


Рис. 8.5 — Подключение к МК51 контроллера клавиатуры и дисплея

Контроллер клавиатуры и дисплея К580ВД79 (ККД) освобождает микроконтроллер от выполнения задач постоянного сканирования клавиатуры и поддержания изображения на цифровом дисплее.

При нажатии клавиши обеспечивается антидребезговый контроль состояния клавиш, код нажатой клавиши вводится в память клавиатуры ККД и возбуждается линия прерывания IRQ, которая может опознаваться в МК51. Информация, выводимая на дисплей, записывается в ОЗУ дисплея ККД (до 16 байт).

С точки зрения программиста ККД представляет собой два порта: регистр данных (A0=0) и регистр управления (A0=1). При A0=1 из МК51 в ККД передаются управляющие слова, а из ККД в МК51 — байт состояния. При A0=0 передается байт данных. Загружая в ККД определенные управляющие слова (УС), можно настроить ККД на работу в требуемом режиме или предписать выполнение некоторой операции. В рассматриваемом примере ККД использует команды работы с внешним ОЗУ (сигналы WR и RD) и закрывает все пространство ВПД.

Линия P1.0 должна быть установлена/сброшена в зависимости от типа обращения (управление/данные). Выход сигнала запроса прерывания IRQ ККД может быть программно опрошен для определения факта нажатия клавиши. На вход CLK подается сигнал ALE (2 МГц при частоте кварца 12 МГц). Для сканирования клавиатуры и дисплея используется дешифратор на восемь выходов. Каждая клавиша клавиатуры расположена таким образом, что ее код соответствует двоичному коду цифры, нанесенной на клавишу.

Код символа, выводимый на дисплей с выходов четырехразрядных регистров OA и OB, подается одновременно на все цифровые индикаторы. Символ, соответствующий этому коду, загорается на том индикаторе, который получает питание по цепи, открытой активным выходом дешифратора. Сигнал бланкирования BD используется для гашения дисплея в момент переключения цифр. Это позволяет избежать наложения символов в соседних позициях дисплея.

Для программирования ККД используются управляющие слова:

УС0 | 000DDKKS | — инициализации клавиатуры и дисплея;

УС1 | 001PPPPP | — инициализации опорной частоты;

УС2 | 010IXAAA | — чтения памяти клавиатуры/датчиков;

УС3 | 011IAAAA | — чтения памяти дисплея;

УС4 | 100IAAAA | — записи в память дисплея.

Здесь DD кодирует режим работы дисплея, КК — режим работы клавиатуры, S — режим сканирования, I — признак автоинкрементной адресации, ААА — адрес байта в ОЗУ клавиатуры, АААА — в ОЗУ дисплея.

РРРРР устанавливает коэффициент деления частоты внешнего синхросигнала CLK для получения внутреннего опорного сигнала с частотой не более 100 кГц. После сброса устанавливается максимальный РРРРР, равный 11111.

Режимы работы дисплея (DD):

00 — дисплей на 8 символов с вводом слева;
01 — дисплей на 16 символов с вводом слева;
10 — дисплей на 8 символов с вводом справа;
11 — дисплей на 16 символов с вводом справа.

Режимы работы клавиатуры (КК):

00 — клавиатура в режиме одиночного нажатия клавиш;
01 — клавиатура в режиме N-клавишного нажатия;
10 — сканирование матрицы датчиков;
11 — режим стробируемого ввода.

При S=0 сканирование реализуется в режиме 4-разрядного двоичного счетчика, при S=1 — в режиме инверсного дешифратора на четыре выхода (SL0-SL3). RL0-RL7 — линии возврата.

Управляющие слова УС2-УС4 должны предшествовать чтению или записи информации в память ККД.

Форматы данных, записываемых в ОЗУ клавиатуры:

режим клавиатуры — CNTL SHIFT SL2 SL1 SL0 R2 R1 R0
режим матрицы датчиков — RL7 RL6 RL5 RL4 RL3 RL2 RL1 RL0

Фрагмент программы инициализации ККД:

```
SETB    P1.0            ; Установка адреса регистра управления
MOV     A,#00000000B    ; Загрузка УС0 (8 символов, ввод слева,
MOVX    @R0,A           ; одиночное нажатие клавиш, счетчик)
MOV     A,#(20H+20)     ; Загрузка УС1 (коэффициент деления
MOVX    @R0,A           ; синхросигнала равен 20)
```

Содержимое регистра R0 не имеет значения, так как ККД перекрывает все адресное пространство внешней памяти данных МК51.

Принцип опознания нажатых клавиш в режиме одиночного нажатия таков: если обнаружено нажатие одной клавиши, то в течение следующих двух циклов сканирования клавиатуры будет производиться проверка нажатия других клавиш. Если таких клавиш не будет, то нажатая клавиша признается единственной и код ее записывается в память клавиатуры. Если в течение этих двух циклов будет обнаружено нажатие еще одной клавиши, то в память клавиатуры не заносится код ни одной клавиши до тех пор, пока не будут освобождены все клавиши, кроме одной. Код клавиши заносится в память клавиатуры только один раз на каждое нажатие.

Подпрограмма ввода информации с клавиатуры в РПД МК51 с дублированием информации на дисплее.

```
        ORG     300H            ; Начальный адрес таблицы
                                ; кодов чисел от 0 до 9
        DB      3FH,6,5BH,66H,4FH,6DH,7DH,7,7FH,6FH
CCD:    MOV     R1,#20H         ; Начало массива в РПД МК51
        MOV     R2,#8           ; Число элементов массива
        MOV     R3,#80H         ; Начальное значение УС4
        MOV     DPTR,#300H      ; Настройка указателя данных
WAIT:   JNB     P1.1,WAIT       ; Ожидание нажатия клавиши
        SETB    P1.0            ; Подготовка к чтению буфера
        MOV     A,#40H          ; клавиатуры (УС2)
        MOVX    @R0,A           ;
        CLR     P1.0            ; Чтение кода клавиши
        MOVX    A,@R0           ; из буфера
        MOV     @R1,A           ; Код клавиши в РПД МК51
        MOVC    A,@A+DPTR       ; Код символа в аккумуляторе
        MOV     R4,A            ; Сохранение кода в R4
        SETB    P1.0            ; Подготовка к записи
        MOV     A,R3            ; в ОЗУ дисплея (УС4)
        MOVX    @R0,A           ;
        CLR     P1.0            ; Код символа
        MOV     A,R4            ; в ОЗУ дисплея
        MOVX    @R0,A           ;
        INC     R1              ; Наращивание указателей
        INC     R3              ; памяти
        DJNZ    R2,WAIT         ; Цикл ввода восьми чисел
        RET
```

9 НАПРАВЛЕНИЕ РАЗВИТИЯ ЭЛЕМЕНТНОЙ БАЗЫ 8-РАЗРЯДНЫХ МИКРОКОНТРОЛЛЕРОВ

Отличительные признаки современной элементной базы

1. Завершился переход к однокристальным МК (не предполагается подключение внешней памяти данных и программ).
2. Переход к закрытой архитектуре МК.
3. Модульная организация МК.
4. Выделение типовых функциональных модулей (таймеры, процессоры событий, АЦП, контроллеры последовательных интерфейсов).

Направления развития 8-разрядных МК

Разнообразие структурной организации. Позволяет разработчику для каждой задачи найти МК практически без избыточных ресурсов архитектуры, что обуславливает низкую стоимость конечного изделия. Модульный принцип построения МК — путь к решению этой задачи.

Совершенствование технических характеристик периферийных модулей. Позволяет свести к минимуму число периферийных ИМС на плате контроллера. Один из путей миниатюризации встраиваемой МПС.

Сопряжение с периферийными ИМС по высокоскоростному последовательному интерфейсу. Обеспечивает минимизацию площади проводников на печатной плате. Еще один путь к миниатюризации встраиваемой МПС.

Минимизация энергии потребления. Позволяет уменьшить размеры корпуса МК и габаритные размеры источника питания. Третий путь к миниатюризации.

Расширение диапазона напряжения питания. Одновременно с минимизацией энергии потребления позволяет перевести системы с МК на долговременное питание от автономных источников (аккумуляторов и батареек), что позволяет встраивать их в переносные изделия.

Переход к новым технологиям памяти программ. Мелкосерийное производство вынуждает отказаться от МК с масочным ПЗУ. Сохранение низкой стоимости элементной базы диктует необходимость перехода к EEPROM и FLASH РПП.

Повышение надежности. Способность восстановления нормального функционирования программного обеспечения при его нарушениях в условиях электромагнитных помех и при кратковременных провалах напряжения питания открывает новые возможности применения МК.

Снижение стоимости процесса отладки. Способствует расширению круга разработчиков простейших МПС. Оказывает существенное влияние на стоимость конечного изделия.

Повышение технологичности занесения программы в память МК. Повышает надежность сохранения программы в памяти МК. Переход к технологии программирования в устройстве позволяет отказаться от размещения МК в специальных площадках (снижение стоимости и площади).

Модульный принцип построения

При модульном принципе построения все МК одного семейства содержат в себе базовый функциональный блок — процессорное ядро, который одинаков для всех МК семейства, и изменяемый функциональный блок, который отличает МК разных моделей в пределах одного семейства.

Процессорное ядро включает:

- центральный процессор;
- внутренние магистрали адреса, данных и управления;
- сему формирования многофазной импульсной последовательности для тактирования ЦП и межмодульных магистралей;
- устройство управления режимами работы МК, такими как активный режим, режим пониженного энергопотребления, состояние начального запуска и прерывания.

Изменяемый функциональный блок строится из пяти функциональных групп:

- модули памяти;
- модули встроенных генераторов синхронизации;

- модули периферийных устройств (параллельные порты, многорежимные таймеры/счетчики, процессоры событий, контроллеры последовательного интерфейса, многоканальный АЦП, контроллеры ЖК-индикаторов и светодиодной матрицы);
- модули контроля за напряжением питания и ходом выполнения программы;
- модули внутрисхемной отладки и программирования.

Процессорное ядро представляет неразрывное единство трех составляющих его технического решения, определяющих его производительность:

- архитектуры с присущим ей набором регистров, способами адресации операндов, системой команд, организацией процесса выборки и исполнения команд;
- схемотехники, которая определяет последовательность перемещения данных по внутренним магистралям между РОНами, АЛУ и памятью при выполнении каждой команды;
- технологии, которая позволяет разместить схему той или иной сложности на полупроводниковом кристалле, определяет допустимую частоту переключений в схеме и энергию потребления.

Резидентная память МК

Закрытая архитектура современных 8-разрядных МК стала реализуемой лишь при условии интеграции на кристалл МК модулей памяти двух типов: энергонезависимого ЗУ памяти программ и ОЗУ памяти данных.

С момента появления МК технология ПЗУ претерпела множество изменений, что привело к повышению быстродействия, информационной емкости, надежности хранения информации и появлению принципиально новых технологий программирования РПП.

Различают пять типов ПЗУ:

- ПЗУ масочного типа — mask-ROM. Самое простое, надежное, но экономически оправдано при партии в несколько десятков тысяч штук;

- ПЗУ, однократно программируемое пользователем — OTPROM (One-Time Programmable ROM). Рекомендуется в изделиях, выпускаемых небольшими партиями;
- ПЗУ, программируемое пользователем, с ультрафиолетовым стиранием — EPROM (Erasable Programmable ROM). Имеют высокую стоимость. Рекомендуется только в опытных образцах изделий;
- ПЗУ, программируемое пользователем, с электрическим стиранием — EEPROM (Electrically Erasable Programmable ROM). Побайтовое стирание и запись информации. Имеют ограниченную емкость;
- ПЗУ с электрическим стиранием типа FLASH — FLASH ROM. Транзистор адресации каждого бита удален. Программирование и стирание осуществляется страницами (8, 16 или 32 байта) или блоками (до 60 Кбайт). Снижение стоимости и размеров делает FLASH ROM конкурентным даже с масочным ПЗУ.

В ранних разработках повышенное напряжение для программирования подавалось на один из выводов МК. В новейших версиях EEPROM и FLASH ПЗУ содержат встроенные схемы повышающих преобразователей напряжения — генераторы накачки. Допускается включение и отключение генераторов накачки посредством установки битов в регистре специальных функций модуля памяти без остановки выполнения прикладной программы. Появляется возможность программирования под управлением программы (FLASH ПЗУ используется для хранения программы, а EEPROM — для хранения изменяемых в процессе эксплуатации настроек пользователя.

ОЗУ РПД всегда статического типа, что допускает снижение частоты тактирования до сколь угодно малых значений с целью снижения энергии потребления. Уровень напряжения хранения информации в режиме микропотребления порядка 1 В, что позволяет перейти на питание от батарейки. Появились МК со встроенным в корпус автономным источником питания, гарантирующим сохранность данных в ОЗУ в течение 10 лет (МК DS5000 фирмы Dallas Semiconductor).

Таймеры и процессоры событий

Управление в реальном времени означает способность МПС получить информацию о состоянии управляемого объекта, выполнить необходимые расчеты и сформировать управляющее воздействие в течение интервала времени, по истечении которого эти воздействия вызовут желаемое поведение объекта. От процессорного ядра зависит время вычислений. Но надо тратить как можно меньше времени на прием информации с датчиков и выдачу управляющих сигналов. Для решения этих задач используют прежде всего систему прерываний и модуль таймера. Развитая система прерываний позволяет сократить время реакции МП-системы на изменения состояния объекта. Модули таймеров служат для приема информации от датчиков с времяимпульсными выходами, а также для формирования управляющих воздействий в виде последовательности импульсов с изменяющимися параметрами.

Опыт построении МПС позволяет выделить типовые задачи, которые должен уметь решать МК для эффективного управления в реальном времени:

- отсчет равных интервалов времени заданной длительности, повтор алгоритма управления по истечении каждого такого временного интервала. Обычно эту функцию называют формированием меток реального времени;
- контроль за изменением состояния линии ввода МК;
- измерение длительности сигнала заданного логического уровня на линии ввода МК;
- подсчет числа импульсов внешнего сигнала на заданном временном интервале;
- формирование на линии вывода МК сигнала заданного логического уровня с программируемой задержкой по отношению к изменению сигнала на линии ввода;
- формирование на линии вывода МК импульсного сигнала с программируемой частотой и программируемым коэффициентом заполнения.

Каждая из перечисленных задач может быть выполнена программными средствами, без использования специальных аппаратных решений. Можно реализовать программную временную задержку, загрузив в регистр число и повторяя декрементирование регистра до нуля. Для контроля за изменением состояния линии ввода можно организовать постоянный опрос линии — поллинг. Однако нельзя будет параллельно с этим производить вычисления или решать несколько задач. Поэтому, для выполнения функций, связанных с управлением в реальном времени, в состав МК включают специальные аппаратные средства, называемые таймерами.

Модуль таймера 8-разрядного МК представляет собой 16-разрядный счетчик со схемой управления. В карте памяти он отображается двумя регистрами — TH и TL. Они доступны для чтения и записи. Счетчик работает на сложение. Может использовать для тактирования импульсную последовательность с управляемого делителя частоты f_{bus}, либо внешнюю импульсную последовательность, поступающую на один из входов МК. При переполнении устанавливается флаг TF и генерируется запрос на прерывания, если прерывания разрешены. Классический модуль таймера используется в МК с архитектурой MCS-51. Дополнительная логика позволяет измерять длительность импульса, поступающего на одну из линий МК. Наличие режима перезагрузки позволяет получить метки реального времени с периодом, отличным от $К_{сч}=2^{16}$.

Совершенствование системы реального времени достигается увеличением числа модулей таймеров, а также введением аппаратных средств входного захвата (Input Capture) и выходного сравнения (Output Compare).

Канал входного захвата (рис. 9.1) содержит детектор событий, который наблюдает за уровнем напряжения на одном из входов МК. При изменении сигнала на линии с 0 на 1 или наоборот текущее состояние счетчика таймера записывается в 16-разрядный регистр данных TIC канала захвата. Выбор типа события захвата устанавливается в процессе инициализации модуля таймера и может многократно изменяться по ходу выполнения программы. Каждое событие захвата отмечается установкой в 1 триггера TIC. Состояние триггера может быть считано программно, а если разрешены прерывания по событию входного захвата, то генерируется запрос на прерывание.

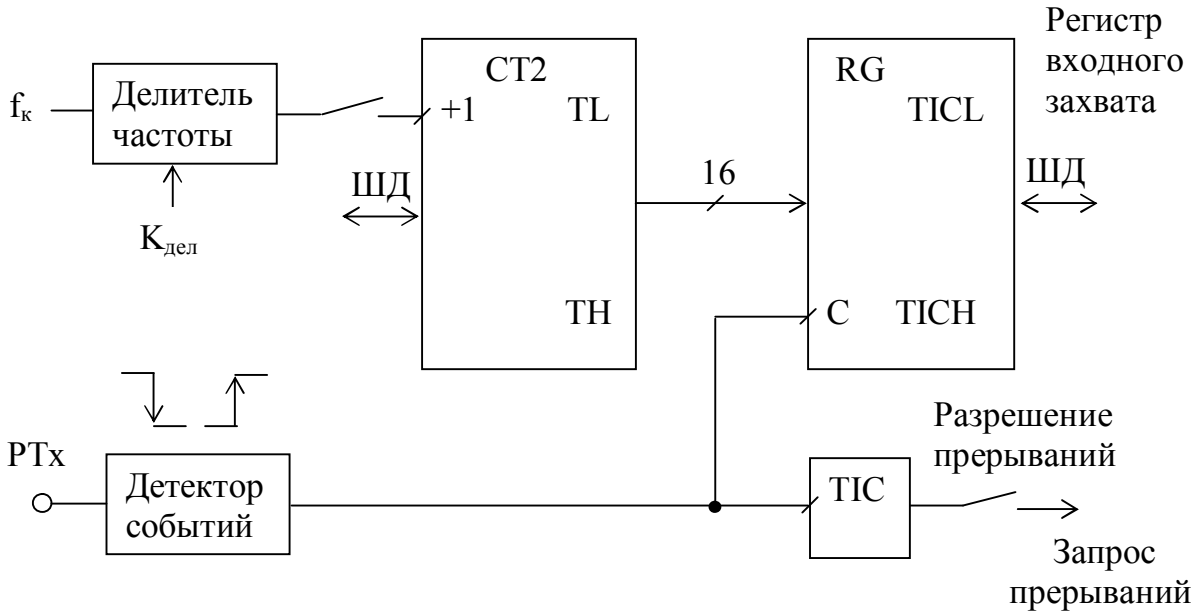


Рис. 9.1 — Канал входного захвата таймера

В канале выходного сравнения (рис. 9.2) многоразрядный цифровой компаратор непрерывно сравнивает изменяющийся во времени код счетчика таймера с кодом, который записан в 16-разрядном регистре TOC канала сравнения.

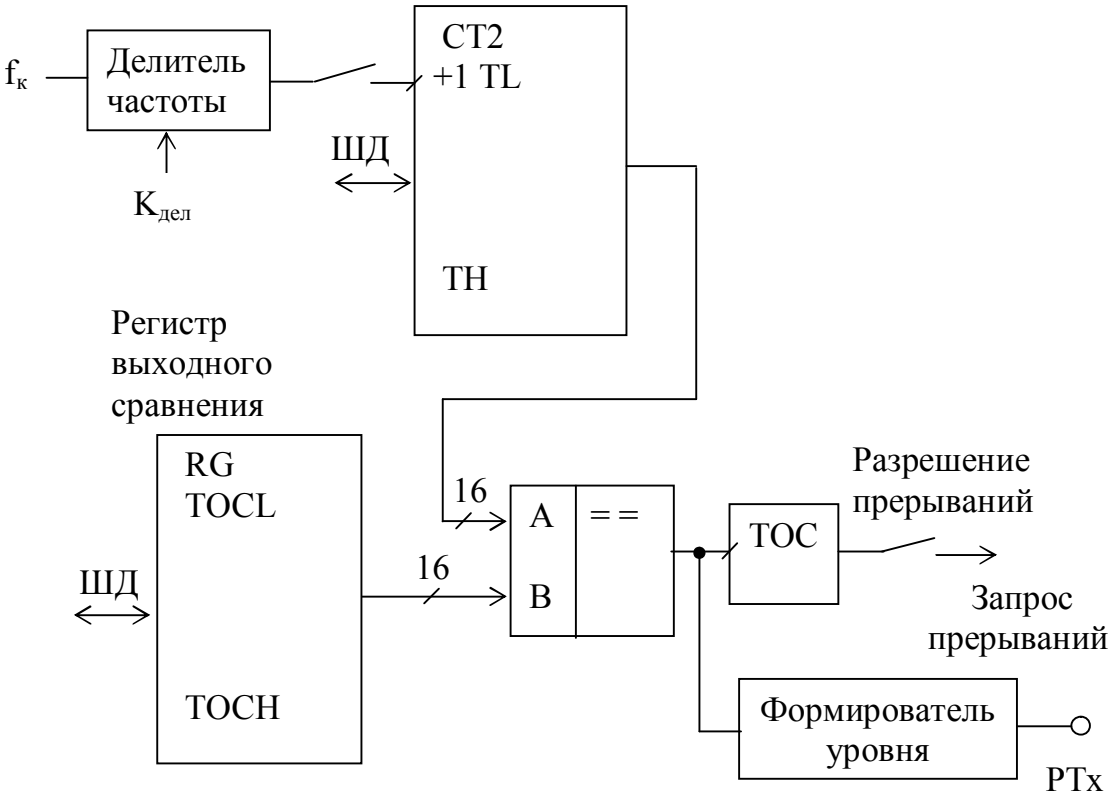


Рис. 9.2 — Канал выходного сравнения таймера

В момент равенства кодов на одном из выводов МК устанавливается заданный уровень выходного сигнала. Каждое событие выходного сравнения отмечается установкой в 1 триггера TOC. Состояние триггера может быть считано программно, а если разрешены прерывания по событию выходного сравнения, то генерируется запрос на прерывание.

Следующий этап в развитии подсистемы реального времени — модули процессоров событий. Модуль содержит в себе 16-разрядный счетчик временной базы и некоторое количество универсальных каналов захвата/сравнения, которые могут работать в одном из трех режимов:

- режим входного захвата;
- режим выходного сравнения;
- режим широтно-импульсной модуляции.

Сторожевой таймер

Предназначен для защиты микроконтроллера от сбоев в процессе работы. Представляет собой многоразрядный счетчик. При сбросе он обнуляется. При переходе в активный режим работы увеличивает код независимо от выполняемой программы и при переполнении генерирует сигнал внутреннего сброса МК. Для исключения этого события прикладная программа должна периодически сбрасывать счетчик.

Контроллеры последовательного ввода/вывода

Задачи, которые решаются средствами контроллера последовательного ввода/вывода могут быть условно разделены на три группы:

- связь встраиваемой МПС с системой управления верхнего уровня: промышленным компьютером, офисным компьютером. Наиболее часто для этой цели используются интерфейсы RS-232 и RS-485;
- связь с внешними по отношению к МК периферийными ИС встраиваемой МПС, а также с датчиками физических величин

с последовательным выходом. Для этой цели используются интерфейсы SPI, I2C;

- интерфейс связи с локальной сетью в мультимикропроцессорных системах. В сложных системах более популярным становится протокол типа CAN.

Минимизация потребления энергии в системах с МК

Кроме активного режима большинство современных МК могут быть переведены в режим ожидания и режим останова с пониженным энергопотреблением.

В режиме ожидания прекращает работу центральный процессор, но продолжают работать периферийные модули, которые отслеживают поведение объекта управления. При необходимости периферийные модули переводят МК в активный режим для вычисления корректирующих воздействий на объект управления. Мощность потребления снижается в 5–10 раз. Выход из режима по сбросу и прерыванию.

В режиме останова прекращают работу и ЦП, и большинство периферийных модулей. Мощность потребления составляет единицы мкВт. Выход из режима по сбросу.

Различают три группы исполнения по напряжению питания: 5 В±10 %, с расширенным диапазоном питания (от 2–3 до 5–7 В), с пониженным напряжением питания (от 1,8 до 3 В).

Примеры устройств, где используется режим пониженного энергопотребления: пульт дистанционного управления бытовой аппаратуры и счетчик тепловой энергии.

Микроконтроллер семейства AT89 фирмы Atmel представляет собой восьмиразрядную однокристальную микроЭВМ с системой команд MCS-51 фирмы Intel. Базовая структура микроконтроллеров совпадает с базовой структурой микроконтроллеров семейства MCS-51 и отечественных микроконтроллеров МК51 серий К1816/51 и К1830/51, однако микроконтроллеры многих типов содержат новые запоминающие и периферийные устройства, а некоторые устройства базовой структуры имеют иные характеристики.

10 МИКРОКОНТРОЛЛЕРЫ СЕМЕЙСТВА AT89 фирмы Atmel

В табл. 10.1 перечислены типы микроконтроллеров семейства AT89, указаны запоминающие и периферийные устройства и некоторые узлы, входящие в состав микроконтроллеров каждого типа, и приведены их характеристики.

Таблица 10.1

Тип МК	FLASH	SRAM	EEPROM	EM	I/O	SP	T/C	IS	IV	SPI	WDT	AC	DPTR
AT89C1051	1K	64	–	–	15	–	1	3	3	–	–	+	1
AT89C1051U	1K	64	–	–	15	+	2	6	5	–	–	+	1
AT89C2051	2K	128	–	–	15	+	2	6	5	–	–	+	1
AT89C4051	4K	128	–	–	15	+	2	6	5	–	–	+	1
AT89C51 AT89LV51	4K	128	–	+	32	+	2	6	5	–	–	–	1
AT89C52 AT89LV52	8K	256	–	+	32	+	3	8	6	–	–	–	1
AT89C55 AT89LV55	20K	256	–	+	32	+	3	8	6	–	–	–	1
AT89S53 AT89LS53	12K	256	–	+	32	+	3	9	6	+	+	–	2
AT89S8252 AT89LS8252	8K	256	2K	+	32	+	3	9	6	+	+	–	2
AT89S4D12	4K	256	128K	–	5	–	–	–	–	+	–	–	2

В число запоминающих устройств входят внутреннее постоянное запоминающее устройство (FLASH), предназначенное для хранения команд программы и констант, и внутреннее оперативное запоминающее устройство (SRAM), предназначенное для хранения данных. FLASH память программ выдерживает до 1000 циклов перепрограммирования. SRAM является статическим оперативным запоминающим устройством.

Микроконтроллеры некоторых типов имеют «новое» запоминающее устройство — внутреннее перепрограммируемое запоминающее устройство для хранения данных (EEPROM). Первоначальная запись данных в EEPROM производится при программировании микроконтроллера. В процессе выполнения программы обращения к EEPROM для чтения и записи выполняются с использованием команд с мнемокодами операции MOVX. После обращения для записи в EEPROM выполняется цикл записи длительностью несколько мс, в течение которого новое обращение к EEPROM невозможно.

В табл. 10.1 указана емкость названных запоминающих устройств (число восьмиразрядных ячеек памяти). К микроконтроллерам некоторых типов не может подключаться внешняя память (External Memory, EM). Отсутствие возможности подключения внешней памяти отмечено знаком <–> в колонке EM.

К числу периферийных устройств относятся восьмиразрядные параллельные порты ввода-вывода P0, P1, P2, P3, последовательный порт SP, таймеры-счетчики T/C0, T/C1, T/C2 и контроллер прерываний. Микроконтроллеры некоторых типов содержат меньшее число параллельных портов, а некоторые порты имеют меньшее число входов-выходов. Суммарное число входов-выходов параллельных портов у микроконтроллера указано в табл. 10.1 в колонке I/O. У микроконтроллеров некоторых типов отсутствует таймер-счетчик T/C2, при этом у некоторых микроконтроллеров отсутствует также таймер-счетчик T/C1. Число таймеров-счетчиков у микроконтроллера указано в колонке T/C.

Система прерываний имеет два уровня приоритета. Число источников запросов прерывания (Interrupt Source, IS) и векторов прерывания (Interrupt Vector, IV) у микроконтроллеров разных типов указано в табл. 10.1 в колонках IS и IV соответственно. «Новыми» периферийными устройствами являются блок последовательного периферийного интерфейса (SPI), сторожевой таймер (WDT) и аналоговый компаратор (AC). Наличие у микроконтроллера названных устройств отмечено знаком <+> в колонках SPI, WDT и AC соответственно.

Блок SPI предназначен для последовательного ввода и вывода данных с использованием трех шин. При этом микроконтроллер может работать в качестве ведущего или ведомого уст-

ройства. Блок SPI может быть использован также для программирования микроконтроллера после установки его в аппаратуру.

Аналоговый компаратор сравнивает по величине напряжения сигналы, поступающие на входы P1.0 и P1.1. (рис. 10.1). Результат сравнения подается на вход P3.6, не имеющий внешнего вывода. Процессор у микроконтроллеров некоторых типов содержит два регистра-указателя данных — DPTR0 и DPTR1. Количество регистров-указателей данных у микроконтроллера указано в колонке DPTR.

Микроконтроллеры семейства AT89 выпускаются для работы при разных значениях напряжения питания и тактовой частоты, определяемой частотой подключенного к микроконтроллеру кварцевого резонатора. Диапазоны значений напряжения питания (Vcc) и тактовой частоты (Fosc) у микроконтроллеров разных типов указаны в табл. 10.2. Ток потребления зависит от величины напряжения питания и тактовой частоты. В табл. 10.2 приведены значения тока потребления в рабочем режиме (Icc) при максимальном значении напряжения питания и Fosc=12 МГц.

Таблица 10.2

Тип МК	**Vcc (В)**	**Fosc (МГц)**	**Icc (мА)**	**N**
AT89C1051	2,7–6,0	0–24	15	20
AT89C1051U	2,7–6,0	0–24	15	20
AT89C2051	2,7–6,0	0–24	15	20
AT89C4051	2,7–6,0	0–24	15	20
AT89C51	4,0–6,0	0–24	20	40
AT89LV51	2,7–6,0	0–12	20	40
AT89C52	4,0–6,0	0–24	25	40
AT89LV52	2,7–6,0	0–12	25	40
AT89C55	4,0–6,0	0–33	25	40
AT89LV55	2,7–6,0	0–12	25	40
AT89S53	4,0–6,0	0–33	25	40
AT89LS53	2,7–6,0	0–12	25	40
AT89S8252	4,0–6,0	0–33	25	40
AT89LS8252	2,7–6,0	0–12	25	40
AT89S4D12	3,3 (+–10%)	12–15	20	10

Ниже приведены основные характеристики двух микросхем семейства AT89.

Микроконтроллер AT89C4051

- Совместимость с ИС семейства MCS-51.
- 4 Кбайта перепрограммируемой Flash памяти.
- Ресурс: 1000 циклов записи/ стирания.
- Напряжение питания от 2,7 В до 6 В.
- Полностатический режим работы: от 0 Гц до 24 МГц.
- Двухуровневая защита программирования памяти.
- Встроенная 128 x 8 бит RAM.
- 15 программируемых линий I/O.
- Два 16-разрядных таймера/ счетчика.
- 6 источников прерывания.
- Программируемый последовательный канал UART.
- Выходы с поддержкой прямого управления светодиодными сегментами.
- Встроенный аналоговый компаратор.
- Экономичные режимы ожидания (Idle) и отключения (Power — down).
- Распознавание режима аварийного отключения питания.

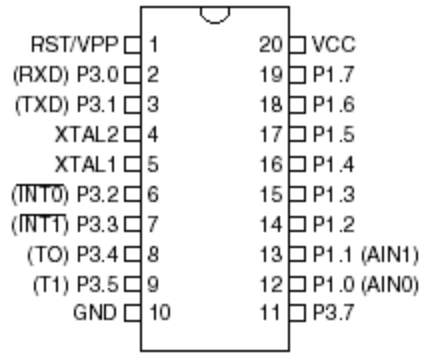


Рис. 10.1 — Расположение выводов AT89C4051

Микроконтроллер AT89S51

- Совместимость с серией MCS-51.

- 4 КБ флэш-памяти с внутрисхемным программированием (ISP).
- Рабочий диапазон питания 4,0…5,5 В
- Полностью статическое функционирование: 0 …33 МГц.
- Три уровня защиты памяти программ.
- Внутреннее ОЗУ размером 128×8.
- 32 программируемые линии ввода-вывода.
- Два 16-разрядных таймера-счетчика.
- Шесть источников прерываний.
- Полнодуплексный канал последовательной связи на UART.
- Режимы снижения потребления: холостой ход и экономичный.
- Восстановление прерываний при выходе из экономичного режима.
- Сторожевой таймер.
- Двойной указатель данных.
- Флаг выключения питания.
- Быстрое время программирования.
- Гибкое внутрисхемное программирование.

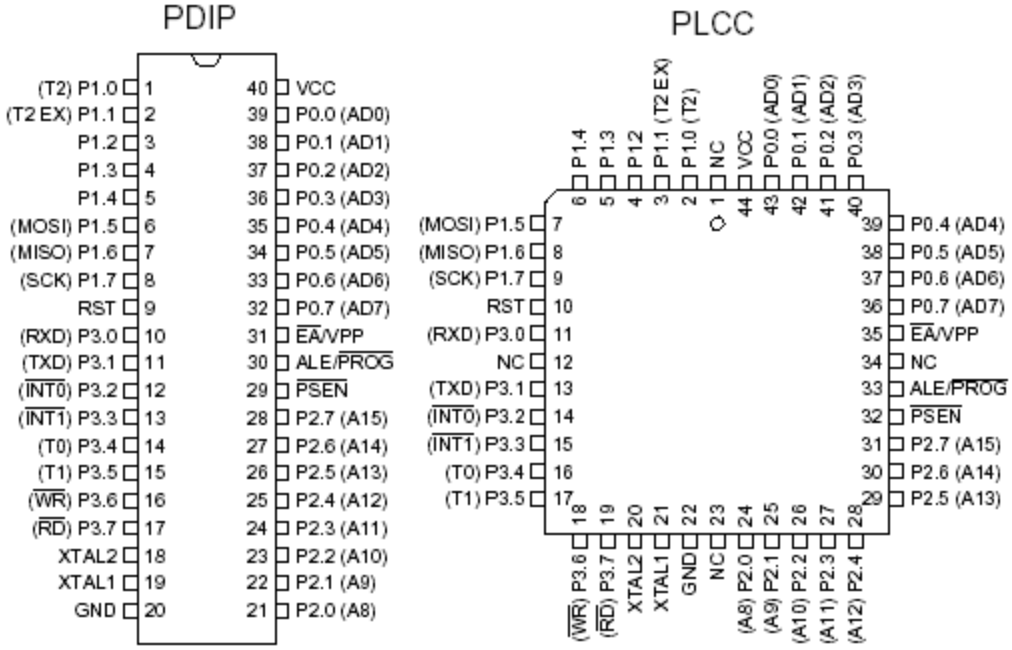


Рис. 10.2 — Расположение выводов AT89S51

11 ПРИМЕРЫ ВОПРОСОВ КОМПЬЮТЕРНОЙ КОНТРОЛЬНОЙ РАБОТЫ

1. Определить содержимое аккумулятора после выполнения команд (два шестнадцатеричных символа)

```
ORG     0
MOVC    A,@A+PC
CLR     C
SUBB A,#100
```

2. Определить содержимое аккумулятора после выполнения команд (два шестнадцатеричных символа)

```
MOV     B,#27H
MOV     A,#100
ADD     A,B
DA      A
XRL     A,#5
```

3. Определить содержимое регистра DPTR после выполнения команд (четыре шестнадцатеричных символа)

```
 ORG    0
MOV     B,SP
MOV     A,#100
MUL     AB
MOV     DPH,B
MOV     DPL,A
```

4. Записать третий байт команды CJNE A,#100,$-5

5. Транслировать команду SJMP $+10

6. Записать второй байт команды MOV C,P

7. Оценить время выполнения команд в микросекундах (fк=12 МГц)

```
MOV     A,#64H
DJNZ    ACC,$
```

8. Какое из прерываний будет иметь высший приоритет после выполнения команд

```
 MOV    IE,#9FH
MOV     IP,#0AH
```

9. Оценить время (в мкс) выполнения команды MUL AB при fк=4 МГц

ЛИТЕРАТУРА

1. Шарапов А.В. Цифровые и микропроцессорные устройства: Учебное пособие. — Томск: ТМЦ ДО, 2003. — 166 с.

2. Сташин В.В., Урусов А.В., Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах. — М.: Энергоатомиздат, 1990. — 224 с.

3. Боборыкин А.В., Липовецкий Г.П. и др. Однокристальные микроЭВМ: Справочник. — М.: БИНОМ, 1994. — 400 с.

4. Ремизевич Т.В. Микроконтроллеры для встраиваемых приложений: от общих подходов — к семействам HC05 и HC08 фирмы Motorola /под ред. Кирюхина И.С. — М.: ДОДЭКА, 2000. — 272 с.

ЧАСТЬ 2. Микроконтроллеры AVR

ПРЕДИСЛОВИЕ

За время своего относительно недолгого существования (с 1997 года) микроконтроллеры семейства AVR (серия AT90S) фирмы Atmel Corp. приобрели чрезвычайную популярность в мире и в России. В 2001 году более половины российских разработок с использованием 8-разрядных микроконтроллеров было создано именно на AVR. На сегодняшний день продолжается развитие в том же направлении. Это легко объяснить: благодаря очень удачной архитектуре ядра процессора и широкому набору периферийных модулей на кристалле существенно облегчается процесс программирования конечного устройства. Еще одним фактором, привлекательным для разработчиков, стал выпуск в 2002 году AVR-микроконтроллеров следующего поколения. Эти микросхемы производятся по технологическим нормам 0,35 мкм, в отличие от первых AT90S, выпускающихся по технологии 0,5 мкм. Заметное отличие – увеличенная в два раза тактовая частота новых контроллеров – 16 МГц, обеспечивающая производительность до 16 MIPS (типовая команда у AVR-контроллеров выполняется за один период тактовой частоты).

Кроме лекционного курса вторая часть пособия включает руководство к выполнению трех компьютерных лабораторных работ. Для их реализации используются программные средства отладки, свободно распространяемые фирмой Atmel (отладчик AVR Studio, компилятор CVAVR и симулятор VMLAB).

В заключительном разделе пособия приведен пример проектирования измерителя частоты сети (микроконтроллер ATmega16 с программой на Си) и отладкой устройства с помощью симуляторов VMLAB и Proteus VSM.

1 ОБЩАЯ ХАРАКТЕРИСТИКА МИКРОКОНТРОЛЛЕРОВ AVR, ПРОГРАММНАЯ МОДЕЛЬ И СИСТЕМА КОМАНД

В настоящее время в серийном производстве находятся два семейства AVR – Tiny и Mega. Микроконтроллеры семейства Tiny имеют небольшой объем памяти программ и весьма ограниченную периферию. Они выпускаются в 8-выводных корпусах и являются самыми дешевыми. Наиболее развитую периферию, наибольшие объемы памяти данных и программ имеют микроконтроллеры семейства Mega.

AVR, пожалуй, одно из самых интересных направлений, развиваемых корпорацией Atmel. Они представляют собой мощный инструмент для создания современных высокопроизводительных и экономичных многоцелевых контроллеров. 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 становится еще одним индустриальным стандартом среди 8-разрядных микроконтроллеров общего назначения.

Сама идея создания нового RISC-ядра родилась в 1994 году в Норвегии. В 1995 году два его изобретателя Альф Боген (Alf-Egil Bogen) и Вегард Воллен (Vegard Wollen) предложили корпорации Atmel выпускать новый 8-разрядный RISC-микроконтроллер как стандартное изделие и снабдить его Flash-памятью программ на кристалле. Руководство Atmel Corp. приняло решение инвестировать данный проект. В 1996 году был основан исследовательский центр в городе Тронхейм (Норвегия). Оба изобретателя стали директорами нового центра, а микроконтроллерное ядро было запатентовано и получило название **AVR** (**A**lf-Egil Bogen + **V**egard Wollen + **R**ISC). Первый опытный кристалл 90S1200 был выпущен на стыке 1996–1997 годов, а с 3 квартала 1997 года корпорация Atmel приступила к серийному производству нового семейства микроконтроллеров и к их рекламной и технической поддержке.

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) и индустриальный (-40C...+85C).

Все микроконтроллеры 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 (рис. 1.1). Число циклов перезаписи – не менее 1000.

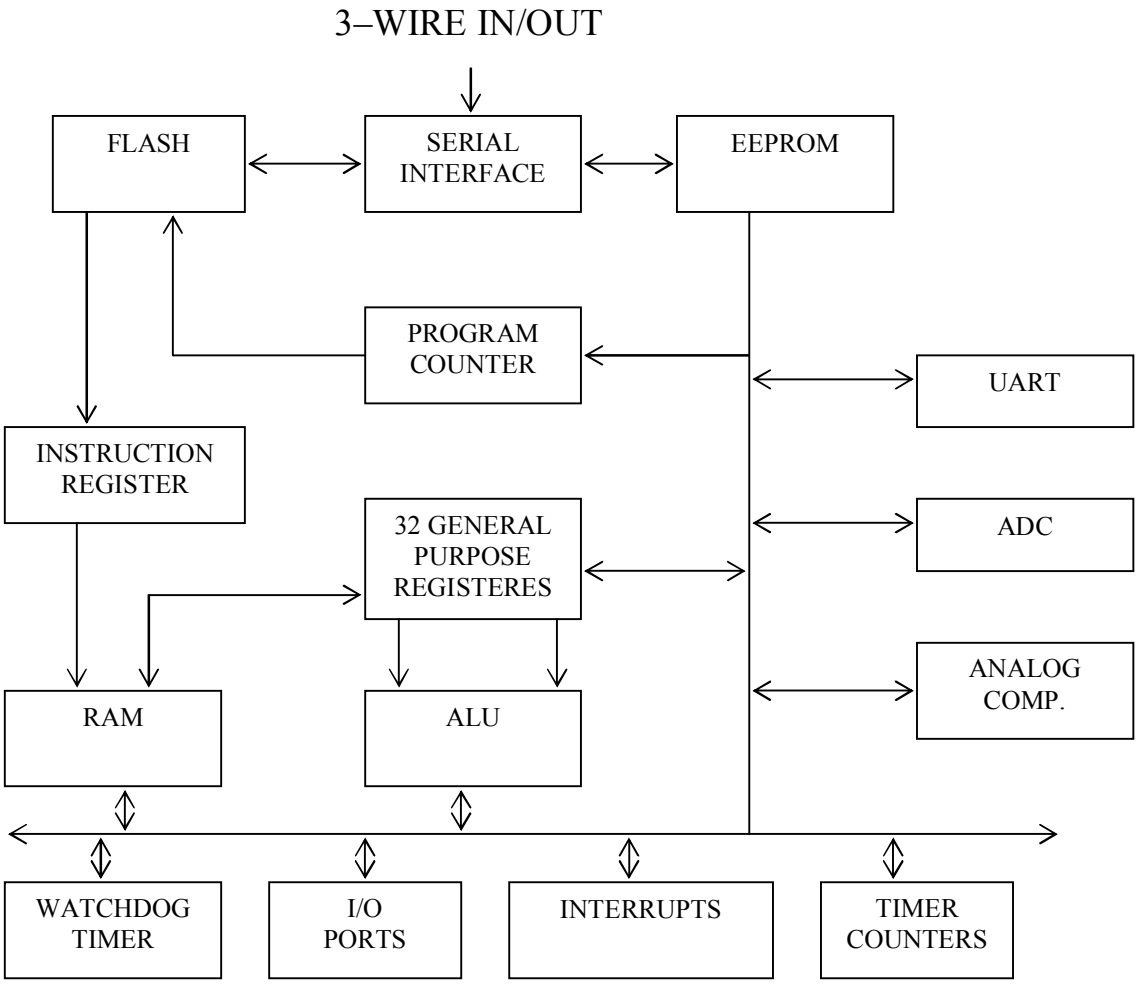


Рис. 1.1 – Структурная схема AVR

Все AVR имеют также блок энергонезависимой электрически стираемой памяти данных EEPROM. Этот тип памяти, дос-

тупный 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.

Следующая отличительная черта архитектуры микроконтроллеров AVR – регистровый файл быстрого доступа (рис. 1.2). 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 MCS-51). 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, причем этот цикл равен периоду тактового генератора. Старшие регистры объединены парами и образуют три 16-разрядных регистра X, Y и Z, предназначенных для косвенной адресации ячеек памяти (AVR без RAM имеют только один 16-битный регистр Z).

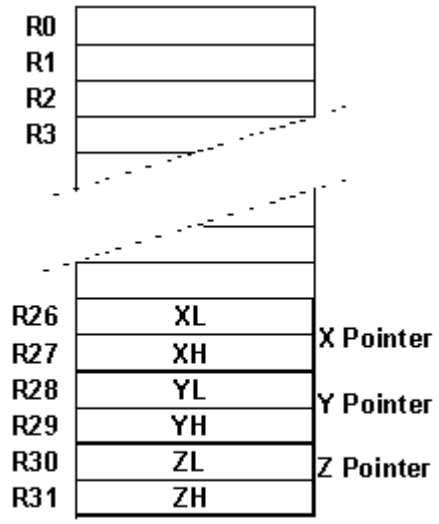


Рис. 1.2 – Регистровый файл

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. Верхние границы частотного диапазо-

на гарантируют устойчивую работу микроконтроллеров при работе во всем температурном диапазоне.

Программная модель AVR-микроконтроллеров приведена на рис.1.3. Для хранения оперативных данных программист, кроме регистрового файла, может использовать внутреннюю и внешнюю (если они имеются) блоки RAM. Поскольку внутренняя и внешняя RAM входят в единое адресное пространство (вместе с оперативными регистрами и регистрами ввода/вывода), то для доступа к ячейкам внутренней и внешней памяти используются одни и те же команды. Внутренняя оперативная память RAM отсутствует у кристаллов семейства Tiny, но имеется у всех AVR семейства Mega. Для некоторых микроконтроллеров возможна организация подключения внешней памяти данных объемом до 64Кб.

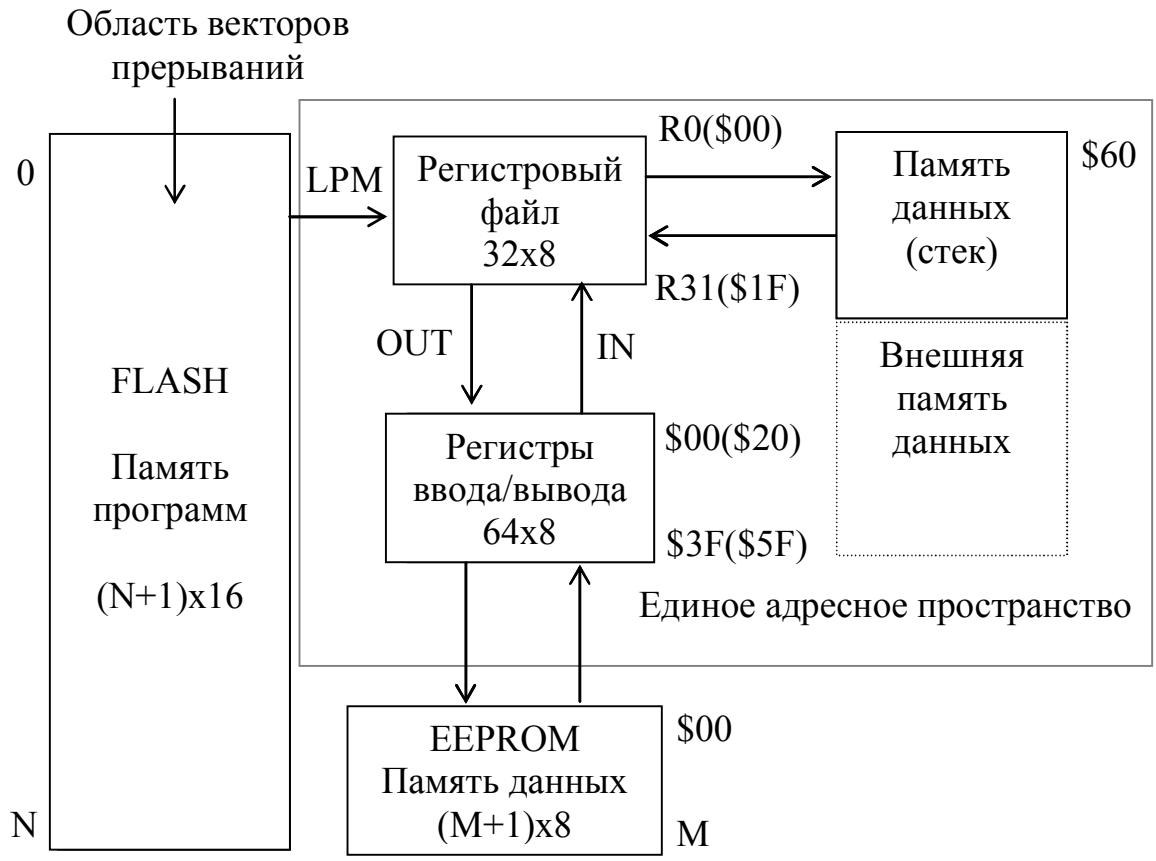


Рис. 1.3 – Программная модель AVR-микроконтроллеров

Регистровый файл, блок регистров ввода/вывода и память данных образуют единое адресное пространство, что дает воз-

можность при программировании обращаться к 32 оперативным регистрам и к регистрам ввода/вывода как к ячейкам памяти, используя команды доступа к RAM (в том числе и с косвенной адресацией). Младшие 32 адреса ($0 — $1F) соответствуют оперативным регистрам. Следующие 64 адреса ($20 — $5F) зарезервированы для регистров ввода/вывода. Внутренняя RAM начинается с адреса $60 (знак $ указывает на шестнадцатеричную систему счисления).

Таким образом, регистры ввода/вывода имеют двойную нумерацию. Если используются команды IN, OUT, SBI, CBI, SBIC, SBIS, то следует использовать нумерацию регистров ввода/вывода, начинающуюся с нуля (назовем ее основной). Если же к регистрам ввода/вывода доступ осуществляется как к ячейкам памяти, то необходимо использовать нумерацию единого адресного пространства оперативной памяти данных AVR. Очевидно, что адрес в едином адресном пространстве памяти данных получается путем прибавления числа $20 к основному адресу регистра ввода/вывода. Кроме оперативной памяти программно доступными ресурсами микроконтроллера являются энергонезависимые, электрически программируемые FLASH и EEPROM блоки памяти, которые имеют отдельные адресные пространства.

Так как все команды AVR представляют собой 16-разрядные слова, FLASH-память организована как последовательность 16-разрядных ячеек и имеет емкость от 512 слов до 128K слов в зависимости от типа кристалла.

Во FLASH-память, кроме программы, могут быть записаны постоянные данные, которые не изменяются во время функционирования микропроцессорной системы. Это различные константы, таблицы знакогенераторов, таблицы линеаризации датчиков и т.п. Данные из FLASH памяти могут быть программным образом считаны в регистровый файл при помощи команды LPM (см. группу команд передачи данных).

EEPROM-блок электрически стираемой памяти данных AVR предназначен для хранения энергонезависимых данных, которые могут изменяться непосредственно на объекте. Это калибровочные коэффициенты, различные уставки, конфигурационные параметры системы и т.п. EEPROM-память данных может быть программным путем как прочитана, так и записана. Однако спе-

циальных команд обращения к EEPROM-памяти нет. Чтение и запись ячеек EEPROM выполняются через регистры ввода/вывода EEAR (регистр адреса), EEDR (регистр данных) и EECR (регистр управления).

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 своим 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).

Система реального времени (RTC) реализована во всех микроконтроллерах семейства Mega. Таймер/счетчик RTC имеет свой собственный предделитель, который может быть программным способом подключен или к основному внутреннему источнику тактовой частоты микроконтроллера, или к дополнительному асинхронному источнику опорной частоты (кварцевый резонатор или внешний синхросигнал). Для этой цели зарезервированы два внешних вывода микроконтроллера. Внутренний осциллятор, нагруженный на счетный вход таймера/счетчика RTC, оптимизирован для работы с внешним «часовым» кварцевым резонатором 32,768 кГц.

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. Мощные 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).

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 системе 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 сигна-

лов, а также максимально просто реализовывать АЦП двухтактного интегрирования.

Аналого-цифровой преобразователь (АЦП) 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 разряда. 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. 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.

Выполнять арифметико-логические операции и операции сдвига непосредственно над содержимым ячеек памяти нельзя. Нельзя также записать константу или очистить содержимое ячейки памяти. Система команд AVR позволяет лишь выполнять операции обмена данными между ячейками RAM и оперативными регистрами. Достоинством системы команд можно считать разнообразные режимы адресации ячеек памяти. Кроме прямой адресации (см. группу команд передачи данных) имеются следующие режимы: косвенная, косвенная с пост-инкрементом, косвенная с предекрементом и косвенная со смещением.

Регистры ввода/вывода располагаются в так называемом адресном пространстве ввода/вывода размером 64 байт. Их можно разделить на две группы: служебные регистры микроконтроллера и регистры, относящиеся к периферийным устройствам (в том числе порты ввода/вывода). Изучение данных регистров удобно

выполнять одновременно с изучением конкретного периферийного узла.

Среди регистров ввода/вывода есть регистр, используемый наиболее часто в процессе выполнения программы. Это регистр статуса SREG. Он располагается по адресу $3F и содержит набор флагов (табл. 1.1), показывающих текущее состояние микроконтроллера. Большинство флагов автоматически устанавливаются в соответствии с результатом выполнения команд. Все разряды SREG доступны как для записи, так и для чтения. После сброса микроконтроллера регистр обнулен.

Все регистры ввода/вывода могут считываться и записываться через оперативные регистры при помощи команд IN, OUT (см. группу команд передачи данных). Регистры ввода/вывода, имеющие адреса в диапазоне $00 — $1F, обладают возможностью побитовой адресации. Непосредственная установка и сброс отдельных разрядов этих регистров выполняется командами SBI и CBI (см. группу команд работы с битами). Для признаков результата операции, которые являются битами регистра ввода/вывода SREG, имеется целый набор команд установки и сброса. Команды условных переходов в качестве своих операндов могут иметь как биты-признаки результата операции, так и отдельные разряды побитно адресуемых регистров ввода/вывода.

Следует также иметь в виду, что у разных типов AVR одни и те же регистры ввода/вывода могут иметь различные адреса. Для того, чтобы обеспечить переносимость программного обеспечения с одного типа кристалла на другой, следует использовать в программе стандартные, принятые в оригинальной фирменной документации, символические имена регистров ввода/вывода, а соответствие этих имен реальным адресам задавать, подключая в начале своей программы (при помощи директивы ассемблера **.INCLUDE**) файл определения адресов регистров ввода/вывода. Файлы определения адресов регистров ввода/вывода имеют расширение **.inc**. Они уже созданы разработчиками фирмы ATMEL и свободно распространяются вместе с документацией на AVR-микроконтроллеры. В этих файлах задается соответствие символических имен основным адресам регистров ввода/вывода.

Таблица 1.1 – Разряды регистра состояния SREG

Разряд	Название	Описание
7	**I**	**Общее разрешение прерываний.** Для разрешения прерываний этот флаг должен быть установлен в 1. Флаг сбрасывается аппаратно после входа в подпрограмму обслуживания прерываний и восстанавливается командой RETI для разрешения обработки следующих прерываний
6	**T**	**Хранение копируемого бита.** 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 потетрадного переноса.** Этот флаг устанавливается в 1, если произошел перенос из младшей тетрады байта (из 3-го разряда в 4-й) или заем из старшей тетрады при выполнении некоторых арифметических операций
4	**S**	**Флаг знака.** Этот флаг равен результату операции «Исключающее ИЛИ» между флагами N и V. Он 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
2	**N**	**Флаг отрицательного значения.** Этот флаг устанавливается в 1, если старший разряд результата операции (7 разряд) равен 1
1	**Z**	**Флаг нуля**. Этот флаг устанавливается в 1 при нулевом результате выполнения операции
0	**C**	**Флаг переноса.** Этот флаг устанавливается в 1, если в результате выполнения операции произошел выход за границы байта

Младшие адреса памяти программ имеют специальное назначение. Адрес $000 является адресом, с которого начинает выполняться программа после сброса процессора. Начиная со следующего адреса ячейки памяти программ образуют область векторов прерывания. В этой области для каждого возможного источника прерывания отведен свой адрес, по которому (в случае использования данного прерывания) размещают команду относи-

тельного перехода RJMP на подпрограмму обработки прерывания. Следует помнить, что адреса векторов прерывания одних и тех же аппаратных узлов для разных типов AVR могут иметь разное значение. Поэтому для обеспечения переносимости программного обеспечения удобно, так же как и в случае с регистрами ввода/вывода, использовать символические имена адресов векторов прерывания, которые определены в соответствующем **inc**-файле.

В ячейках оперативной памяти организуется системный стек, который используется автоматически для хранения адресов возврата при выполнении подпрограмм, а также может использоваться программистом для временного хранения содержимого оперативных регистров (команды PUSH и POP). Стек растет от старших адресов к младшим, поэтому, учитывая, что начальное значение указателя стека после сброса равно нулю, программист AVR обязательно должен в инициализирующей части программы позаботиться об установке указателя стека, если он предполагает использовать хотя бы одну подпрограмму. Микроконтроллеры, не имеющие RAM (семейства Tiny), содержат трехуровневый аппаратный стек.

Система команд AVR представлена в табл. 1.2.

Операнды могут быть таких видов:

Rd: Результирующий (и исходный) регистр в регистровом файле;

Rr: Исходный регистр в регистровом файле;

b: Константа (3 бита), может быть константное выражение;

s: Константа (3 бита), может быть константное выражение;

P: Константа (5-6 бит), может быть константное выражение;

K6; Константа (6 бит), может быть константное выражение;

K8: Константа (8 бит), может быть константное выражение;

k: Константа (размер зависит от инструкции), может быть константное выражение;

q: Константа (6 бит), может быть константное выражение;

Rdl: R24, R26, R28, R30. Для инструкций ADIW и SBIW;

X,Y,Z: Регистры косвенной адресации (X=R27:R26, Y=R29:R28, Z=R31:R30).

Ассемблер не различает регистр символов.

Таблица 1.2 — Инструкции процессоров AVR

Арифметические и логические команды

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
ADD	Rd,Rr	Суммирование без переноса	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Суммирование с переносом	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
SUB	Rd,Rr	Вычитание без переноса	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Вычитание константы	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Вычитание с переносом	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Вычитание константы с переносом	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Логическое И	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Логическое И с константой	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Логическое ИЛИ	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Логическое ИЛИ с константой	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Логическое исключающее ИЛИ	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	Побитная Инверсия	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Изменение знака (Доп. код)	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Установить бит (биты) в регистре	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Сбросить бит (биты) в регистре	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Инкрементировать значение регистра	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Декрементировать значение регистра	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Проверка на ноль либо отрицательность	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Очистить регистр	Rd = 0	Z,C,N,V,S	1
SER	Rd	Установить регистр	Rd = $FF	None	1
ADIW	Rdl,K6	Сложить константу и слово	Rdh:Rdl = Rdh:Rdl + K6	Z,C,N,V,S	2

Продолжение табл. 1.2

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
SBIW	Rdl,K6	Вычесть константу из слова	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Умножение чисел без знака	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Умножение чисел со знаком	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Умножение числа со знаком с числом без знака	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Умножение дробных чисел без знака	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Умножение дробных чисел со знаком	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Умножение дробного числа со знаком с числом без знака	R1:R0 = (Rd * Rr) << 1	Z,C	2

Команды ветвления

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
RJMP	k	Относительный переход	PC = PC + k +1	None	2
IJMP	Нет	Косвенный переход на (Z)	PC = Z	None	2
EIJMP	Нет	Расширенный косвенный переход на (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
JMP	k	Переход	PC = k	None	3
RCALL	k	Относительный вызов под-программы	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	Нет	Косвенный вызов (Z)	STACK = PC+1, PC = Z	None	3/4*
EICALL	Нет	Расширенный косвенный вызов (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND	None	4*
CALL	k	Вызов подпрограммы	STACK = PC+2, PC = k	None	4/5*
RET	Нет	Возврат из подпрограммы	PC = STACK	None	4/5*
RETI	Нет	Возврат из прерывания	PC = STACK	I	4/5*

Продолжение табл. 1.2

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
CPSE	Rd,Rr	Сравнить, пропустить если равны	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Сравнить	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Сравнить с переносом	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Сравнить с константой	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Пропустить если бит в регистре очищен	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Пропустить если бит в регистре установлен	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Пропустить если бит в порту очищен	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Пропустить если бит в порту установлен	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Перейти если флаг в SREG очищен	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Перейти если флаг в SREG установлен	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Перейти если равно	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Перейти если не равно	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Перейти если перенос установлен	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Перейти если перенос очищен	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Перейти если равно или больше	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Перейти если меньше	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Перейти если минус	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Перейти если плюс	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Перейти если больше или равно (со знаком)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Перейти если меньше (со знаком)	if(S==1) PC = PC + k + 1	None	1/2

Продолжение табл. 1.2

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
BRHS	k	Перейти если флаг внутреннего переноса установлен	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Перейти если флаг внутреннего переноса очищен	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Перейти если флаг T установлен	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Перейти если флаг T очищен	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Перейти если флаг переполнения установлен	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Перейти если флаг переполнения очищен	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Перейти если прерывания разрешены	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Перейти если прерывания запрещены	if(I==0) PC = PC + k + 1	None	1/2

* Для операций доступа к данным количество циклов указано при условии доступа к внутренней памяти данных, и не корректно при работе с внешним ОЗУ. Для инструкций CALL, ICALL, EICALL, RCALL, RET и RETI необходимо добавить три цикла плюс по два цикла для каждого ожидания в контроллерах с PC, меньшим 16 бит (128KB памяти программ). Для устройств с памятью программ свыше 128KB добавьте пять циклов плюс по три цикла на каждое ожидание.

Команды передачи данных

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
MOV	Rd,Rr	Скопировать регистр	Rd = Rr	None	1
MOVW	Rd,Rr	Скопировать пару регистров	Rd+1:Rd = Rr+1:Rr	None	1
LDI	Rd,K8	Загрузить константу	Rd = K	None	1
LDS	Rd,k	Прямая загрузка	Rd = (k)	None	2*
LD	Rd,X	Косвенная загрузка	Rd = (X)	None	2*
LD	Rd,X+	Косвенная загрузка с пост-инкрементом	Rd = (X), X=X+1	None	2*

Продолжение табл. 1.2

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
LD	Rd,-X	Косвенная загрузка с пре-декрементом	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Косвенная загрузка	Rd = (Y)	None	2*
LD	Rd,Y+	Косвенная загрузка с пост-инкрементом	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Косвенная загрузка с пре-декрементом	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Косвенная загрузка с замещением	Rd = (Y+q)	None	2*
LD	Rd,Z	Косвенная загрузка	Rd = (Z)	None	2*
LD	Rd,Z+	Косвенная загрузка с пост-инкрементом	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Косвенная загрузка с пре-декрементом	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Косвенная загрузка с замещением	Rd = (Z+q)	None	2*
STS	k,Rr	Прямое сохранение	(k) = Rr	None	2*
ST	X,Rr	Косвенное сохранение	(X) = Rr	None	2*
ST	X+,Rr	Косвенное сохранение с пост-инкрементом	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Косвенное сохранение с пре-декрементом	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Косвенное сохранение	(Y) = Rr	None	2*
ST	Y+,Rr	Косвенное сохранение с пост-инкрементом	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Косвенное сохранение с пре-декрементом	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Косвенное сохранение с замещением	(Y+q) = Rr	None	2
ST	Z,Rr	Косвенное сохранение	(Z) = Rr	None	2
ST	Z+,Rr	Косвенное сохранение с пост-инкрементом	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Косвенное сохранение с пре-декрементом	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Косвенное сохранение с замещением	(Z+q) = Rr	None	2
LPM	Нет	Загрузка из программной памяти	R0 = (Z)	None	3
LPM	Rd,Z	Загрузка из программной памяти	Rd = (Z)	None	3
LPM	Rd,Z+	Загрузка из программной памяти с пост-инкрементом	Rd = (Z), Z=Z+1	None	3
ELPM	Нет	Расширенная загрузка из программной памяти	R0 = (RAMPZ:Z)	None	3
ELPM	Rd,Z	Расширенная загрузка из программной памяти	Rd = (RAMPZ:Z)	None	3

Продолжение табл. 1.2

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
ELPM	Rd,Z+	Расширенная загрузка из программной памяти с пост-инкрементом	Rd = (RAMPZ:Z), Z = Z+1	None	3
SPM	Нет	Сохранение в программной памяти	(Z) = R1:R0	None	-
ESPM	Нет	Расширенное сохранение в программной памяти	(RAMPZ:Z) = R1:R0	None	-
IN	Rd,P	Чтение порта	Rd = P	None	1
OUT	P,Rr	Запись в порт	P = Rr	None	1
PUSH	Rr	Занесение регистра в стек	STACK = Rr	None	2
POP	Rd	Извлечение регистра из стека	Rd = STACK	None	2

* Для операций доступа к данным количество циклов указано при условии доступа к внутренней памяти данных, и не корректно при работе с внешним ОЗУ. Для инструкций LD, ST, LDD, STD, LDS, STS, PUSH и POP необходимо добавить один цикл плюс по одному циклу для каждого ожидания.

Команды работы с битами

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
LSL	Rd	Логический сдвиг влево	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Логический сдвиг вправо	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Циклический сдвиг влево через C	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Циклический сдвиг вправо через C	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Арифметический сдвиг вправо	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Перестановка тетрад	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Установка флага	SREG(s) = 1	SREG(s)	1
BCLR	s	Очистка флага	SREG(s) = 0	SREG(s)	1
SBI	P,b	Установить бит в порту	I/O(P,b) = 1	None	2

Продолжение табл. 1.2

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
CBI	P,b	Очистить бит в порту	I/O(P,b) = 0	None	2
BST	Rr,b	Сохранить бит из регистра в T	T = Rr(b)	T	1
BLD	Rd,b	Загрузить бит из T в регистр	Rd(b) = T	None	1
SEC	Нет	Установить флаг переноса	C =1	C	1
CLC	Нет	Очистить флаг переноса	C = 0	C	1
SEN	Нет	Установить флаг отрицательного числа	N = 1	N	1
CLN	Нет	Очистить флаг отрицательного числа	N = 0	N	1
SEZ	Нет	Установить флаг нуля	Z = 1	Z	1
CLZ	Нет	Очистить флаг нуля	Z = 0	Z	1
SEI	Нет	Установить флаг прерываний	I = 1	I	1
CLI	Нет	Очистить флаг прерываний	I = 0	I	1
SES	Нет	Установить флаг числа со знаком	S = 1	S	1
CLS	Нет	Очистить флаг числа со знаком	S = 0	S	1
SEV	Нет	Установить флаг переполнения	V = 1	V	1
CLV	Нет	Очистить флаг переполнения	V = 0	V	1
SET	Нет	Установить флаг T	T = 1	T	1
CLT	Нет	Очистить флаг T	T = 0	T	1
SEH	Нет	Установить флаг внутреннего переноса	H = 1	H	1
CLH	Нет	Очистить флаг внутреннего переноса	H = 0	H	1
NOP	Нет	Нет операции	Нет	None	1
SLEEP	Нет	Спать (уменьшить энергопотребление)	Смотрите описание инструкции	None	1
WDR	Нет	Сброс сторожевого таймера	Смотрите описание инструкции	None	1

2 ДИРЕКТИВЫ АССЕМБЛЕРА

Компилятор поддерживает ряд директив. Директивы не транслируются непосредственно в код. Вместо этого они используются для указания положения в программной памяти, определения макросов, инициализации памяти и т.д. Все директивы предваряются точкой.

BYTE — Зарезервировать байты в ОЗУ

Директива BYTE резервирует байты в ОЗУ. Если Вы хотите иметь возможность ссылаться на выделенную область памяти, то директива BYTE должна быть предварена меткой. Директива принимает один обязательный параметр, который указывает количество выделяемых байт. Эта директива может использоваться только в сегменте данных (смотреть директивы CSEG и DSEG). Выделенные байты не инициализируются.

Синтаксис:
МЕТКА: .BYTE выражение
Пример:

```
.DSEG
var1:   .BYTE 1          ; резервирует 1 байт для var1
table:  .BYTE tab_size   ; резервирует tab_size байт
.CSEG
ldi r30,low(var1)        ; Загружает младший байт регистра Z
ldi r31,high(var1)       ; Загружает старший байт регистра Z
ld r1,Z                  ; Загружает VAR1 в регистр 1
```

CSEG — Программный сегмент

Директива CSEG определяет начало программного сегмента. Исходный файл может состоять из нескольких программных сегментов, которые объединяются в один программный сегмент при компиляции. Программный сегмент является сегментом по умолчанию. Программные сегменты имеют свои собственные счётчики положения, которые считают не побайтно, а пословно. Директива ORG может быть использована для размещения кода и констант в необходимом месте сегмента. Директива CSEG не имеет параметров.

Синтаксис:
.CSEG
Пример:
```
.DSEG                ; Начало сегмента данных
vartab: .BYTE 4      ; Резервирует 4 байта в ОЗУ
.CSEG                ; Начало кодового сегмента
const:  .DW 2        ; Разместить константу 0x0002 в памяти программ
      mov r1,r0      ; Выполнить действия
```

DB — Определить байты во FLASH или EEPROM

Директива DB резервирует необходимое количество байт в памяти программ или в EEPROM. Если Вы хотите иметь возможность ссылаться на выделенную область памяти, то директива DB должна быть предварена меткой. Директива DB должна иметь хотя бы один параметр. Данная директива может быть размещена только в сегменте программ (CSEG) или в сегменте EEPROM (ESEG).

Параметры, передаваемые директиве — это последовательность выражений, разделённых запятыми. Каждое выражение должно быть или числом в диапазоне (–128…255), или в результате вычисления должно давать результат в этом же диапазоне, в противном случае число усекается до байта, причём БЕЗ выдачи предупреждений.

Если директива получает более одного параметра и текущим является программный сегмент, то параметры упаковываются в слова (первый параметр — младший байт), и если число параметров нечётно, то последнее выражение будет усечено до байта и записано как слово со старшим байтом, равным нулю, даже если далее идет ещё одна директива DB.

Синтаксис:
МЕТКА: .DB список_выражений
Пример:
```
.CSEG
consts: .DB 0, 255, 0b01010101, -128, 0xaa
.ESEG
const2: .DB 1,2,3
```

DEF — Назначить регистру символическое имя

Директива DEF позволяет ссылаться на регистр через некоторое символическое имя. Назначенное имя может использоваться во всей нижеследующей части программы для обращений к данному регистру. Регистр может иметь несколько различных имен. Символическое имя может быть переназначено позднее в программе.

Синтаксис:
.DEF Символическое_имя = Регистр
Пример:

```
.DEF temp=R16
.DEF ior=R0
.CSEG
 ldi temp,0xf0  ; Загрузить 0xf0 в регистр temp (R16)
 in ior,0x3f    ; Прочитать SREG в регистр ior (R0)
 eor temp,ior   ; Регистры temp и ior складываются по исключающему ИЛИ
```

DEVICE — Определить устройство, для которого компилируется программа

Директива DEVICE позволяет указать, для какого устройства компилируется программа. При использовании данной директивы компилятор выдаст предупреждение, если будет найдена инструкция, которую не поддерживает данный микроконтроллер. Также будет выдано предупреждение, если программный сегмент, либо сегмент EEPROM превысят размер, допускаемый устройством. Если же директива не используется, то все инструкции считаются допустимыми, и отсутствуют ограничения на размер сегментов.

Синтаксис:
.DEVICE AT90S1200 |AT90S2313 | AT90S2323 | AT90S2333 | AT90S2343 | AT90S4414 | AT90S4433 | AT90S4434 | AT90S8515 | AT90S8534 | AT90S8535 | ATtiny11 | ATtiny12 | ATtiny22 | ATmega603 | ATmega103
Пример:

```
.DEVICE AT90S1200  ; Используется AT90S1200
.CSEG
      push r30       ; Эта инструкция вызовет предупреждение
                     ; поскольку AT90S1200 её не имеет
```

DSEG — Сегмент данных

Директива DSEG определяет начало сегмента данных. Исходный файл может состоять из нескольких сегментов данных, которые объединяются в один сегмент при компиляции. Сегмент данных обычно состоит только из директив BYTE и меток. Сегменты данных имеют свои собственные побайтные счётчики положения. Директива ORG может быть использована для размещения переменных в необходимом месте ОЗУ. Директива не имеет параметров.

Синтаксис:

```
.DSEG
```

Пример:

```
 .DSEG                ; Начало сегмента данных
var1:  .BYTE 1        ; зарезервировать 1 байт для var1
table:  .BYTE tab_size    ; зарезервировать tab_size байт.
.CSEG
    ldi r30,low(var1)    ; Загрузить младший байт регистра Z
    ldi r31,high(var1)   ; Загрузить старший байт регистра Z
    ld r1,Z              ; Загрузить var1 в регистр r1
```

DW — Определить слова во FLASH или EEPROM

Директива DW резервирует необходимое количество слов в памяти программ или в EEPROM. Если Вы хотите иметь возможность ссылаться на выделенную область памяти, то директива DW должна быть предварена меткой. Директива DW должна иметь хотя бы один параметр. Данная директива может быть размещена только в сегменте программ (CSEG) или в сегменте EEPROM (ESEG).

Параметры, передаваемые директиве — это последовательность выражений, разделённых запятыми. Каждое выражение должно быть или числом в диапазоне (–32768…65535), или в результате вычисления должно давать результат в этом же диапазоне, в противном случае число усекается до слова, причем БЕЗ выдачи предупреждений.

```
Синтаксис:
МЕТКА:      .DW expressionlist
                Пример:
                .CSEG
varlist:        .DW 0, 0xffff, 0b1001110001010101, -32768, 65535
                .ESEG
        eevarlst: .DW 0,0xffff,10
```

EQU — Установить постоянное выражение

Директива EQU присваивает метке значение. Эта метка может позднее использоваться в выражениях. Метка, которой присвоено значение данной директивой, не может быть переназначена и её значение не может быть изменено.

```
Синтаксис:
.EQU метка = выражение
 Пример:
.EQU io_offset = 0x23
 .EQU porta     = io_offset + 2
.CSEG                   ; Начало сегмента команд
clr r2                  ; Очистить регистр r2
out porta,r2            ; Записать в порт A
```

ESEG — Сегмент EEPROM

Директива ESEG определяет начало сегмента EEPROM. Исходный файл может состоять из нескольких сегментов EEPROM, которые объединяются в один сегмент при компиляции. Сегмент EEPROM обычно состоит только из директив DB, DW и меток. Сегменты EEPROM имеют свои собственные побайтные счётчики положения. Директива ORG может быть использована для размещения переменных в необходимом месте EEPROM. Директива не имеет параметров.

```
Синтаксис:
.ESEG
Пример:
 .DSEG                  ; Начало сегмента данных
var1:   .BYTE 1         ; зарезервировать 1 байт для var1
table:  .BYTE tab_size  ; зарезервировать tab_size байт.
.ESEG
eevar1: .DW 0xffff      ; проинициализировать 1 слово в EEPROM
```

EXIT — Выйти из файла

Встретив директиву EXIT, компилятор прекращает компиляцию данного файла. Если директива использована во вложенном файле (см. директиву INCLUDE), то компиляция продолжается со строки, следующей после директивы INCLUDE. Если же файл не является вложенным, то компиляция прекращается.

Синтаксис:
.EXIT
Пример:
```
.EXIT           ; Выйти из данного файла
```

INCLUDE — Вложить другой файл

Встретив директиву INCLUDE, компилятор открывает указанный в ней файл, компилирует его, пока файл не закончится или не встретится директива EXIT, после этого продолжает компиляцию начального файла со строки, следующей за директивой INCLUDE. Вложенный файл может также содержать директивы INCLUDE.

Синтаксис:
.INCLUDE "имя_файла"
Пример:
```
                          ; файл iodefs.asm:
.EQU sreg   = 0x3f         ; Регистр статуса
.EQU sphigh = 0x3e         ; Старший байт указателя стека
.EQU splow  = 0x3d         ; Младший байт указателя стека
                          ; файл incdemo.asm
.INCLUDE iodefs.asm     ; Вложить определения портов
 in r0,sreg                ; Прочитать регистр статуса
```

LIST — Включить генерацию листинга

Директива LIST указывает компилятору на необходимость создания листинга. Листинг представляет собой комбинацию ассемблерного кода, адресов и кодов операций. По умолчанию генерация листинга включена, однако данная директива используется совместно с директивой NOLIST для получения листингов отдельных частей исходных файлов.

Синтаксис:
.LIST

Пример:

```
.NOLIST                    ; Отключить генерацию листинга
.INCLUDE "macro.inc"       ; Вложенные файлы не будут
.INCLUDE "const.def"       ; отображены в листинге
.LIST                      ; Включить генерацию листинга
```

NOLIST — Выключить генерацию листинга

Директива NOLIST указывает компилятору на необходимость прекращения генерации листинга. Листинг представляет собой комбинацию ассемблерного кода, адресов и кодов операций. По умолчанию генерация листинга включена, однако может быть отключена данной директивой. Кроме того, данная директива может быть использована совместно с директивой LIST для получения листингов отдельных частей исходных файлов

Синтаксис:
.NOLIST

Пример:

```
.NOLIST                    ; Отключить генерацию листинга
.INCLUDE "macro.inc"       ; Вложенные файлы не будут
.INCLUDE "const.def"       ; отображены в листинге
.LIST                      ; Включить генерацию листинга
```

ORG — Установить положение в сегменте

Директива ORG устанавливает счётчик положения равным заданной величине, которая передаётся как параметр. Для сегмента данных она устанавливает счётчик положения в SRAM (ОЗУ), для сегмента программ это программный счётчик, а для сегмента EEPROM это положение в EEPROM. Если директиве предшествует метка (в той же строке), то метка размещается по адресу, указанному в параметре директивы. Перед началом компиляции программный счётчик и счётчик EEPROM равны нулю, а счётчик ОЗУ равен 32 (поскольку адреса 0-31 заняты регистрами). Обратите внимание, что для ОЗУ и EEPROM используются побайтные счётчики, а для программного сегмента — пословный.

Синтаксис:
.ORG выражение
Пример:

```
.DSEG                ; Начало сегмента данных
.ORG 0x37            ; Установить адрес SRAM равным 0x37
variable: .BYTE 1    ; Зарезервировать байт по адресу 0x37
.CSEG
.ORG 0x10            ; Установить программный счётчик равным 0x10
 mov r0,r1           ; Данная команда будет размещена по адресу 0x10
```

SET — Установить переменный символический эквивалент выражения

Директива SET присваивает имени некоторое значение. Это имя позднее может быть использовано в выражениях. Причем в отличие от директивы EQU значение имени может быть изменено другой директивой SET.

Синтаксис:
.SET имя = выражение
Пример:

```
.SET io_offset = 0x23
.SET porta     = io_offset + 2
.CSEG                ; Начало кодового сегмента
clr r2               ; Очистить регистр 2
out porta,r2         ; Записать в порт A
```

Форматы представления чисел

- Десятичный (принят по умолчанию): 10, 255.
- Шестнадцатеричный (два варианта записи): 0x0a, $0a, 0xff, $ff .
- Двоичный: 0b00001010, 0b11111111.
- Восьмеричный (начинаются с нуля): 010, 077.

3 ПРОГРАММНЫЙ ПАКЕТ AVR Studio

Популярность микроконтроллеров AVR способствовала тому, что многие фирмы-производители программных средств поддержки микроконтроллеров (ассемблеров, компиляторов, отладчиков) создали программные пакеты поддержки AVR. Данный раздел знакомит с основным программным пакетом – AVR Studio, предлагаемым самой фирмой Atmel.

AVR Studio – это интегрированная отладочная среда разработки приложений (IDE) для микроконтроллеров семейства AVR (AT90S, ATmega, ATtiny) фирмы Atmel.

IDE AVR Studio содержит:

- транслятор языка ассемблера (Atmel AVR macroassembler);
- отладчик (Debugger);
- программное обеспечение верхнего уровня для поддержки внутрисхемного программирования (In-System Programming, ISP).

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.

После запуска AVR Studio для создания нового проекта необходимо в меню **Project** выбрать команду **New Project**. В результате на экране появляется диалоговое окно (рис. 3.1), в котором необходимо ввести название проекта (**Project name**) и его расположение (**Location**). Новый проект удобнее создавать в отдельной папке.

Далее выбирается **AVR Simulator.** В левом окне выбирается тип микроконтроллера. После нажатия кнопки **Finish** на экране появляется окно организации проекта (рис. 3.2), показывающее все связанные с проектом файлы и окно для редактирования программы.

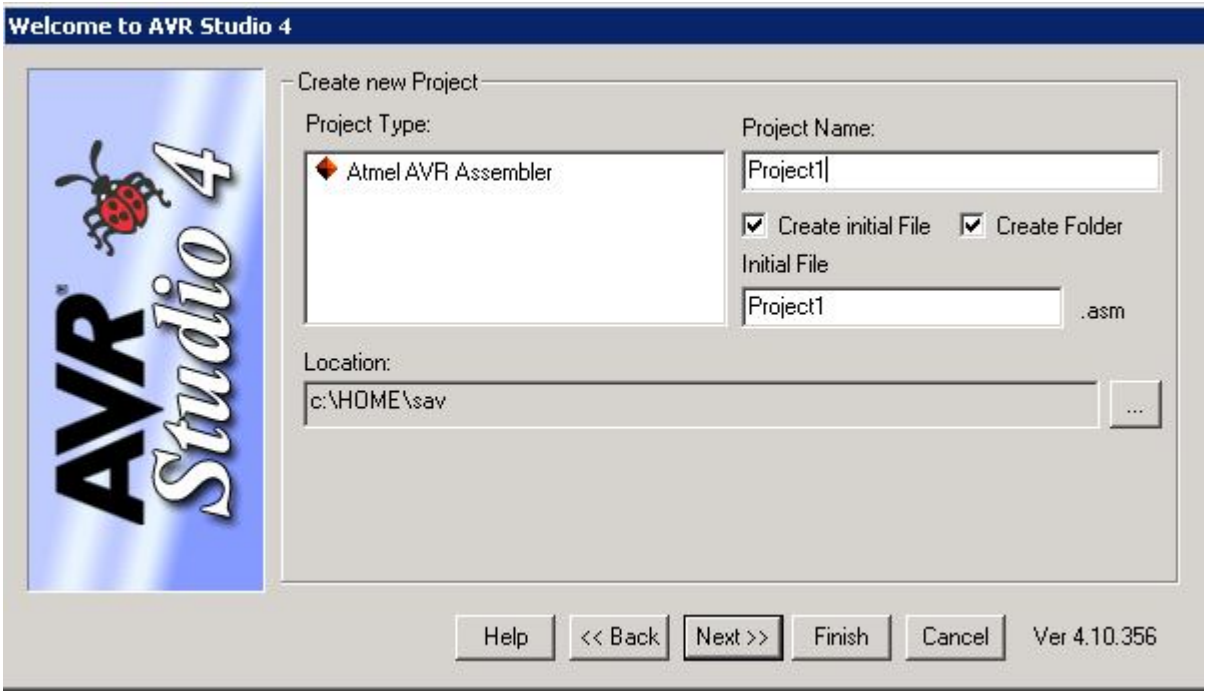


Рис. 3.1 – Окно создания нового проекта

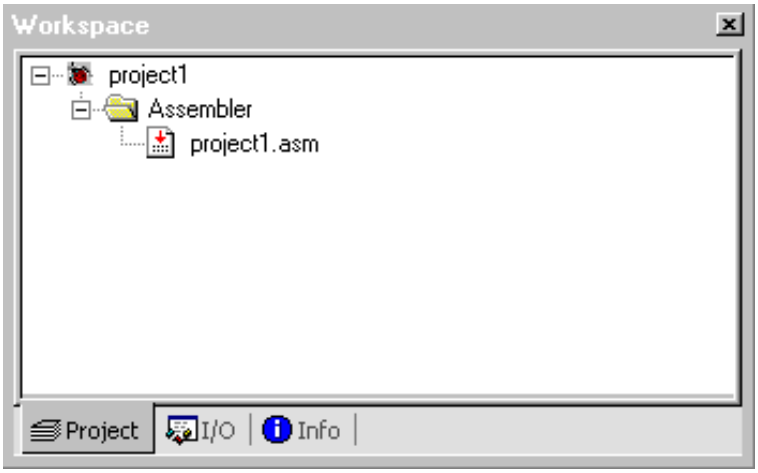


Рис. 3.2 – Окно организации проекта

В это окно для редактирования файла можно с клавиатуры ввести текст программы на языке ассемблера или открыть уже существующий файл (пункт Add Fail в меню Project) (рис.3.3).

Еще один способ создания проекта: в меню Fail выбираем Open Fail, подключаем уже созданный файл программы. Далее можно транслировать программу.

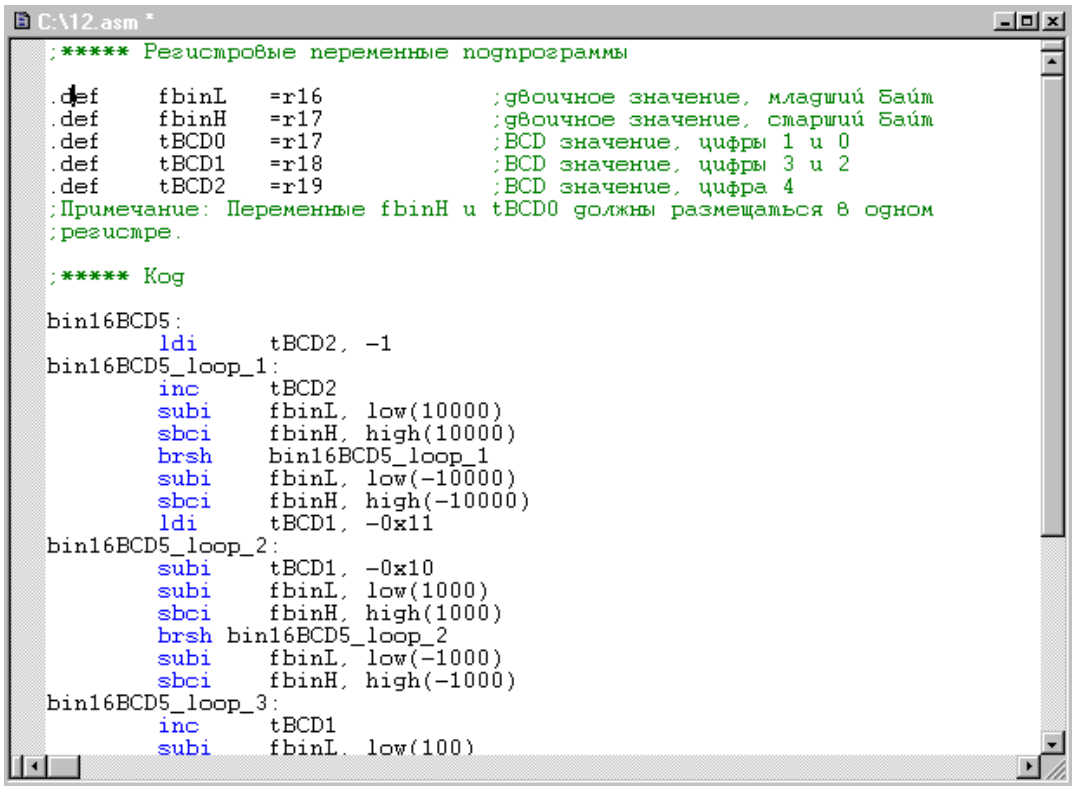


Рис. 3.3 – Окно редактирования программы на языке ассемблера

Созданный (или найденный) таким образом файл будет помещен в группу **Assembler** в окне организации проекта. Подобным же образом можно подключить к проекту и другие ассемблерные файлы, но группа **Assembler** может содержать только один файл, с которого в дальнейшем будет начинаться трансляция проекта. Назовем этот файл входным ассемблерным файлом проекта. Значок этого файла в окне организатора проекта отмечен красной стрелкой вправо, все другие файлы проекта будут отмечены синими стрелками, направленными вниз. Все файлы проекта должны быть включены во входной файл проекта с помощью ассемблерной директивы .include. Для смены входного файла проекта на другой надо установить курсор мыши на нужный файл в окне организации проекта и щелкнуть правой кнопкой мыши. В открывшемся всплывающем окне надо указать этот файл как **Assembler entry file**.

Для осуществления трансляции программы и проверки правильности её написания выбирается пункт **Build** (кнопка F7) в меню **Project**. Окно **View Output** содержит сообщения ассембле-

ра. В это окно выводится информация о количестве слов кода и данных, о наличии ошибок, и другая информация (рис. 3.4).

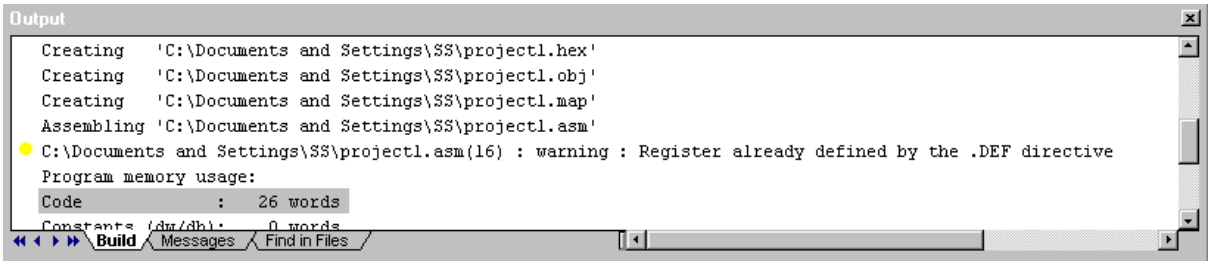


Рис. 3.4 – Окно сообщений ассемблера

Для локализации ошибок трансляции в случае их наличия можно в окне сообщений ассемблера установить курсор мыши на сообщение об ошибке и два раза щелкнуть левой кнопкой мыши. При этом в окне редактирования исходного текста программы курсор будет установлен на строку, вызвавшую сообщение об ошибке, и эта строка будет выделена цветом.

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.

Для запуска отладчика необходимо выполнить процедуру **Build and Run,** которая вызывается при нажатии на соответствующую кнопку (F7+Ctrl) на панели управления. Процедура **Build and Run** выполняется в два этапа. Сперва происходит трансляция входного ассемблерного файла, при которой независимо от установок проекта, кроме выходного файла заданного формата генерируется и объектный файл. Затем этот объектный файл загружается в отладчик.

Экран AVR Studio в режиме отладки представлен на рис. 3.5.

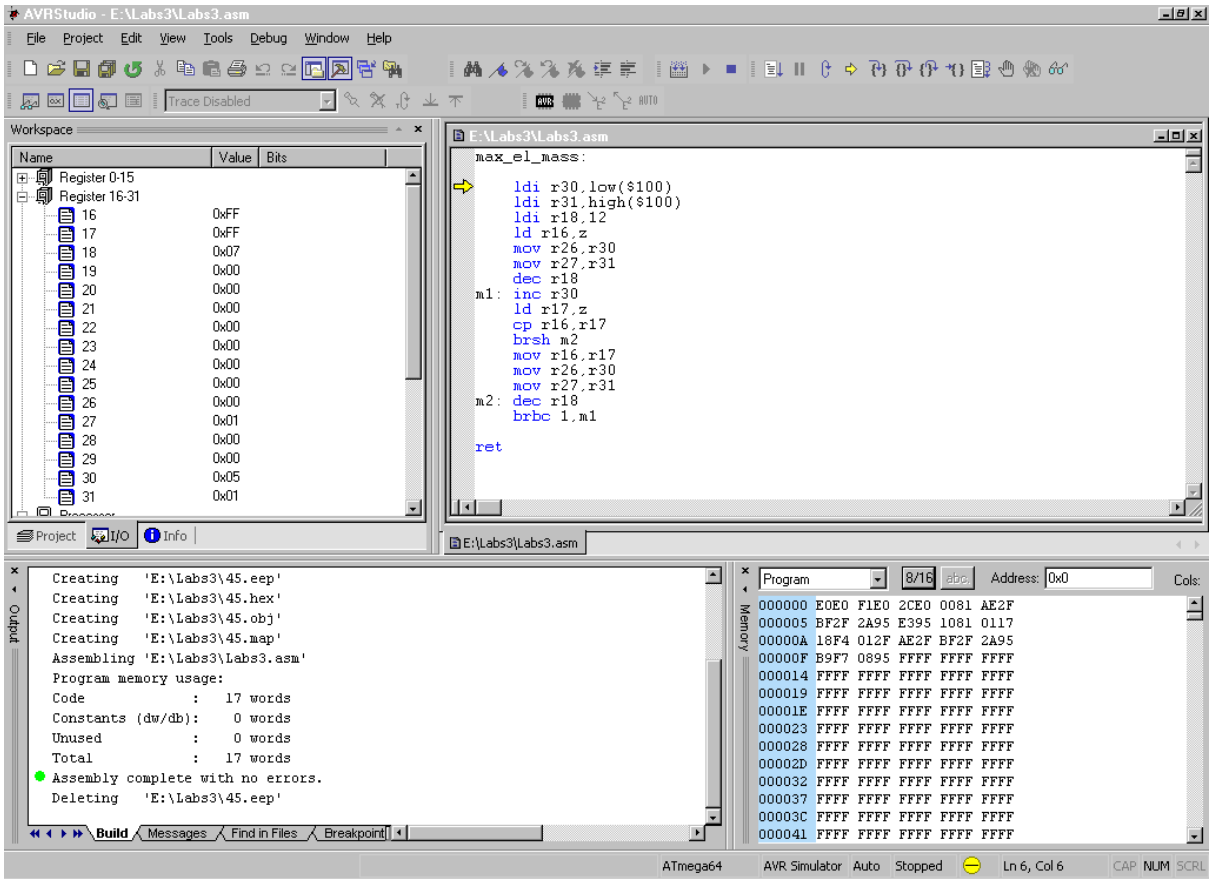


Рис. 3.5 – Экран AVR Studio в режиме отладки

При выполнении процедуры **Build and run** (или при загрузке объектного файла) автоматически открывается окно исходного текста исполняемой микроконтроллером программы.

После выполнения процедуры появляется желтая стрелка, указывающая позицию программного счетчика микроконтроллера (рис. 3.6). Этот указатель всегда находится на строке, которая будет выполнена в следующем цикле.

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 **Step Into**. Разница между ними в том, что команда Step Over не работает в подпрограммах. С помощью команд пошагового режима можно просле-

дить изменения значений в регистрах устройств ввода/вывода, памяти и регистрового файла. К командам шагового режима относятся также **Auto Step** и **Multi Step**. Помимо шагового режима, возможна отладка программы с использованием точек останова (**Breakpoints**). Командой Go запускается исполнение программы. Программа будет выполняться до остановки пользователем или до обнаружения точки останова.

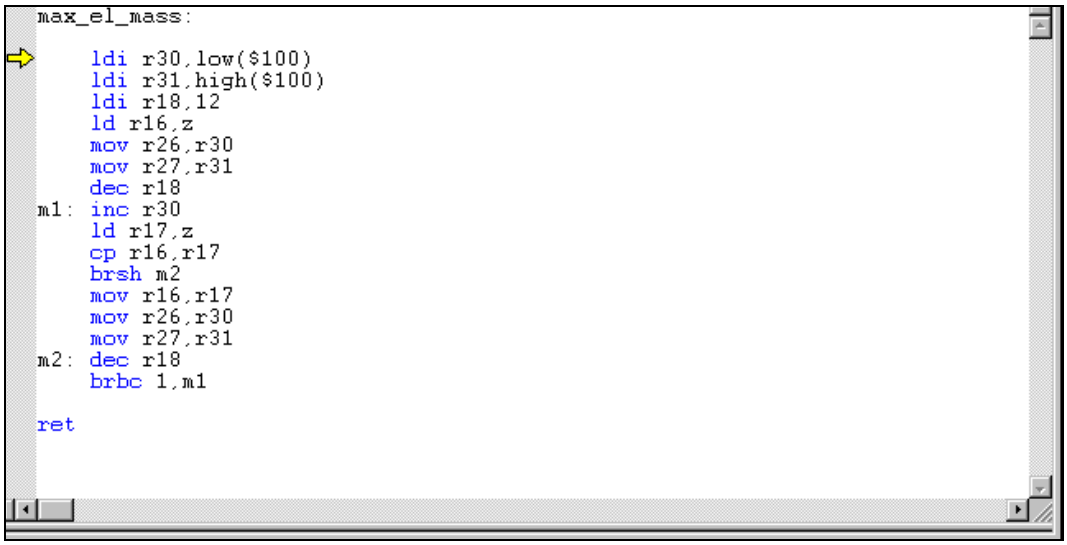


Рис. 3.6 – Окно исходного текста программы в режиме отладки

Для установки точки останова в AVR Studio служит пункт меню **Debug -> Toggle Breakpoint**. Точка останова ставится в строке, отмеченной курсором (рис. 3.7).

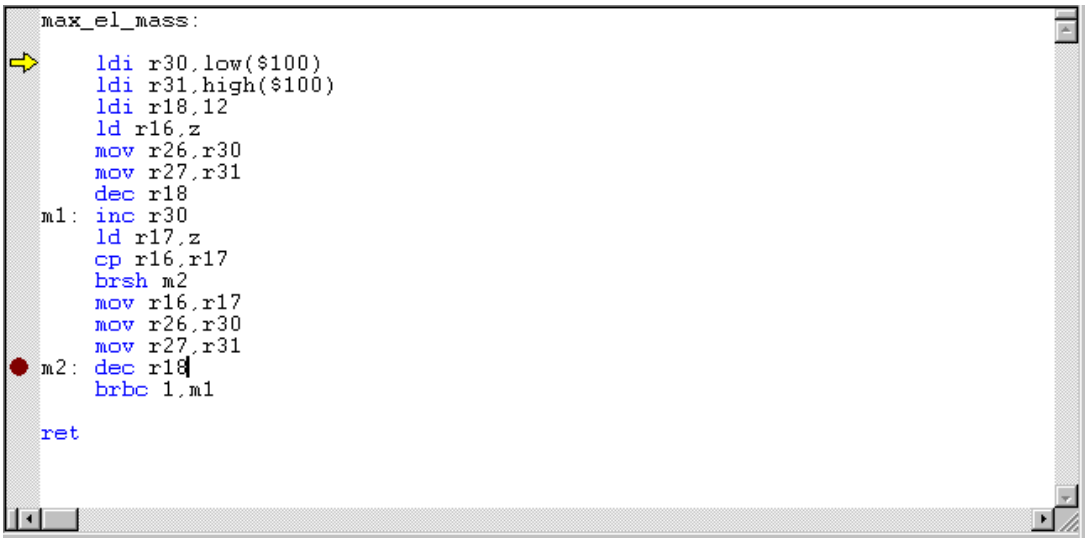


Рис. 3.7 – Точка останова в окне исходного текста программы в режиме отладки

Красная отметка в левом поле окна исходного текста программы показывает установленную точку останова.

В процессе отладки также можно выбрать пункт меню **Debug -> Run To Cursor (Ctrl+F10)**. При выборе этого пункта исполняемый код выполняется до достижения команды, обозначенной курсором. При этом, если отладчик обнаруживает точку останова, установленную ранее положения курсора, то останов будет выполнен только в случае его разрешения в окне **Debug Option**, в противном случае выполнение не приостанавливается. Если команда, обозначенная курсором, не достигается, отладчик продолжает исполнять код программы до тех пор, пока исполнение не будет прервано пользователем. Поскольку режим **Run To Cursor** зависит от позиции курсора, он доступен только при активном окне исходного текста.

Для остановки исполнения программы пользователем служит команда **Break (Ctrl+F5)**. В состоянии останова эта команда недоступна. При отладке с использованием точек останова, или если адрес останова указан курсором в окне исходного текста, модификация информации во всех окнах происходит только при достижении останова (или при прекращении исполнения программы пользователем).

Пункт меню **Debug -> Reset** (Shift+F5) выполняет сброс микроконтроллера. Если программа при этом выполняется, то ее исполнение будет остановлено. После сброса информация во всех окнах модифицируется.

Для наблюдения за работой программы можно открыть несколько окон, отображающих состояние различных узлов микроконтроллера. Окна открываются нажатием соответствующих кнопок на панели инструментов или при выборе соответствующего пункта меню **View**.

Регистровый файл микроконтроллера AVR отображается в окне **Work space** (вкладка I/O, рис. 3.8), а также можно открыть отдельное окно **Registers** (рис. 3.9). 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.

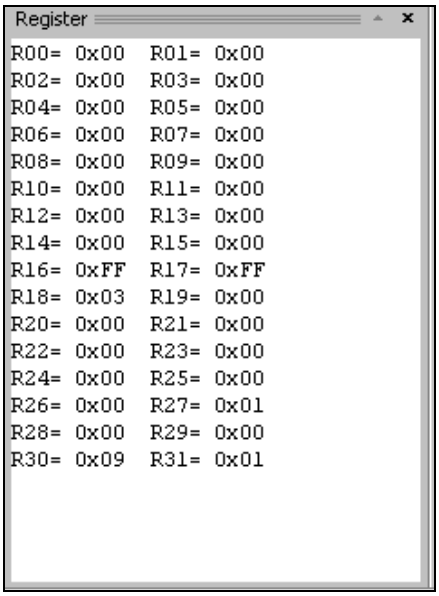


Рис. 3.8 – Окно состояния регистрового файла

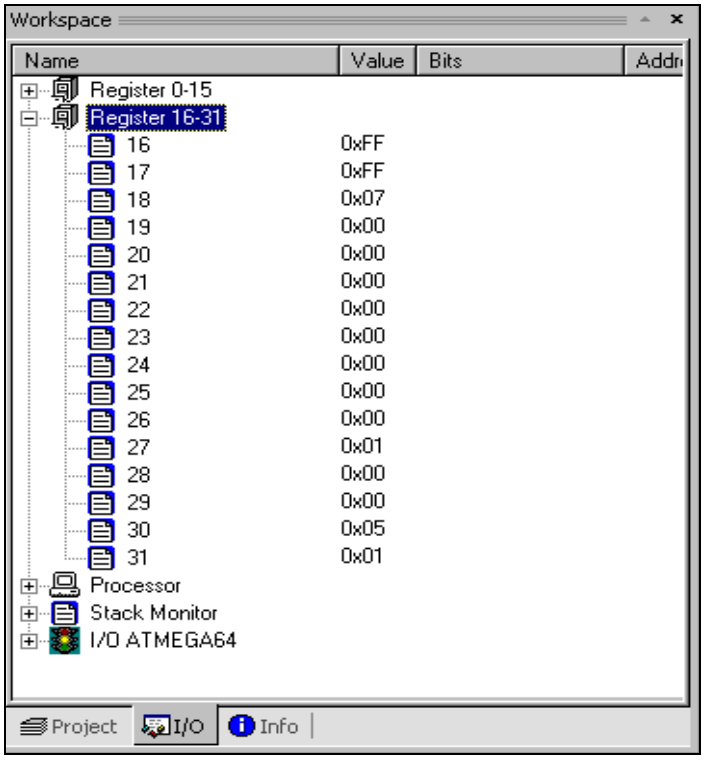


Рис. 3.9 – Окно I/O WorkSpace

Также в окне I/O WorkSpace отображается состояние встроенных периферийных устройств микроконтроллера (рис. 3.10).

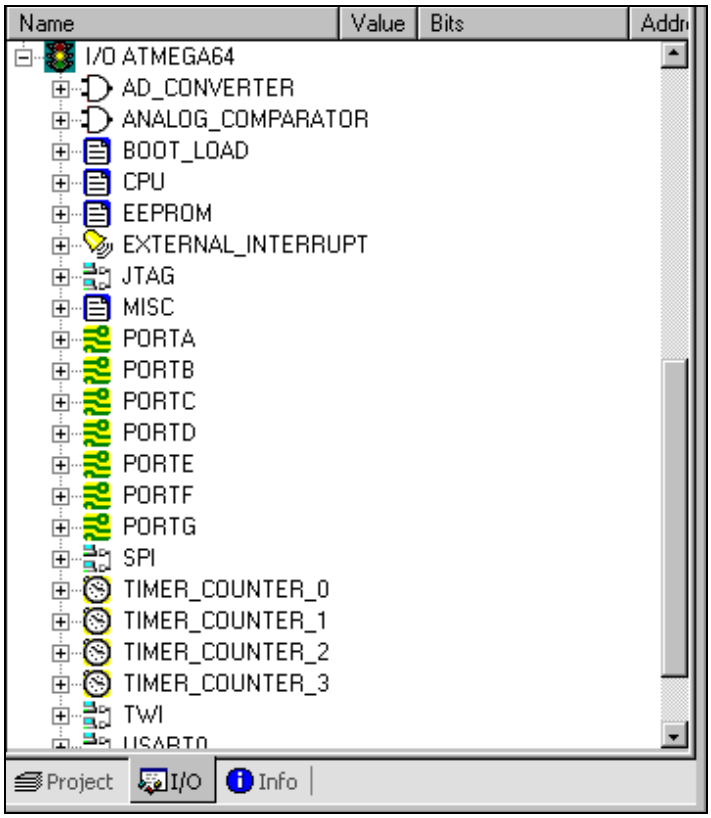


Рис. 3.10 – Окно состояния устройств ввода/вывода

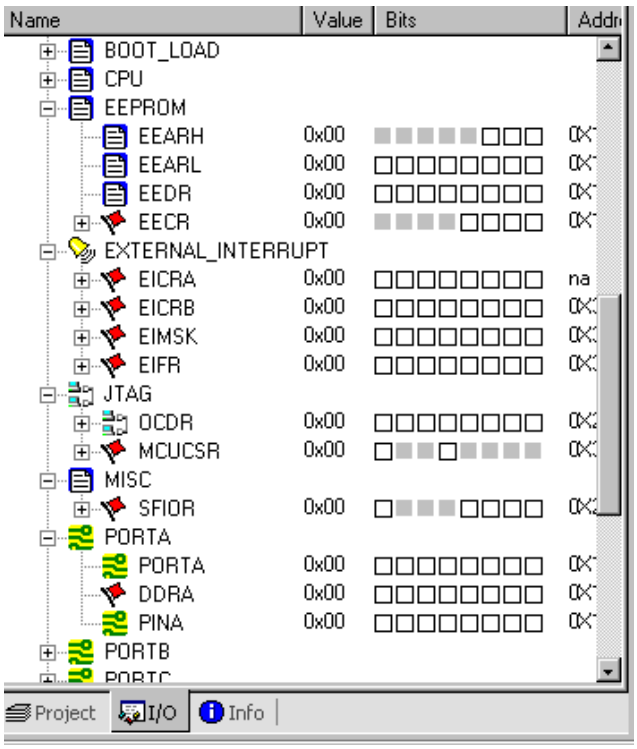


Рис. 3.11 – Развернутый порт PORTA и отображение регистров EEPROM, JTAG, MISC

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 (рис. 3.11). Каждый доступный для модификации бит может быть установлен или сброшен как программой по ходу ее исполнения, так и пользователем вручную (указав курсором нужный бит и щелкнув левой кнопкой мыши пользователь может изменить значение бита на обратное) – в режиме программной симуляции это является способом имитации входного воздействия на микроконтроллер.

Для индикации состояния программного счетчика, указателя стека, содержимого регистра статуса SREG и индексных регистров X, Y и Z в процессе отладки программы предназначена вкладка **Processor** в окне I/O WorkSpace (рис. 3.12).

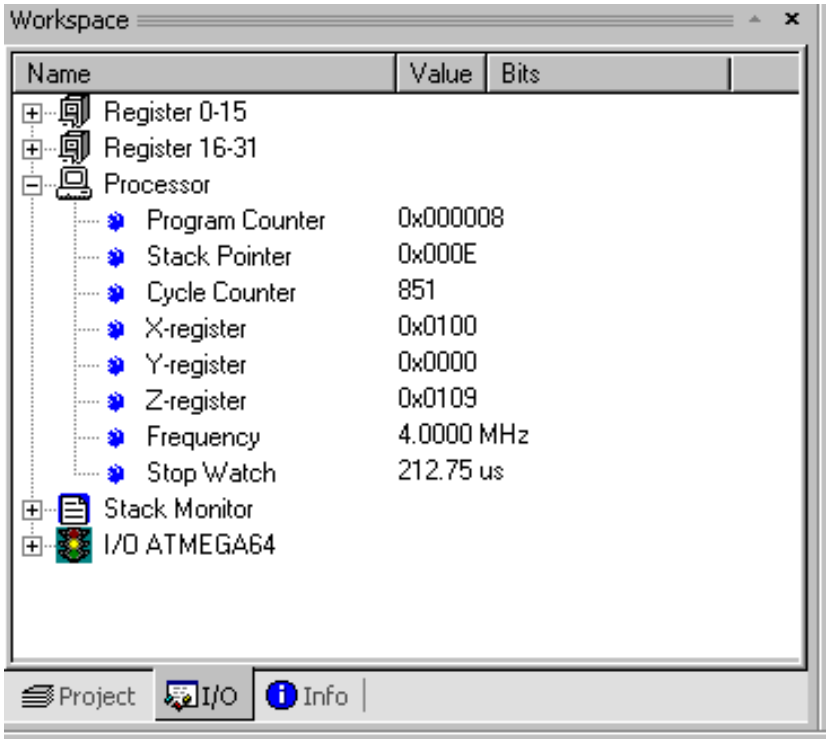


Рис. 3.12 – Окно состояния процессорного ядра

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

Просмотр ячеек памяти программ, памяти данных, EEPROM и регистров портов ввода/вывода в ходе исполнения программы осуществляется также с помощью диалогового окна **Memory**.

Падающее меню диалогового 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 (рис. 3.13).

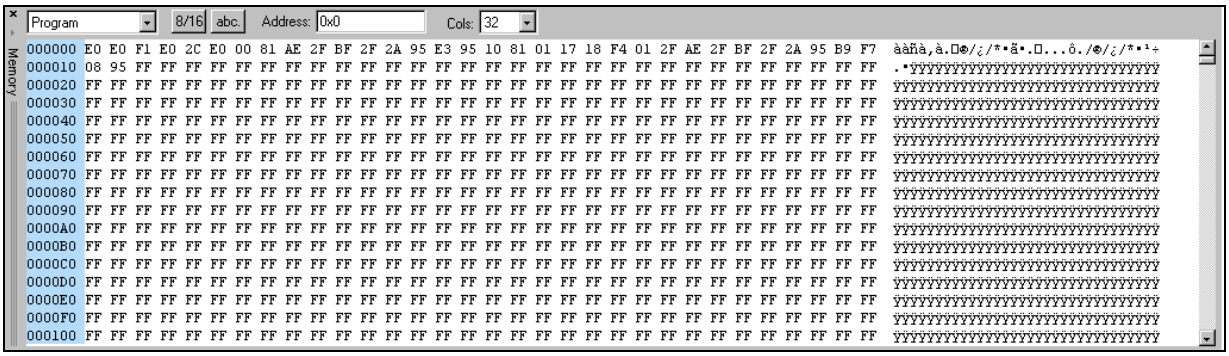


Рис. 3.13 – Окно просмотра содержимого памяти

Чтобы внести изменения в ячейке памяти достаточно дважды щелкнуть мышкой по данной ячейке.

Для внесения изменений в программу в процессе отладки необходимо редактировать её исходный текст. При попытке запуска симулятора на исполнение программы после редактирования на экране появляется окно, сообщающее об изменении программы и необходимости её компиляции.

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

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 электропотребления (инструкция «sleep» интерпретируется программным симулятором как «nop»).

4 МИКРОКОНТРОЛЛЕР ATtiny15L (лабораторная работа №3)

Цель работы. Целью лабораторной работы является отладка прикладных программ для микроконтроллера AVR семейства Tiny с помощью персонального компьютера и программных средств отладки.

ATtiny15L является 8-разрядным микроконтроллером с низким уровнем энергопотребления, основанным на AVR RISC архитектуре. Благодаря выполнению высокопроизводительных инструкций за один период тактового сигнала, ATtiny15L достигает производительности, приближающейся к уровню 1 MIPS на МГц, обеспечивая разработчику возможность оптимизировать уровень энергопотребления в соответствии с необходимым быстродействием. Ядро AVR содержит мощный набор инструкций (90 команд). Все 32 регистра общего назначения напрямую подключены к арифметико-логическому устройству (АЛУ), что обеспечивает доступ к двум независимым регистрам при выполнении одной инструкции за один такт. Данная архитектура позволяет повысить быстродействие вплоть до 10 раз по сравнению со стандартными микроконтроллерами CISC (рис. 4.1).

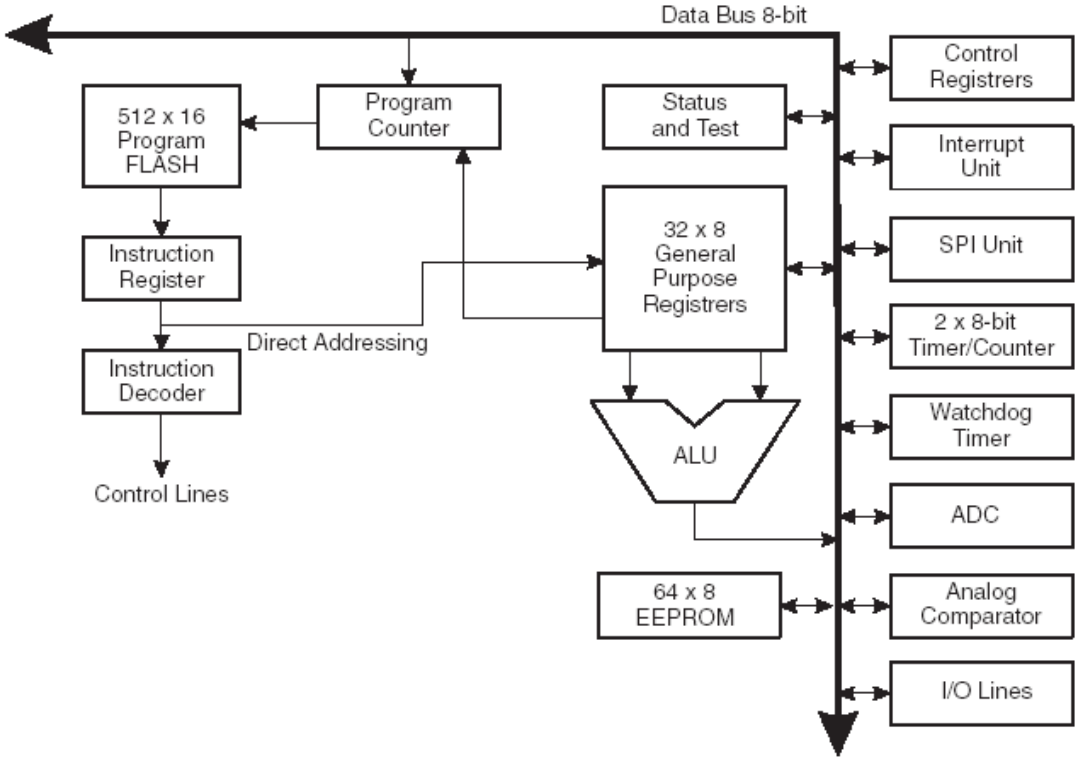


Рис. 4.1 – Структурная схема ATtiny15L

ATtiny15L имеет: 1 Кбайт Flash-памяти программ (512 16-разрядных ячеек), 64 байта энергонезависимой памяти данных EEPROM, 6 линий I/O общего назначения, 32 регистра общего назначения, два 8-разрядных универсальных таймера/счетчика, один с высокоскоростным выходом с ШИМ, встроенные генераторы, внутренние и внешние прерывания, программируемый сторожевой таймер, аналоговый компаратор, 4-канальный 10-разрядный АЦП, а также три программно выбираемых режима экономии энергопотребления. Режим ожидания «Idle Mode» останавливает CPU, но позволяет функционировать АЦП, аналоговому компаратору, таймеру/счетчикам и системе прерываний. Режим подавления шумов АЦП обеспечивает высокопрецизионные АЦП-измерения путем остановки CPU и сохранения работоспособности АЦП. Режим экономии энергопотребления «Power Down» сохраняет содержимое регистров, но останавливает тактовые генераторы, отключая все остальные функции микроконтроллера, вплоть до следующего внешнего прерывания, или до аппаратной инициализации. Функция активации, или прерывания при смене логического уровня на линии порта, позволяет ATtiny15L быть высокочувствительным к внешним событиям, при сохранении минимального уровня энергопотребления при нахождении в режимах экономии энергопотребления.

Устройство производится с применением технологии энергонезависимой памяти с высокой плотностью размещения, разработанной в корпорации Atmel. Благодаря совмещению усовершенствованного 8-разрядного RISC CPU с Flash-памятью с поддержкой внутрисистемного программирования на одном кристалле получился высокопроизводительный микроконтроллер ATtiny15L, обеспечивающий гибкое и экономически высокоэффективное решение для многих приложений встраиваемых систем управления, особенно в случае применения в зарядных устройствах, системах балластного освещения, и во всех типах приложений, использующих интеллектуальные датчики.

Напряжение питания от 2,7 В до 5,5 В. Внутренняя тактовая частота 1,6 МГц. Коммерческий и индустриальный диапазоны эксплуатационных температур. Корпус имеет 8 выводов (рис. 4.2). Альтернативные функции линий порта B указаны в табл. 4.1.

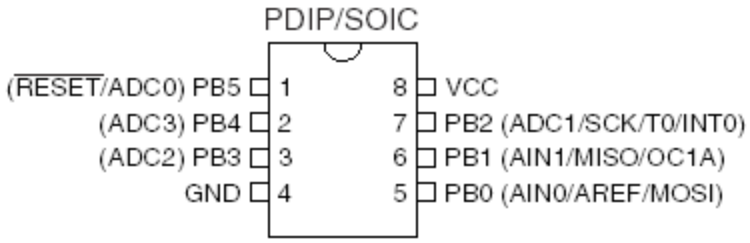


Рис. 4.2 – Расположение выводов ATtiny15L

Таблица 4.1 – Дополнительные функции линий порта B

Линия PB	Альтернативная функция
PB0	MOSI (Вход данных при программировании) AREF (Вход опорного напряжения для АЦП) AIN0 (Неинвертирующий вход аналогового компаратора)
PB1	MISO (Выход данных при программировании) OC1A (Выход таймера/счетчика T1 в режиме ШИМ) AIN1 (Инвертирующий вход аналогового компаратора)
PB2	SCK (Вход тактового сигнала при программировании) INT0 (Вход внешнего прерывания) ADC1 (Вход АЦП) T0 (Вход внешнего тактового сигнала таймера/счетчика T0)
PB3	ADC2 (Вход АЦП)
PB4	ADC3 (Вход АЦП)
PB5	RESET (Вход сброса) ADC0 (Вход АЦП)

Обращение к порту B производится с помощью регистров PORTB (регистр данных порта B), DDRB (регистр направления порта B), PINB (регистр выводов порта B). Формат этих регистров приведен в табл. 4.2.

Таблица 4.2 – Формат регистров порта B

7	6	5	4	3	2	1	0
–	–	–	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
–	–	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

Максимальная нагрузочная способность выводов PB4…PB0 составляет 20 мА, а вывода PB5 – 12 мА. Вывод PB5 может функ-

ционировать либо как вход, либо как выход с открытым стоком (на выводе может присутствовать только сигнал логического 0).

При установке разряда DDBn в 1 соответствующий n-вывод порта является выходом, при сбросе в 0 – входом. Для линий, сконфигурированных как входные, имеется возможность подключения внутренних подтягивающих резисторов сопротивлением 35…120 кОм между входом и шиной питания Vcc. Если разряд PUD (6-ой бит регистра MCUCR) установлен в 1, подтягивающие резисторы отключаются от всех линий порта. Для подключения подтягивающего резистора необходимо сбросить разряд PUD и записать 1 в соответствующий разряд регистра PORTB (вывод PB5 не имеет внутреннего подтягивающего резистора).

Полный перечень регистров ввода/вывода микроконтроллера ATtiny15L приведен в табл. 4.3. Младшие адреса памяти программ отведены под таблицу векторов прерываний (табл. 4. 4).

Таблица 4.3 – Регистры ввода/вывода (адрес, название и функция)

$3F	SREG	Status Register
$3B	GIMSK	General Interrupt Mask Register
$3A	GIFR	General Interrupt Flag Register
$39	TIMSK	Timer/Counter Interrupt Mask Register
$38	TIFR	Timer/Counter Interrupt Flag Register
$35	MCUCR	MCU Control Register
$34	MCUSR	MCU Status Register
$33	TCCR0	Timer/Counter0 Control Register
$32	TCNT0	Timer/Counter0 (8-bit)
$31	OSCCAL	Oscillator Calibration Register
$30	TCCR1	Timer/Counter1 Control Register
$2F	TCNT1	Timer/Counter1 (8-bit)
$2E	OCR1A	Timer/Counter1 Output Compare Register A
$2D	OCR1B	Timer/Counter1 Output Compare Register B
$2C	SFIOR	Special Function I/O Register
$21	WDTCR	Watchdog Timer Control Register
$1E	EEAR	EEPROM Address Register
$1D	EEDR	EEPROM Data Register
$1C	EECR	EEPROM Control Register
$18	PORTB	Data Register, Port B
$17	DDRB	Data Direction Register, Port B
$16	PINB	Input Pins, Port B
$08	ACSR	Analog Comparator Control and Status Register

Окончание табл. 4.3

$07	ADMUX	ADC Multiplexer Select Register
$06	ADCSR	ADC Control and Status Register
$05	ADCH	ADC Data Register High
$04	ADCL	ADC Data Register Low

Таблица 4.4 – Векторы прерываний (адрес, источник и описание)

$000	RESET	External Reset, Power-on Reset, Brown-out Reset, and Watchdog Reset
$001	INT0	External Interrupt Request 0
$002	I/O Pins	Pin Change Interrupt
$003	TIMER1, COMPA	Timer/Counter1 Compare Match A
$004	TIMER1, OVF	Timer/Counter1 Overflow
$005	TIMER0, OVF	Timer/Counter0 Overflow
$006	EE_RDY	EEPROM Ready
$007	ANA_COMP	Analog Comparator
$008	ADC	ADC Conversion Complete

Таймеры ATtiny15L

Таймер/счетчик T0 (счетный регистр TCNT0) может работать в режиме таймера (на вход поступают импульсы тактового сигнала микроконтроллера непосредственно или через предделитель) или в режиме счетчика внешних событий. Режим работы задают три младших разряда регистра управления TCCR0 (табл. 4.5). Другие разряды этого регистра не используются.

Таблица 4.5 – Выбор источника тактового сигнала для T0

CS02	CS01	CS00	Источник тактового сигнала
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 T0, инкремент счетчика производится по спадающему фронту импульсов
1	1	1	Вывод T0, инкремент счетчика производится по нарастающему фронту импульсов

При переходе таймера/счетчика из состояния $FF в состояние $00 устанавливается флаг TOV0 (табл. 4.6) и генерируется запрос на прерывание. Разрешение прерывания осуществляется установкой в 1 разряда TOIE0 (табл. 4.7) при условии, что флаг общего разрешения прерываний I регистра SREG также установлен в 1.

Таблица 4.6 – Формат регистра флагов прерываний TIFR

7	6	5	4	3	2	1	0
–	OCF1A	–	–	–	TOV1	TOV0	–

Таблица 4.7 – Формат регистра масок прерываний TIMSK

7	6	5	4	3	2	1	0
–	OCIE1A	–	–	–	TOIE1	TOIE0	–

В таймере/счетчике T1 возможность счета внешних импульсов отсутствует. Однако он может выполнять определенные действия при равенстве содержимого счетного регистра TCNT1 и регистров выходного сравнения OCR1A и OCR1B. Кроме того, он может работать как широтно-импульсный модулятор для генерирования сигнала с программируемой частотой и скважностью. Дальнейшее описание T1 приводится именно для режима ШИМ.

При работе таймера/счетчика в режиме ШИМ состояние счетного регистра изменяется от $00 до значения, находящегося в регистре OCR1B, после чего счетный регистр сбрасывается и цикл повторяется. При равенстве содержимого счетного регистра и регистра OCR1A состояние вывода PB1 (OC1A) изменяется в соответствии со значениями разрядов COM1A1 и COM1A0 регистра управления TCCR1 (табл. 4.9). Выбор источника тактового сигнала задается разрядами CS13…CS10 этого регистра (табл. 4.10), а включение режима ШИМ – записью 1 в разряд PWM1 (табл. 4.8).

Таблица 4.8 – Формат регистра TCCR1

7	6	5	4	3	2	1	0
CTC1	PWM1	COM1A1	COM1A0	CS13	CS12	CS11	CS10

Таблица 4.9 – Поведение вывода PB1 (OC1A) в режиме ШИМ

COM1A1	COM1A0	PB1 (OC1A)
0	0	Таймер/счетчик T1 отключен от вывода
0	1	Таймер/счетчик T1 отключен от вывода
1	0	Сбрасывается в 0 при равенстве TCNT1 и OCR1A, устанавливается в 1 при TCNT1=$00
1	1	Устанавливается в 1 при равенстве TCNT1 и OCR1A, сбрасывается в 0 при TCNT1=$00

Таблица 4.10 – Выбор источника тактового сигнала для таймера T1

Регистр TCCR1				Источник тактового сигнала
CS13	CS12	CS11	CS10	
0	0	0	0	Таймер/счетчик остановлен
0	0	0	1	СКх16
0	0	1	0	СКх8
0	0	1	1	СКх4
0	1	0	0	СКх2
0	1	0	1	СК (тактовый сигнал МК)
0	1	1	0	СК/2
0	1	1	1	СК/4
1	0	0	0	СК/8
1	0	0	1	СК/16
1	0	1	0	СК/32
1	0	1	1	СК/64
1	1	0	0	СК/128
1	1	0	1	СК/256
1	1	1	0	СК/512
1	1	1	1	СК/1024

Содержимое регистра сравнения OCR1B определяет частоту ШИМ-сигнала (табл. 4.11), содержимое регистра сравнения OCR1A определяет скважность ШИМ-сигнала.

Таблица 4.11 – Зависимость частоты ШИМ-сигнала от тактовой частоты

Частота тактового сигнала таймера/счетчика	Содержимое регистра OCR1B	Частота ШИМ-сигнала (кГц)
СК	159	10
СКх2	159	20
СКх4	213	30
СКх4	159	40
СКх8	255	50
СКх8	213	60
СКх8	181	70
СКх8	159	80
СКх8	141	90
СКх16	255	100
СКх16	231	110
СКх16	213	120
СКх16	195	130
СКх16	181	140
СКх16	169	150

При работе таймера/счетчика T1 в режиме ШИМ может генерироваться прерывание по переполнению T1, а также прерывание от схемы сравнения (см. флаги и биты разрешения регистров TIFR и TIMSK).

В микроконтроллере ATtiny15L имеется встроенный синтезатор частоты, формирующий сигнал с частотой, в 16 раз превышающей частоту тактового сигнала встроенного RC-генератора. Номинальная частота RC-генератора равна 1,6 МГц, а частота на выходе синтезатора частоты равна 25,6 МГц.

Основная функция сторожевого таймера – защита устройства от сбоев. Благодаря сторожевому таймеру можно прервать выполнение зациклившейся программы. Если сторожевой таймер включен, то через определенные промежутки времени выполняется сброс микроконтроллера. При нормальном выполнении программы сторожевой таймер должен периодически сбрасываться командой WDR.

Для управления сторожевым таймером предназначен регистр WDTCR (табл. 4.12). Краткое описание разрядов этого регистра приведено в табл. 4.13. Непосредственно перед включени-

ем сторожевого таймера рекомендуется выполнить его сброс командой WDR.

Таблица 4.12 – Формат регистра WDTCR

7	6	5	4	3	2	1	0
–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0

Таблица 4.13 – Разряды регистра WDTCR

Разряд	Название	Описание
7…5	–	Зарезервированы
4	WDTOE	Разрешение выключения сторожевого таймера
3	WDE	Разрешение включения сторожевого таймера
2…0	WDP2…WDP0	Коэффициент деления предделителя частоты

Период наступления тайм-аута сторожевого таймера задается с помощью разрядов WDP2…WDP0 согласно табл. 4.14. Сторожевой таймер имеет независимый тактовый генератор с номинальным значением частоты 1 МГц и может работать даже в режиме Power Down.

Таблица 4.14 – Задание периода сторожевого таймера

WDP2	WDP1	WDP0	Число тактов генератора
0	0	0	16К
0	0	1	32К
0	1	0	64К
0	1	1	128К
1	0	0	256К
1	0	1	512К
1	1	0	1024К
1	1	1	2048К

Стек

В микроконтроллерах AVR семейства Tiny стек реализован аппаратно. Глубина стека равна трем уровням, а разрядность равна размеру счетчика команд (9 разрядов). При вызове подпрограммы адрес команды, расположенной за командой RCALL, со-

храняется в стеке. При возврате из подпрограммы этот адрес извлекается из стека и загружается в счетчик команд. То же происходит и во время прерывания программы.

Непосредственно из программы стек недоступен, так как в наборе команд микроконтроллера отсутствуют команды занесения в стек и извлечения из стека. Указатель стека также недоступен из программы. Микроконтроллер сам управляет перемещением данных по стеку.

Энергонезависимая память данных EEPROM

Для обращения к EEPROM (ее объем составляет 64 байта) используются три регистра ввода/вывода: регистр адреса EEAR, регистр данных EEDR и регистр управления EECR (табл. 4.15 и 4.16).

Таблица 4.15 – Формат регистра EECR

7	6	5	4	3	2	1	0
–	–	–	–	EERIE	EEMWE	EEWE	EERE

Таблица 4.16 – Разряды регистра EECR

Разряд	Название	Описание
7…4	–	Не используются, читаются как 0
3	EERIE	Разрешение прерывания от EEPROM. Данный разряд управляет генерацией прерывания, возникающего при завершении цикла записи в EEPROM. Если этот разряд установлен в 1, прерывания разрешены (если флаг I регистра SREG также установлен в 1). При сброшенном разряде EEMWE прерывание генерируется постоянно
2	EEMWE	Управление разрешением записи в EEPROM. После программной установки этот разряд сбрасывается аппаратно через 4 такта
1	EEWE	Разрешение записи в EEPROM. При установке этого разряда в 1 происходит запись данных в EEPROM, если EEMWE=1
0	EERE	Разрешение чтения из EEPROM. По окончании чтения сбрасывается аппаратно

Для записи одного байта в EEPROM необходимо:

- дождаться готовности EEPROM к записи (ждать пока не сбросится флаг EEWE);
- загрузить байт данных в регистр EEDR, а требуемый адрес – в регистр EEAR;
- установить в 1 флаг EEMWE;
- в течение 4-х машинных циклов после установки EEMWE записать 1 в разряд EEWE.

Рекомендуется запрещать все прерывания при выполнении пунктов 2…4 описанной последовательности. Длительность процесса записи составляет 4…8 мс. Процедура чтения из EEPROM гораздо проще. После загрузки требуемого адреса в регистр EEAR программа должна установить в 1 разряд EERE. Когда запрошенные данные будут находиться в регистре данных EEDR, произойдет аппаратный сброс этого разряда.

Аналоговый компаратор

Будучи включенным, компаратор позволяет сравнить значения напряжений на выводах PB0 и PB1. Чтобы указанные линии порта могли использоваться аналоговым компаратором, они должны быть сконфигурированы как входы. Внутренние подтягивающие резисторы, если они подключены, при разрешении работы компаратора отключаются автоматически.

Результатом сравнения является логическое значение, которое может быть прочитано из программы. По результату сравнения может быть сгенерировано прерывание. Управление работой компаратора осуществляется с помощью битов регистра ACSR (табл. 4.17–4.18). При включении напряжения питания все разряды регистра ACSR сбрасываются в 0. К неинвертирующему входу компаратора вместо вывода AIN0 микроконтроллера может быть подключен внутренний источник опорного напряжения величиной 1.22±0.05 В.

Таблица 4.17 – Разряды регистра ACSR

Разряд	Название	Описание
7	ACD	Выключение компаратора (1 – выключен)
6	ACBG	Подключение к неинвертирующему входу компаратора внутреннего ИОН (1 – подключен, 0 – не подключен)
5	ACO	Результат сравнения (выход компаратора)
4	ACI	Флаг прерывания от компаратора
3	ACIE	Разрешение прерывания от компаратора
2	-	Зарезервирован
1,0	ACIS1:ACIS0	Условия возникновения прерывания от компаратора

Таблица 4.18 – Условия генерации запроса на прерывание от компаратора

ACIS1	ACIS0	Условие
0	0	Любое изменение состояния выхода компаратора
0	1	Зарезервировано
1	0	Изменение состояния выхода компаратора с 1 на 0
1	1	Изменение состояния выхода компаратора с 0 на 1

Аналого-цифровой преобразователь

В процессе работы АЦП может функционировать в двух режимах:

- режим одиночного преобразования – запуск каждого преобразования инициируется пользователем;
- режим непрерывного преобразования – запуск преобразований выполняется непрерывно через определенные интервалы времени.

Управление модулем АЦП и контроль его состояния осуществляется с помощью регистра ADCSR (табл. 4.19).

Таблица 4.19 – Разряды регистра ADCSR

Разряд	Название	Описание
7	ADEN	Разрешение АЦП (1 – включен)
6	ADSC	Запуск преобразования (1 – начать преобразование)
5	ADFR	Выбор режима работы АЦП (0 – одиночное преобразование)
4	ADIF	Флаг прерывания от АЦП
3	ADIE	Разрешение прерывания от АЦП
2…0	ADPS2:ADPS0	Выбор частоты преобразования

Таблица 4.20 – Задание коэффициента деления предделителя АЦП

ADRS2	ADRS1	ADRS0	Коэффициент деления
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Наибольшая точность преобразования достигается при тактовой частоте модуля АЦП в диапазоне 50…200 кГц. Для этого тактовая частота микроконтроллера поступает на АЦП через предделитель с программируемым коэффициентом деления. Для повышения точности преобразования (чтобы свести к минимуму помехи, наводимые ядром процессора) в микроконтроллере предусмотрен специальный спящий режим – ADC Noise Reduction. В этом режиме из всех периферийных устройств функционируют только АЦП и сторожевой таймер. Сразу же после остановки процессора начнется цикл преобразования. По завершении преобразования будет сгенерировано прерывание от АЦП, которое переведет микроконтроллер в рабочий режим и начнется выполнение подпрограммы обработки этого прерывания.

Поскольку АЦП 10-разрядный, результат преобразования размещен в двух регистрах, доступных только для чтения: ADCH и ADCL. Сначала необходимо прочитать ADCL, а затем ADCH.

Если достаточно точности восьми разрядов, для получения результата достаточно прочитать содержимое регистра ADCH.

Номер активного канала (аналоговый вход, подключаемый ко входу АЦП) и источника опорного напряжения задается регистром ADMUX (табл. 4.21–4.23). Разряд ADLAR служит для управления выравниванием результата преобразования. Если этот разряд установлен в 1, результат преобразования выравнивается по левой границе 16-разрядного слова, если сброшен в 0 – по правой границе.

Таблица 4.21 – Разряды регистра ADMUX

Разряд	Название	Описание
7,6	REFS1:REFS0	Выбор источника опорного напряжения
5	ADLAR	Выравнивание результата преобразования
4,3	–	Зарезервировано
2…0	MUX2:MUX0	Выбор входного канала

Таблица 4.22 – Выбор источника опорного напряжения

REFS1	REFS0	Источник опорного напряжения
0	0	Напряжение питания микроконтролера
0	1	Внешний ИОН, подключенный к выводу PB0, внутренний ИОН отключен
1	0	Внутренний ИОН напряжением 2,56 В, отключенный от вывода PB0 (AREF)
1	1	Внутренний ИОН напряжением 2,56 В, подключенный к выводу PB0 (AREF)

Таблица 4.23 – Номер активного канала

MUX2	MUX 1	MUX 0	Вход
0	0	0	ADC0 (PB5)
0	0	1	ADC1 (PB2)
0	1	0	ADC2 (PB3)
0	1	1	ADC3 (PB4)

Программа работы

1. Загрузить для отладки в AVR Studio программу преобразования целых 16-битных чисел в двоично-десятичные числа. Алгоритм программы «**bin16BCD5**» заключается в следующем. Предположим, что имеется целое беззнаковое 16-битное число

(диапазон от 0 до 65535). Очевидно, что необходимо найти 5 десятичных цифр. Способ преобразования заключается в том, чтобы, вычитая из исходного числа число 10000, сначала определить десятичную цифру десятков тысяч. Затем находится цифра тысяч последовательным вычитанием числа 1000 и т.д. Вычитание каждый раз производится до получения отрицательной разности с подсчетом числа вычитаний. При переходе к определению каждого следующего десятичного разряда в регистрах исходного числа восстанавливается последняя положительная разность. После того, как будет найдена десятичная цифра десятков, в регистрах исходного числа останется десятичная цифра единиц.

Проследить выполнение программы в пошаговом и автоматическом режиме, записав предварительно в регистры r16 и r17 шестнадцатеричное число $NNNN, где N — номер варианта, рассчитанный по методике ТМЦ ДО (число от 1 до 9). В окне I/O раскройте содержимое Register 1–31, Processor, I/O ATTINY15 (CPU, WATCHDOG). Какие команды программы влияют на флаги регистра статуса SREG? Зафиксируйте в отчете результат преобразования. В программе часто используются команды вычитания константы из регистра. Есть ли в системе команд AVR аналогичные команды сложения регистра и константы? Как будет работать программа, если в ней удалить последнюю команду?

```
;***** Программа bin16BCD5
.DEVICE ATtiny15        ; Определить устройство
.INCLUDE
"C:\Program Files\Atmel\AVR Tools\AvrAssembler\Appnotes\tn15def.inc"
; Вложить файл определения адресов регистров ввода\вывода

;***** Регистровые переменные
.def   fbinL    =r16        ; двоичное значение, младший байт
.def   fbinH    =r17        ; двоичное значение, старший байт
.def   tBCD0    =r17        ; BCD значение, цифры 1 и 0
.def   tBCD1    =r18        ; BCD значение, цифры 3 и 2
.def   tBCD2    =r19        ; BCD значение, цифра 4
; Переменные fbinH и tBCD0 должны размещаться в одном регистре
      WDR                       ; Сброс сторожевого таймера
      ldi    r20,0b00001000     ; Включение сторожевого
      out    WDTCR,r20          ; таймера
      ldi    tBCD2, -1          ; Начало преобразования
```

```
m1:
      inc    tBCD2
      subi   fbinL, low(10000)
      sbci   fbinH, high(10000)
      brsh   m1
      subi   fbinL, low(-10000)
      sbci   fbinH, high(-10000)
      ldi    tBCD1, -0x11
m2:
      subi   tBCD1, -0x10
      subi   fbinL, low(1000)
      sbci   fbinH, high(1000)
      brsh   m2
      subi   fbinL, low(-1000)
      sbci   fbinH, high(-1000)
m3:
      inc    tBCD1
      subi   fbinL, low(100)
      sbci   fbinH, high(100)
      brsh   m3
      subi   fbinL, -100
      ldi    tBCD0, -0x10
m4:
      subi   tBCD0, -0x10
      subi   fbinL, 10
      brsh   m4
      subi   fbinL, -10
      add    tBCD0, fbinL          ; Конец преобразования

m5:  rjmp   m5                     ; Зацикливание программы
```

Какой период срабатывания сторожевого таймера задан в программе? Что будет, если дождаться его срабатывания?

2. Загрузить для отладки в AVR Studio программу **CLOK**, реализующую двоично-десятичный счетчик на регистре r19. Счетчик считает с частотой прерываний по переполнению таймера T0. Тактовый сигнал на вход таймера подается через программируемый делитель частоты. Коэффициент пересчета счетчика равен 100. Для счета используются вспомогательные регистры r16 (счет единиц), r17 (счет десятков) и r18 (объединение десят-

ков и единиц). Основная программа обнуляет регистры счетчика, устанавливает режим работы T0, разрешает прерывания по переполнению T0 и зацикливается. Двоично-десятичный счет реализуется в подпрограмме прерывания, расположенной начиная с адреса вектора прерывания по переполнению таймера T0. Заметим, что в системе команд AVR нет команды сложения регистра с константой и команды десятичной коррекции аккумулятора, как и самого аккумулятора.

Набрать исходный текст программы CLOK.asm без комментария. Проверить работу в пошаговом и автоматическом режимах. В окне I/O AVR Studio раскройте содержимое Register 1–31, I/O ATTINY15 (CPU, TIMER_COUNTER_0).

```
;****** Программа CLOK
.DEVICE ATtiny15
.INCLUDE
 "C:\Program Files\Atmel\AVR Tools\AvrAssembler\Appnotes\tn15def.inc"

      rjmp  RESET
.org  $005                  ; Вектор прерывания по переполнению T0
      inc   r16             ; Инкремент единиц
      cpi   r16,$0A
      breq  m1
      rjmp  m3
m1:   clr   r16
      subi  r17,-$10        ; Инкремент десятков
      cpi   r17,$A0
      breq  m2
      rjmp  m3
m2:   clr   r17
m3:   mov   r18,r17
      add   r18,r16
      mov   r19,r18         ; Инкремент двоично-десятичного счета
      reti

RESET:
      clr   r16             ; Обнуление регистров
      clr   r17
      clr   r18
      clr   r19
      ldi   r20,0b00000001  ; Выбор источника тактового сигнала для T0
```

```
        out     TCCR0,r20
        ldi     r20,0b00000010    ; Разрешение прерываний
        out     TIMSK,r20         ; по переполнению таймера T0
        sei                       ; Глобальное разрешение прерываний
m4:     rjmp    m4
```

С какой частотой переполняется T0? Каким образом можно уменьшить скорость счета в 1024 раза? Объяснить поведение регистров SREG и TIFR при работе программы. Пояснить назначение директивы .DEVICE. Пояснить содержимое файла clok.map в окне Project.

Изменить программу так, чтобы уменьшить коэффициент пересчета счетчика до 10N, где N — номер варианта. В отчете представить измененный вариант программы с комментарием.

3. Сформировать на выводе PB1 (OC1A) микроконтроллера ШИМ-сигнал с частотой 50 кГц (программа **PWM1**). Таймер T1 используется как генератор импульсов с программируемым периодом (содержимое регистра сравнения OCR1B) и длительностью (содержимое регистра сравнения OCR1A). В окне I/O раскройте содержимое Register 1–31, Processor, I/O ATTINY15 (PORTB, TIMER_COUNTER_1).

```
;****** Программа PWM1
.DEVICE ATtiny15
.INCLUDE
"C:\Program Files\Atmel\AVR Tools\AvrAssembler\Appnotes\tn15def.inc"

        sbi     DDRB,1            ; Настройка первой линии порта B на вывод
        ldi     r16,0b01100010    ; Режим работы T1 (ШИМ с частотой
                                  ; тактирования 12.8 МГц, 1 при сбросе,
                                  ; 0 при сравнении)
        out     TCCR1,r16
        ldi     r16,0xFF          ; Частота импульсов 50 кГц
        out     OCR1B,r16
        ldi     r16,0x80          ; Скважность импульсов примерно 2
        out     OCR1A,r16
m1:     rjmp    m1
```

Проследить работу программы в пошаговом режиме. На сколько меняется содержимое T1 при выполнении команды rjmp m1? Почему? Какие флаги устанавливаются в регистре TIFR?

Модифицировать программу так, чтобы частота ШИМ составила 20 кГц, а скважность 4 (отношение периода к длительности импульса).

4. Проверить программу обращения к энергонезависимой памяти данных EEPROM (проект **EEPROM**). Открыть окна для просмотра регистров общего назначения 16–31, регистров ввода/вывода (CPU, EEPROM), памяти EEPROM (Memory 3). Проследите выполнение программы в пошаговом режиме. Когда выполняется подпрограмма прерывания и что она делает? Специального флага прерываний от EEPROM нет, поэтому при обращении к подпрограмме прерывания флаг не сбрасывается и прерывания генерируются постоянно.

```
;****** Программа EEPROM
.DEVICE ATtiny15
.INCLUDE
"C:\Program Files\Atmel\AVR Tools\AvrAssembler\Appnotes\tn15def.inc"

.cseg
        ; Рабочие переменные
        .def    AddrReg=r20
        .def    Data1Reg=r21
        .def    Data2Reg=r22

        ; Векторы прерываний
        rjmp    RESET
        reti
        reti
        reti
        reti
        reti
        rjmp    EEPROM_READY    ; Вектор прерывания по записи в EEPROM
        reti
        reti

EEPROM_READY:           ; Подпрограмма прерывания по окончании
        inc     r25             ; цикла записи в EEPROM
        reti
```

```
EEWrite:                    ; Подпрограмма записи байта в EEPROM
        sbic    EECR,EEWE   ; Ждать, пока флаг EEWE
        rjmp    EEWRite     ; не будет сброшен
        cli                 ; Запретить прерывания
        out     EEAR,AddrReg ; Загрузить адрес
        out     EEDR,Data1Reg ; Загрузить данные
        sbi     EECR,EEMWE
        sbi     EECR,EEWE   ; Выдать строб записи байта в EEPROM
        sbi     EECR,EERIE  ; Разрешить прерывание по завершении
        sei                 ; цикла записи в EEPROM
        cbi     EECR,EERIE  ; Запретить дальнейшие прерывания
        ret
EERead:                     ; Подпрограмма чтения байта EEPROM
        sbic    EECR,EEWE   ; Ждать окончания текущей записи пока
        rjmp    EERead      ; флаг EEWE не равен 0
        rjmp    EEWRite
        out     EEAR,AddrReg ; Загрузить адрес
        sbi     EECR,EERE   ; Выдать строб чтения из EEPROM
        in      Data2Reg,EEDR ; Прочитанный байт в регистр
        ret
RESET:                      ; Начало основной программы
        clr     r25         ; Очистка регистров
        clr     r21
        clr     r22
        ldi     AddrReg,$18
        ldi     Data1Reg,$DD
        rcall   EEWrite     ; Вызов подпрограммы записи в EEPROM
        rcall   EERead      ; Вызов подпрограммы чтения из EEPROM
m1:     rjmp    m1
```

Изменить программу так, чтобы она записывала в ячейку N EEPROM число 100+N, а читала записанный байт в регистр rN, где N – номер варианта (число от 1 до 9).

Контрольные вопросы

1. Где сохраняется адрес возврата при обращении к подпрограммам? Почему в программной модели микроконтроллера ATtiny15 нет указателя стека? Какую разрядность имеет стек ATtiny15 и сколько в нем ячеек? Можно ли в ATtiny15 реализовать в вложенные прерывания программы?

2. Перечислите все источники прерываний в ATtiny15 в порядке убывания приоритета.

3. Каким образом программируется FLASH память программ ATtiny15?

4. Какой командой микроконтроллер переводится в «спящий» режим?

5. Биты каких регистров можно устанавливать и сбрасывать командами sbi и cbi?

Содержание отчета

Отчет в формате WORD должен содержать тексты измененных (в соответствии с вариантом задания) программ с комментариями, ответы на вопросы по пунктам работы, рисунки, отображающие окна регистров и памяти, ответы на контрольные вопросы.

Перечень команд микроконтроллера ATtiny15L

Арифметические и логические команды

ADD	Rd, Rr	Add Two Registers
ADC	Rd, Rr	Add with Carry Two Registers
SUB	Rd, Rr	Subtract Two Registers
SUBI	Rd, K	Subtract Constant from Register
SBC	Rd, Rr	Subtract with Carry Two Registers
SBCI	Rd, K	Subtract with Carry Constant from Reg.
AND	Rd, Rr	Logical AND Registers
ANDI	Rd, K	Logical AND Register and Constant
OR	Rd, Rr	Logical OR Registers
ORI	Rd, K	Logical OR Register and Constant
EOR	Rd, Rr	Exclusive OR Registers
COM	Rd	One’s Complement
NEG	Rd	Two’s Complement
SBR	Rd, K	Set Bit(s) in Register
CBR	Rd, K	Clear Bit(s) in Register
INC	Rd	Increment
DEC	Rd	Decrement
TST	Rd	Test for Zero or Minus
CLR	Rd	Clear Register
SER	Rd	Set Register

Команды передачи управления

RJMP	k	Relative Jump
RCALL	k	Relative Subroutine Call
RET		Subroutine Return
RETI		Interrupt Return
CPSE	Rd, Rr	Compare, Skip if Equal)
CP	Rd, Rr	Compare
CPC	Rd, Rr	Compare with Carry
CPI	Rd, K	Compare Register with Immediate
SBRC	Rr, b	Skip if Bit in Register Cleared
SBRS	Rr, b	Skip if Bit in Register is Set
SBIC	P, b	Skip if Bit in I/O Register Cleared)
SBIS	P, b	Skip if Bit in I/O Register is Set
BRBS	s, k	Branch if Status Flag Set
BRBC	s, k	Branch if Status Flag Cleared
BREQ	k	Branch if Equal
BRNE	k	Branch if Not Equal
BRCS	k	Branch if Carry Set
BRCC	k	Branch if Carry Cleared
BRSH	k	Branch if Same or Higher
BRLO	k	Branch if Lower
BRMI	k	Branch if Minus
BRPL	k	Branch if Plus
BRGE	k	Branch if Greater or Equal, Signed
BRLTk		Branch if Less Than Zero, Signed
BRHS	k	Branch if Half-carry Flag Set
BRHC	k	Branch if Half-carry Flag Cleared
BRTS	k	Branch if T-flag Set
BRTC	k	Branch if T-flag Cleared
BRVS	k	Branch if Overflow Flag is Set
BRVC	k	Branch if Overflow Flag is Cleared
BRIE	k	Branch if Interrupt Enabled
BRID	k	Branch if Interrupt Disabled

Команды пересылки данных

LD	Rd, Z	Load Register Indirect
ST	Z, Rr	Store Register Indirect
MOV	Rd, Rr	Move between Registers
LDI	Rd, K	Load Immediate
IN	Rd, P	In Port
OUT	P, Rr	Out Port
LPM		Load Program Memory

Команды операций с битами

SBI	P, b	Set Bit in I/O Register
CBI	P, b	Clear Bit in I/O Register
LSL	Rd	Logical Shift Left
LSR	Rd	Logical Shift Right
ROL	Rd	Rotate Left through Carry
ROR	Rd	Rotate Right through Carry
ASR	Rd	Arithmetic Shift Right
SWAP	Rd	Swap Nibbles
BSET	s	Flag Set
BCLR	s	Flag Clear
BST	Rr, b	Bit Store from Register to T
BLD	Rd, b	Bit Load from T to Register
SEC		Set Carry
CLC		Clear Carry
SEN		Set Negative Flag
CLN		Clear Negative Flag
SEZ		Set Zero Flag
CLZ		Clear Zero Flag
SEI		Global Interrupt Enable
CLI		Global Interrupt Disable
SES		Set Signed Test Flag
CLS		Clear Signed Test Flag
SEV		Set Two’s Complement Overflow
CLV		Clear Two’s Complement Overflow
SET		Set T in SREG
CLT		Clear T in SREG
SEH		Set Half-carry Flag in SREG
CLH		Clear Half-carry Flag in SREG
NOP		No Operation
SLEEP		Sleep
WDR		Watchdog Reset

5 МИКРОКОНТРОЛЛЕР ATmega8 (лабораторная работа № 4)

Цель работы. Целью лабораторной работы является отладка прикладных программ для микроконтроллера AVR семейства Mega с помощью персонального компьютера и программных средств отладки.

Отличительные особенности ATmega8

- 8-разрядный высокопроизводительный AVR микроконтроллер с малым потреблением.
- Прогрессивная RISC архитектура:
  - 130 высокопроизводительных команд, большинство команд выполняется за один тактовый цикл.
  - 32 8-разрядных рабочих регистра общего назначения.
- Полностью статическая работа.
- Приближающаяся к 16 MIPS (при тактовой частоте 16 МГц) производительность.
- Встроенный 2-цикловый перемножитель.
- Энергонезависимая память программ и данных:
  - 8 Кбайт внутрисистемно программируемой Flash памяти.
  - Обеспечивает 1000 циклов стирания/записи.
  - Дополнительный сектор загрузочных кодов с независимыми битами блокировки.
  - 512 байт EEPROM.
  - Обеспечивает 100000 циклов стирания/записи.
  - 1 Кбайт встроенной SRAM.
  - Программируемая блокировка, обеспечивающая защиту программных средств пользователя.
- Встроенная периферия:
  - Два 8-разрядных таймера/счетчика с отдельным предварительным делителем, один с режимом сравнения.
  - Один 16-разрядный таймер/счетчик с отдельным предварительным делителем и режимами захвата и сравнения.
  - Счетчик реального времени с отдельным генератором.
  - Три канала ШИМ (PWM).

- 8-канальный аналого-цифровой преобразователь (в корпусах TQFP и MLF).
- 6 каналов с 10-разрядной точностью.
- 2 канала с 8-разрядной точностью.
- 6-канальный аналого-цифровой преобразователь (в корпусе PDIP).
- 4 канала с 10-разрядной точностью.
- 2 канала с 8-разрядной точностью.
- Байт-ориентированный 2-проводный последовательный интерфейс.
- Программируемый последовательный USART.
- Последовательный интерфейс SPI (ведущий/ведомый).
- Программируемый сторожевой таймер с отдельным встроенным генератором.
- Встроенный аналоговый компаратор.

- Специальные микроконтроллерные функции:
  - Сброс по подаче питания и программируемый детектор кратковременного снижения напряжения питания.
  - Встроенный калиброванный RC-генератор.
  - Внутренние и внешние источники прерываний.
  - Пять режимов пониженного потребления: Idle, Power-save, Power-down, Standby и снижения шумов ADC.
- Выводы I/O и корпуса:
  - 23 программируемые линии ввода/вывода.
  - 28-выводной корпус PDIP, 32-выводной корпус TQFP (рис. 5.1).
- Рабочие напряжения:
  - 2,7 – 5,5 В (ATmega8L).
  - 4,5 – 5,5 В (ATmega8).
- Рабочая частота:
  - 0 – 8 МГц (ATmega8L).
  - 0 – 16 МГц (ATmega8).

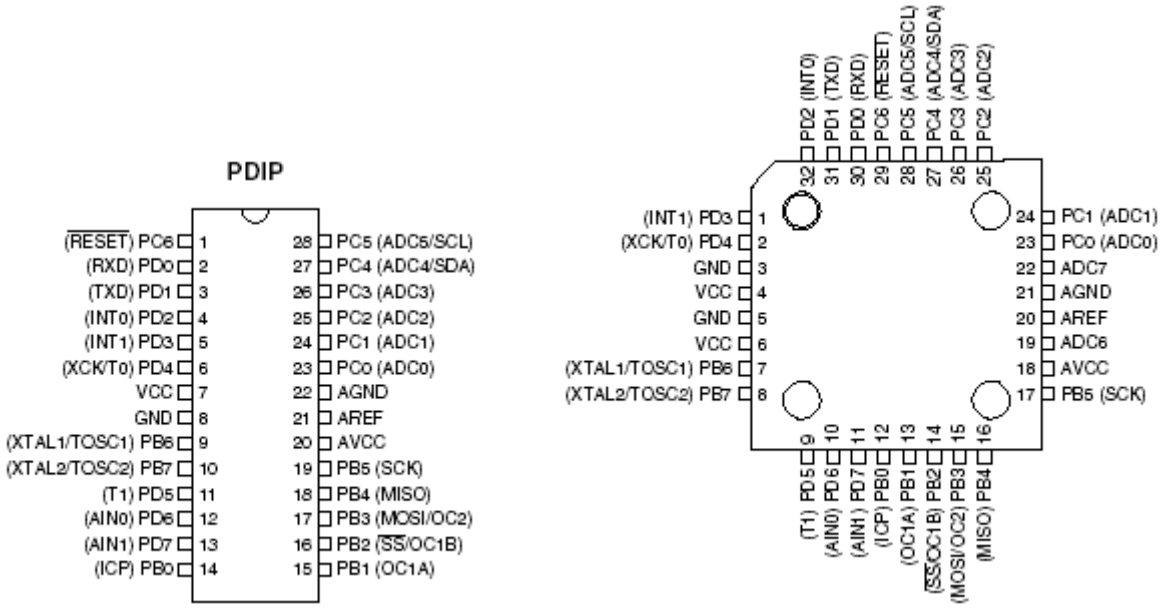


Рис. 5.1 – Расположение выводов ATmega8

Таблица 5.1 – Описание выводов ATmega8

Обозначение	Описание
PB0 (ICP)	B0 (Вход захвата таймера/счетчика T1)
PB1 (OC1A)	B1 (Выход сравнения A таймера/счетчика T1)
PB2 (SS/OC1B)	B2 (Выбор Slave-устройства в канале SPI/ выход сравнения B таймера/счетчика T1)
PB3 (MOSI/OC2)	B3 (Выход (Master) или вход (Slave) данных канала SPI/выход сравнения таймера/счетчика T2)
PB4 (MISO)	B4 (Вход (Master) или выход (Slave) данных канала SPI
PB5 (SCK)	B5 (Выход (Master) или вход (Slave) тактового сигнала SPI
PB6 (XTAL1/TOSC1)	B6 (Вход тактового генератора/вывод для подключения резонатора к таймеру/счетчику T2)
PB7 (XTAL2/TOSC2)	B7 (Выход тактового генератора/вывод для подключения резонатора к таймеру/счетчику T2)
PC0 (ADC0)	C0 (Вход АЦП)
PC1 (ADC1)	C1 (Вход АЦП)
PC2 (ADC2)	C2 (Вход АЦП)
PC3 (ADC3)	C3 (Вход АЦП)
PC4 (ADC4/SDA)	C4 (Вход АЦП/линия данных модуля TWI)
PC5 (ADC5/SCL)	C5 (Вход АЦП/тактовый сигнал модуля TWI)
PC6 (RESET)	C6 (Вход сброса)

Окончание табл. 5.1

Обозначение	Описание
ADC6	Вход АЦП
ADC7	Вход АЦП
PD0 (RXD)	D0 (Вход USART)
PD1 (TXD)	D1 (Выход USART)
PD2 (INT0)	D2 (Вход внешнего прерывания)
PD3 (INT1)	D3 (Вход внешнего прерывания)
PD4 (T0/XCK)	D4 (Вход внешнего тактового сигнала таймера/счетчика T0/тактовый сигнал USART)
PD5 (T1)	D5 (Вход внешнего тактового сигнала таймера/счетчика T1)
PD6 (AIN0)	D6 (Неинвертирующий вход компаратора)
PD7 (AIN1)	D7 (Инвертирующий вход компаратора)
AREF	Вход опорного напряжения для АЦП
AGND	Аналоговый общий вывод
AVcc	Вывод источника питания АЦП
GND	Общий вывод
Vcc	Вывод источника питания

Таблица 5.2 – Регистры ввода/вывода (адрес, название и функция)

0x3F (0x5F)	SREG	Регистр состояния
0x3E (0x5E)	SPH	Указатель стека, старший байт
0x3D (0x5D)	SPL	Указатель стека, младший байт
0x3B (0x5B)	GICR	Общий регистр управления прерываниями
0x3A (0x5A)	GIFR	Общий регистр флагов прерываний
0x39 (0x59)	TIMSK	Маски прерываний от таймеров/счетчиков
0x38 (0x58)	TIFR	Флаги прерываний от таймеров/счетчиков
0x37 (0x57)	SPMCR	Регистр управления памятью программ
0x36 (0x56)	TWCR	Регистр управления TWI
0x35 (0x55)	MCUCR	Регистр управления микроконтроллером
0x34 (0x54)	MCUCSR	Регистр управления и состояния МК
0x33 (0x53)	TCCR0	Регистр управления таймером/счетчиком T0
0x32 (0x52)	TCNT0	Счетный регистр таймера/счетчика T0
0x31 (0x51)	OSCCAL	Регистр калибровки тактового генератора
0x30 (0x50)	SFIOR	Регистр специальный функций
0x2F (0x4F)	TCCR1A	Регистр управления A таймера/счетчика T1
0x2E (0x4E)	TCCR1B	Регистр управления B таймера/счетчика T1
0x2D (0x4D)	TCNT1H	Счетный регистр T1, старший байт
0x2C (0x4C)	TCNT1L	Счетный регистр T1, младший байт
0x2B (0x4B)	OCR1AH	Регистр совпадения A T1, старший байт

Окончание табл. 5.2

0x2A (0x4A)	OCR1AL	Регистр совпадения A T1, младший байт
0x29 (0x49)	OCR1BH	Регистр совпадения B T1, старший байт
0x28 (0x48)	OCR1BL	Регистр совпадения B T1, младший байт
0x27 (0x47)	ICR1H	Регистр захвата T1, старший байт
0x26 (0x46)	ICR1L	Регистр захвата T1, младший байт
0x25 (0x45)	TCCR2	Регистр управления таймера/счетчика T2
0x24 (0x44)	TCNT2	Счетный регистр таймера/счетчика T2
0x23 (0x43)	OCR2	Регистр совпадения таймера/счетчика T2
0x22 (0x42)	ASSR	Регистр состояния асинхронного режима
0x21 (0x41)	WDTCR	Регистр управления сторожевым таймером
0x20 (0x40)	UBRRH	Регистр управления USART
0x1F (0x3F)	EEARH	Регистр адреса EEPROM, старший байт
0x1E (0x3E)	EEARL	Регистр адреса EEPROM, младший байт
0x1D (0x3D)	EEDR	Регистр данных EEPROM
0x1C (0x3C)	EECR	Регистр управления EEPROM
0x18 (0x38)	PORTB	Регистр данных порта B
0x17 (0x37)	DDRB	Регистр направления данных порта B
0x16 (0x36)	PINB	Выводы порта B
0x15 (0x35)	PORTC	Регистр данных порта C
0x14 (0x34)	DDRC	Регистр направления данных порта C
0x13 (0x33)	PINC	Выводы порта C
0x12 (0x32)	PORTD	Регистр данных порта D
0x11 (0x31)	DDRD	Регистр направления данных порта D
0x10 (0x30)	PIND	Выводы порта D
0x0F (0x2F)	SPDR	Регистр данных SPI
0x0E (0x2E)	SPSR	Регистр состояния SPI
0x0D (0x2D)	SPCR	Регистр управления SPI
0x0C (0x2C)	UDR	Регистр данных USART
0x0B (0x2B)	UCSRA	Регистр управления и состояния A USART
0x0A (0x2A)	UCSRB	Регистр управления и состояния B USART
0x09 (0x29)	UBRRL	Регистр скорости передачи USART
0x08 (0x28)	ACSR	Состояние аналогового компаратора
0x07 (0x27)	ADMUX	Регистр управления мультиплексором АЦП
0x06 (0x26)	ADCSR	Регистр управления и состояния АЦП
0x05 (0x25)	ADCH	Регистр данных АЦП, старший байт
0x04 (0x24)	ADCL	Регистр данных АЦП, младший байт
0x03 (0x23)	TWDR	Регистр данных TWI
0x02 (0x22)	TWAR	Регистр адреса TWI
0x01 (0x21)	TWSR	Регистр состояния TWI
0x00 (0x20)	TWBR	Регистр скорости передачи TWI

Таблица 5.3 – Векторы прерываний (номер, адрес, источник и описание)

1	0x000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

Порты ввода-вывода

Все порты ввода-вывода (ПВВ) AVR-микроконтроллеров работают по принципу чтение-модификация-запись при использовании их в качестве портов универсального ввода-вывода. Это означает, что изменение направления ввода-вывода одной линии порта командами SBI и CBI будет происходить без ложных изменений направления ввода-вывода других линий порта. Данное распространяется также и на изменение логического уровня (если линия порта настроена на вывод) или на включение/отключение подтягивающих резисторов (если линия настроена на ввод). Каждый выходной буфер имеет симметричную характеристику управления с высоким втекающим и вытекающим выходными токами. Выходной драйвер обладает нагрузочной способностью, которая позволяет непосредственно управлять светодиодными индикаторами. Ко всем линиям портов может быть подключен индивидуальный выборочный подтягивающий к плюсу питания

резистор, сопротивление которого не зависит от напряжения питания. Ссылки на регистры и биты регистров в данном разделе даны в общей форме. При этом символ «x» заменяет наименование ПВВ, а символ «n» заменяет номер разряда ПВВ. Однако при составлении программы необходимо использовать точную форму записи. Например, PORTB3, означающий разряд 3 порта B, в данном случае записывается как PORTxn.

Для каждого порта ввода-вывода в памяти ввода-вывода зарезервировано три ячейки: одна под регистр данных – PORTx, другая под регистр направления данных – DDRx и третья под состояние входов порта – PINx. Ячейка, хранящая состояние на входах портов, доступна только для чтения, а регистры данных и направления данных имеют двунаправленный доступ. Кроме того, установка бита выключения подтягивающих резисторов PUD регистра SFIOR отключает функцию подтягивания на всех выводах всех портов.

Ниже приведено описание порта ввода-вывода для универсального цифрового ввода-вывода. Большинство выводов портов поддерживают альтернативные функции встроенных периферийных устройств микроконтроллера. Обратите внимание, что для некоторых портов разрешение альтернативных функций некоторых выводов делает невозможным использование других выводов для универсального цифрового ввода-вывода.

Режим и состояние для каждого вывода определяется значением соответствующих разрядов трех регистров: DDxn, PORTxn и PINxn. Доступ к битам DDxn возможен по адресу DDRx в пространстве ввода-вывода и, соответственно, к битам PORTxn по адресу PORTx, а к битам PINxn по адресу PINx.

Биты DDxn регистра DDRx определяют направленность линии ввода-вывода. Если DDxn=1, то Pxn конфигурируется на вывод. Если DDxn=0, то Pxn конфигурируется на ввод. Независимо от значения бита направления данных DDxn состояние вывода порта может быть опрошено через регистровый бит PINxn. В табл. 5.4 подытоживается действие управляющих сигналов на состояние вывода.

Таблица 5.4 – Настройка вывода порта

DDxn	PORTxn	PUD (в SFIOR)	Ввод-вывод	Подтягивающий резистор	Комментарий
0	0	X	Ввод	Нет	Третье состояние (Z-состояние)
0	1	0	Ввод	Да	Pxn будет источником тока при подаче внешнего низкого уровня
0	1	1	Ввод	Нет	Третье состояние (Z-состояние)
1	0	X	Вывод	Нет	Вывод лог. 0 (втекающий ток)
1	1	X	Вывод	Нет	Вывод лог. 1 (вытекающий ток)

16-разрядный таймер-счетчик T1

16-разрядный таймер-счетчик T1 предназначен для точного задания временных интервалов, генерации прямоугольных импульсов и измерения временных характеристик импульсных сигналов.

Регистры таймера T1

Регистр таймера-счетчика (TCNT1), регистры порогов сравнения (OCR1A и OCR1B), а также регистр захвата (ICR1) являются 16-разрядными регистрами. В связи с этим, во время доступа к этим регистрам должна быть соблюдена специальная процедура. Чтобы записать данные в 16-разрядный регистр, необходимо сначала записать старший байт, а затем младший. А при чтении 16-разрядного регистра, наоборот, сначала считывается младший байт, а затем старший.

Регистры управления таймером TCCR1A и TCCR1B (табл. 5.5 и табл. 5.6) являются 8-разрядными регистрами, поэтому доступ к ним со стороны ЦПУ не связан с какими-либо ограничениями. Все сигналы запросов на прерывание представлены в регистре флагов прерываний таймеров (TIFR). Все прерывания ин-

дивидуально маскируются регистром маски прерываний таймеров (TIMSK).

Таймер-счетчик может тактироваться внутренне через предделитель или внешне тактовым источником, подключенным к выводу T1. Блок выбора тактового источника позволяет выбрать тактовый источник и фронт, по которому будет изменяться состояние таймера-счетчика. Если тактовый источник не задан, то таймер-счетчик находится в неактивном состоянии. Сигнал на выходе блока выбора тактового источника является тактовым сигналом таймера.

Значение регистров порогов сравнения (OCR1A и OCR1B), непрерывно сравнивается со значением счетчика. Результат сравнения может использоваться для генерации прямоугольных импульсов с ШИМ или с переменной частотой на выходах OC1A и OC1B. В случае определения совпадения значений сравниваемых регистров устанавливается соответствующий флаг прерываний (OCF1A или OCF1B), который в свою очередь может служить источником прерывания.

Регистр захвата позволяет запомнить состояние таймера-счетчика при возникновении заданного внешнего события (фронт внешнего сигнала) на входе ICP или на выводах аналогового компаратора. На входе захвата фронта предусмотрена схема цифровой фильтрации (подавитель шума) для снижения риска срабатывания схемы захвата от помехи.

Таблица 5.5 – Формат регистра управления TCCR1A

7	6	5	4	3	2	1	0
COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

Таблица 5.6 – Формат регистра управления TCCR1B

7	6	5	4	3	2	1	0
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10

Назначение битов регистров управления:

COM1A1, COM1A0 – режим работы выходного сравнения A;

COM1B1, COM1B0 – режим работы выходного сравнения B;

WGM13, WGM12, WGM11, WGM10 — режим работы таймера/счетчика T1;

FOC1A, FOC1B – при записи в эти биты 1 моментально происходит событие выходного сравнения. Эти биты не работают в ШИМ режимах;

ICNC1 – установка режима подавления шума на входе захвата. При сброшенном в состояние 0 бите ICNC1 функция подавления шума входного триггера захвата запрещена. Вход захвата переключается по первому нарастающему/ падающему фронту, поступившему на вывод входа захвата. При установленном в состояние 1 бите ICNC1 выполняются четыре последовательных опроса состояния вывода и все четыре выборки должны иметь одинаковый (высокий/низкий), определяемый битом ICES1, уровень;

ICES1 – выбор фронта срабатывания на входе захвата. При сброшенном в состояние 0 бите ICES1 содержимое таймера/счетчика по падающему фронту на выводе входа захвата пересылается в регистр захвата входа ICR1. При установленном в 1 бите ICES1 содержимое таймера/счетчика пересылается в регистр захвата входа ICR1 по нарастающему фронту на выводе входа захвата;

CS12, CS11, CS10 – выбор источника тактовой частоты. Установкой состояния данных битов производится выбор источника тактового сигнала (в том числе коэффициента предварительного деления).

Режимы работы таймера T1

Под режимом работы 16-разрядного таймера понимается его алгоритм счета и поведение связанного с ним выхода формирователя импульсов, что определяется комбинацией бит, задающих режим работы таймера (табл. 5.7) и режим формирования выходного сигнала (табл. 5.9). В режимах с ШИМ биты COM1A (и точно также COM1B) позволяют включить/отключить инверсию на генерируемом ШИМ-выходе (т.е. выбрать ШИМ с инверсией или ШИМ без инверсии). Для режимов без ШИМ эти биты определяют, какое действие необходимо выполнить при возникновении совпадения: сбросить, установить или инвертировать выход.

Таблица 5.7 – Выбор режима работы таймера/счетчика T1

Режим	WGM13	WGM12	WGM11	WGM10	Режимы модуляции	Модуль счета
0	0	0	0	0	Нормальный	0xFFFF
1	0	0	0	1	ШИМ ФК 8-bit	0x00FF
2	0	0	1	0	ШИМ ФК 9-bit	0x01FF
3	0	0	1	1	ШИМ ФК 10-bit	0x03FF
4	0	1	0	0	СТС	OCR1A
5	0	1	0	1	Быстрая ШИМ 8-bit	0x00FF
6	0	1	1	0	Быстрая ШИМ 9-bit	0x01FF
7	0	1	1	1	Быстрая ШИМ 10-bit	0x03FF
8	1	0	0	0	ШИМ ФЧК	ICR1A
9	1	0	0	1	ШИМ ФЧК	OCR1A
10	1	0	1	0	ШИМ ФК	ICR1A
11	1	0	1	1	ШИМ ФК	OCR1A
12	1	1	0	0	СТС	ICR1A
13	1	1	0	1	Зарезервировано	–
14	1	1	1	0	Быстрая ШИМ	ICR1A
15	1	1	1	1	Быстрая ШИМ	OCR1A

Таймер-счетчик T1 может использовать как внешний, так и внутренний тактовые сигналы (табл. 5.8).

Таблица 5.8 – Выбор источника тактового сигнала таймера/счетчика T1

CS12	CS11	CS10	Источник тактового сигнала
0	0	0	Stop условие – таймер/счетчик остановлен
0	0	1	CK
0	1	0	CK / 8
0	1	1	CK / 64
1	0	0	CK / 256
1	0	1	CK / 1024
1	1	0	Внешний тактирующий сигнал на выводе T1, спадающий фронт
1	1	1	Внешний тактирующий сигнал на выводе T1, нарастающий фронт

Таблица 5.9 – Режимы работы выходного сравнения А

COM1A1	COM1A0	Описание
Нормальный режим работы		
0	0	Таймер/счетчик отключен от вывода OC1A
0	1	Переключение выходной линии OC1A
1	0	Вывод сбрасывается в 0
1	1	Вывод устанавливается в 1
Режим быстрой ШИМ		
0	0	Таймер/счетчик отключен от вывода OC1A
0	1	В режиме 15 переключение выходной линии OC1A, иначе таймер/счетчик отключен от вывода OC1A
1	0	Очистка выходной линии OC1A при совпадении, установка при достижении верхнего предела
1	1	Установка выходной линии OC1A при совпадении, сброс при достижении верхнего предела
ШИМ ФК и ШИМ ФЧК		
0	0	Таймер/счетчик отключен от вывода OC1A
0	1	В режимах 9 или 11 переключение выходной линии OC1A, иначе таймер/счетчик отключен от вывода OC1A
1	0	Очистка выходной линии OC1A при совпадении во время счёта вверх, установка при совпадении во время счёта вниз
1	1	Установка выходной линии OC1A при совпадении во время счёта вверх, очистка при совпадении во время счёта вниз

Нормальный режим работы (Normal)

Самым простым режимом работы является нормальный режим. В данном режиме счетчик работает как обычный суммирующий счетчик. Переполнение счетчика происходит при переходе через максимальное 16-разрядное значение (0xFFFF) к нижнему пределу счета (0x0000). В нормальном режиме работы флаг переполнения таймера-счетчика TOV1 будет установлен на том же такте синхронизации, когда TCNT1 примет нулевое значение. В нормальном режиме можно использовать блок захвата. Блок сравнения может использоваться для генерации прерываний.

Режим сброса таймера при совпадении (CTC)

В режиме сброса таймера при совпадении разрешающая способность таймера задается регистрами OCR1A или ICR1. В режиме CTC происходит сброс счетчика (TCNT1), если его значение совпадает со значением регистра OCR1A (режим 4) или с ICR1 (режим 12). В данном режиме обеспечивается более широкий диапазон регулировки частоты генерируемых прямоугольных импульсов. Он также упрощает работу счетчика внешних событий. Временная диаграмма работы таймера в режиме CTC показана на рисунке 5.2. Счетчик (TCNT1) инкрементирует свое состояние до тех пор, пока не возникнет совпадение со значением OCR1A или ICR1, а затем счетчик сбрасывается.

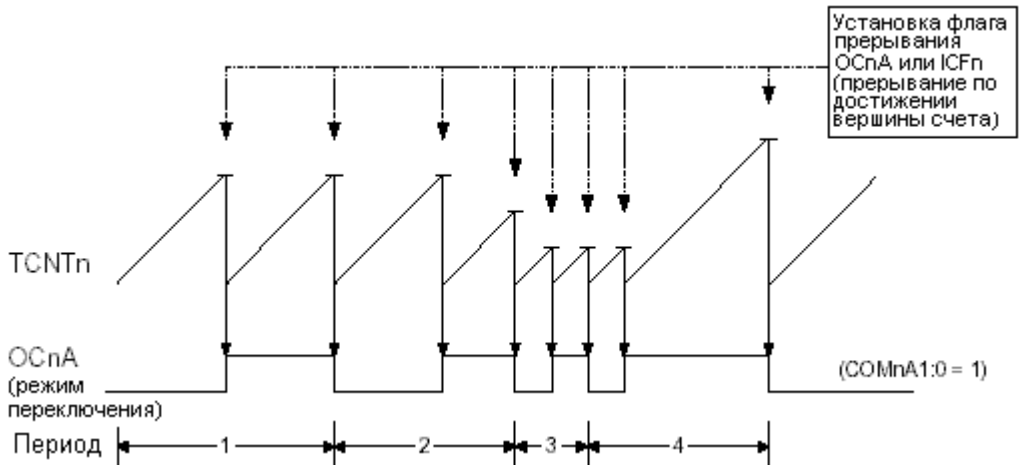


Рис. 5.2 – Временная диаграмма для режима CTC

Для генерации сигнала в режиме CTC выход OC1A может использоваться для изменения логического уровня при каждом совпадении, для чего необходимо задать режим переключения (COM1A1, COM1A0 = 0b01). Значение OC1A будет присутствовать на выводе порта, только если он настроен на выход. Частоту генерируемого сигнала можно определить по формуле $f_{OC1A} = \frac{f_{clk}}{2 \cdot N \cdot (1 + OCR1A)}$, где переменная N задает коэффициент деления предделителя (1, 8, 32, 64, 128, 256 или 1024).

Режим быстрой ШИМ (Fast PWM)

Режим быстрой широтно-импульсной модуляции (ШИМ) предназначен для генерации ШИМ-импульсов повышенной частоты. В отличие от других режимов работы в этом используется однонаправленная работа счетчика. Счет выполняется в направлении от нижнего к верхнему пределу счета.

Если задан неинвертирующий режим выхода, то при совпадении TCNT1 и OCR1A сигнал OC1A устанавливается, а на верхнем пределе счета сбрасывается. Если задан инвертирующий режим, то выход OC1A сбрасывается при совпадении и устанавливается на верхнем пределе счета. За счет однонаправленности счета рабочая частота для данного режима в два раза выше по сравнению с режимом ШИМ с фазовой коррекцией, где используется двунаправленный счет. Возможность генерации высокочастотных ШИМ сигналов делает использование данного режима полезным в задачах стабилизации питания, выпрямления и цифро-аналогового преобразования. Высокая частота, при этом, позволяет использовать внешние элементы физически малых размеров (индуктивности, конденсаторы), тем самым снижая общую стоимость системы.

Разрешающая способность ШИМ может быть фиксированной 8, 9 или 10 разрядов или задаваться регистром ICR1 или OCR1A, но не менее 2 разрядов (ICR1 или OCR1A = 0x0003) и не более 16 разрядов (ICR1 или OCR1A = 0xFFFF).

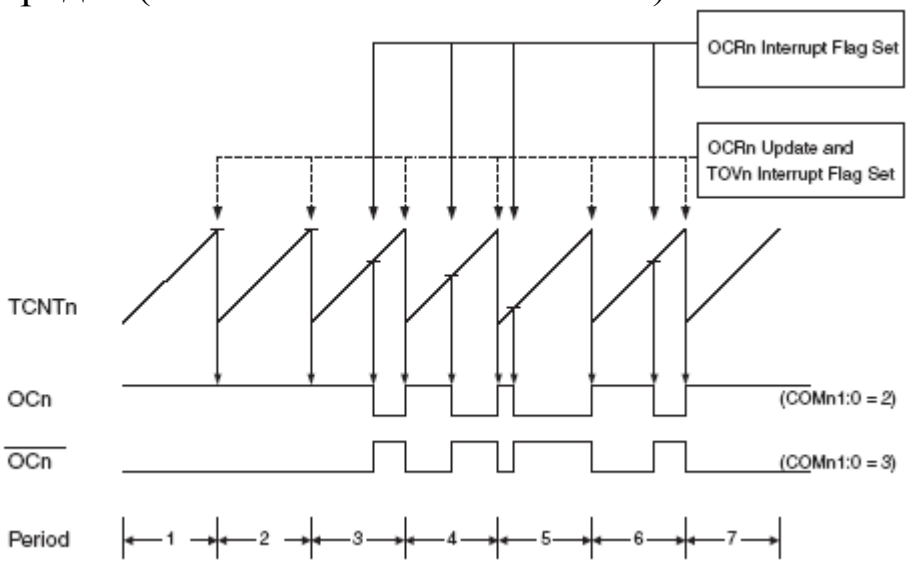


Рис. 5.3 – Временная диаграмма для режима быстрой ШИМ

Временная диаграмма для режима быстрой ШИМ представлена на рис. 5.3. На рисунке показан режим, когда для задания верхнего предела используется регистр OCR1A или ICR1. Значение TCNT1 на временной диаграмме показано в виде графика функции для иллюстрации однонаправленности счета. На диаграмме показаны как инвертированный, так и неинвертированный ШИМ-выходы. Короткой горизонтальной линией показаны точки на графике TCNT1, где совпадают значения OCR1A и TCNT1. Флаг прерывания устанавливается при совпадении. Флаг переполнения таймера-счетчика (TOV1) устанавливается всякий раз, когда счетчик достигает верхнего предела.

Рекомендуется использовать регистр ICR1 для задания верхнего предела, если верхний предел счета является константой. В этом случае также освобождается регистр OCR1A для генерации ШИМ-сигнала на выходе OC1A. Однако если частота ШИМ динамически изменяется (за счет изменения верхнего предела), то в этом случае выгоднее использовать регистр OCR1A для задания верхнего предела, т.к. он поддерживает двойную буферизацию.

Режим ШИМ с фазовой коррекцией (Phase Correct PWM)

Режим широтно-импульсной модуляции с фазовой коррекцией (ШИМ ФК) предназначен для генерации ШИМ-сигнала с фазовой коррекцией и высокой разрешающей способностью. Режим ШИМ ФК основан на двунаправленной работе таймера-счетчика. Счетчик циклически выполняет счет в направлении от нижнего предела (0x0000) до верхнего предела, а затем обратно от верхнего предела к нижнему пределу. При двунаправленной работе максимальная частота ШИМ-сигнала меньше, чем при однонаправленной работе, однако, за счет такой особенности, как симметричность в режимах ШИМ с двунаправленной работой, данные режимы предпочитают использовать при решении задач управления приводами.

Разрешающая способность ШИМ в данном режиме может быть либо фиксированной (8, 9 или 10 разрядов), либо задаваться с помощью регистра ICR1 или OCR1A. Минимальная разрешающая

способность равна 2-м разрядам (ICR1 или OCR1A = 0x0003), а максимальная — 16-ти разрядам (ICR1 или OCR1A =0xFFFF). Временная диаграмма для режима ШИМ ФК представлена на рис. 5.4. На рисунке показан режим ШИМ ФК с использованием регистра OCR1A или ICR1 для задания верхнего предела. Состояние TCNT1 представлено в виде графика функции для иллюстрации двунаправленности счета. На рисунке представлены как неинвертированный, так и инвертированный ШИМ-выход. Короткие горизонтальные линии указывают точки на графике изменения TCNTn, где возникает совпадение со значением OCRnx. Флаг прерывания устанавливается при возникновении совпадения.

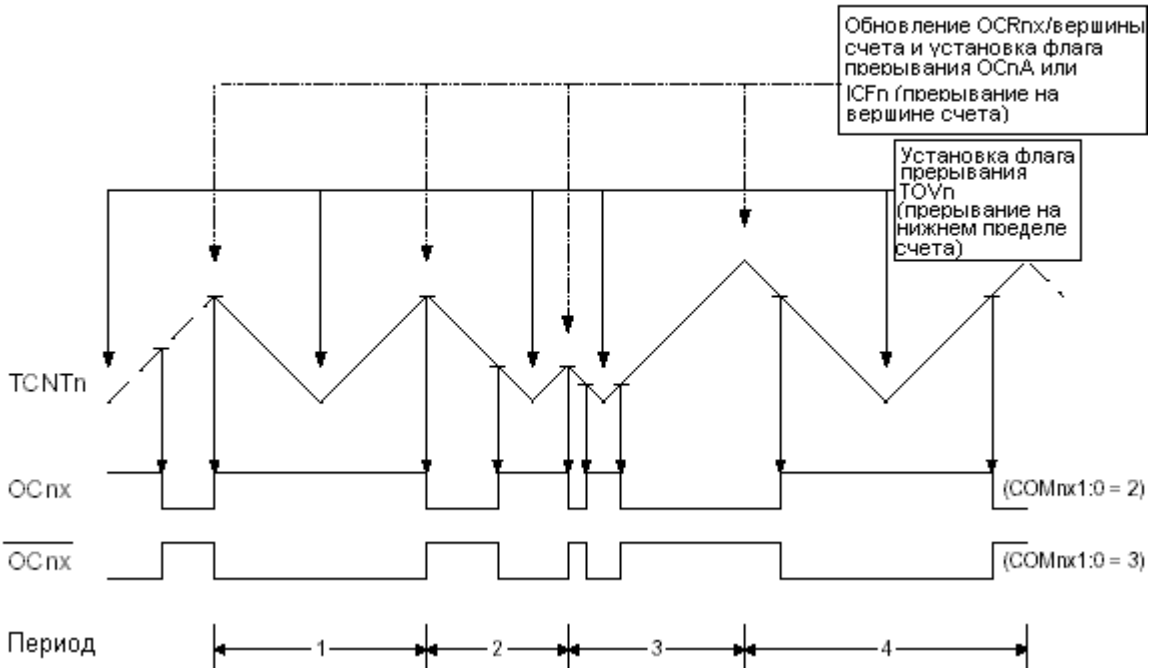


Рис. 5.4 – Временная диаграмма для режима ШИМ ФК

Флаг переполнения таймера-счетчика (TOVn) устанавливается всякий раз, когда счетчик достигает нижнего предела. Если для задания верхнего предела используется регистр OCRnA или ICRn, то, соответственно устанавливается флаг OCnA или ICFn тем же тактовым импульсом, на котором произошло обновление регистра OCRnx из буферного регистра (на вершине счета). Флаги прерывания могут использоваться для генерации прерывания по достижении счетчиком нижнего или верхнего предела.

Если стоит задача изменения верхнего предела при работающем счетчике, то вместо этого режима рекомендуется ис-

пользовать режим ШИМ ФЧК (фазовая и частотная коррекция). Если используется статическое значение верхнего предела, то между данными режимами практически нет отличий.

Режим ШИМ с фазовой и частотной коррекцией

Основное отличие между режимами ШИМ ФК и ШИМ ФЧК состоит в моменте обновления регистра сравнения OCR1A из буферного регистра. Разрешающая способность ШИМ в этом режиме может задаваться с помощью регистра ICR1 или OCR1A. Временная диаграмма для режима ШИМ ФЧК показана на рис. 5.5.

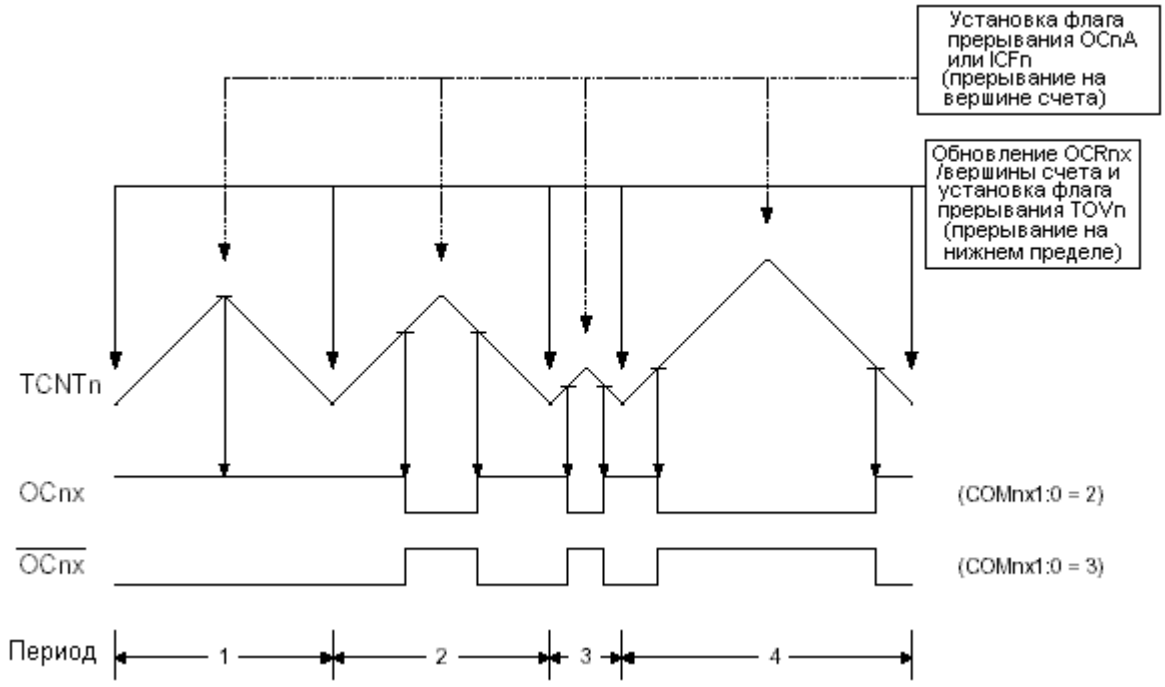


Рис. 5.5 – Временная диаграмма режима ШИМ с фазовой и частотной коррекцией

На рисунке 5.5 показано, что в отличие от режима ШИМ ФК, генерируемый выходной сигнал симметричен на всех периодах. Поскольку, регистры OCR1A обновляются на нижнем пределе счета, то длительности прямого и обратного счетов всегда равны. В результате выходные импульсы имеют симметричную форму, а, следовательно, и откорректированную частоту.

Прерывания от таймеров /счетчиков

Для разрешения/запрещения прерываний от таймеров/счетчиков T0, T1, T2 предназначен регистр TIMSK (табл. 5.10). Для разрешения какого-либо прерывания необходимо установить в 1 соответствующий разряд регистра TIMSK и, разумеется, флаг I регистра SREG. Для индикации наступления прерываний от таймеров/счетчиков T0, T1, T2 предназначен регистр TIFR (табл. 5.11).

Таблица 5.10 – Регистр масок прерываний TIMSK

7	6	5	4	3	2	1	0
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0

OCIEnx – биты разрешения прерывания выходного сравнения;

TICIE1 – бит разрешения прерывания входного захвата;

TOIEn – биты разрешения прерывания по переполнению счётчика.

Таблица 5.11 – Регистр флагов прерываний от таймеров/счётчиков TIFR

7	6	5	4	3	2	1	0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0

OCFnx – флаги прерывания выходного сравнения.;

ICF1 – флаг прерывания входного захвата;

TOVn – флаг прерывания по переполнению счётчика.

Программа работы

1. Рассчитав номер варианта (от N=1 до N=9), загрузить для отладки программу преобразования двоично-десятичного кода числа (байт) в двоичный (программа **decbin_to_bin**).

Перед выполнением программы необходимо в окне «Workspace» загрузить в регистр r16 число 10N+N в двоично-десятичном виде, которое подвергнется преобразованию.

;****** **Программа decbin_to_bin**

```
mov   r17,r16      ; Исходное двоично-десятичное число
                   ; хранится в r16
andi  r17,0xF0     ; Выделение старшей тетрады (десятков)
swap  r17          ;
ldi   r18,10       ; Умножение десятков на десять
mul   r17,r18      ;
mov   r17,r16      ; Выделение младшей тетрады (единиц)
andi  r17,0x0F     ;
add   r17,r0       ; Сложение результатов. Результат остаётся в r17
```

При выполнении программы число копируется в регистр r17, там на него накладывается маска 0b11110000 при помощи команды логического умножения (andi), которая выделяет десятки десятичного кода. После этого результат перемещается в младшую тетраду и умножается на 10. Результат умножения сохраняется в регистровой паре r1:r0. Т.к. наше число не может быть больше 99, то оно целиком поместится в младший байт результата, т.е. в r0. После этого повторно загружаем исходное число в r17, ещё раз накладываем маску, только на этот раз для выделения младшей тетрады, и суммируем результат с содержимым регистра r0, в котором хранились десятки. Конечный результат будет представлен в регистре r17.

Проследите процесс выполнения программы в пошаговом и автоматическом режиме. Поясните содержимое строк окна «Disassembler».

Внести ошибки в программу (неверная мнемоника команды, неверный операнд и т.п.) и проследить сообщения ассемблера при этом в окне **View Output.**

Модифицировать программу в соответствии со своим вариантом индивидуального задания:

1. Преобразовать дополнительный код числа (байт) в прямой.

2. Преобразовать двоичный код (от 0 до 99) в двоично-десятичный.

3. Просуммировать два числа в двоичном коде. Сумму, большую 255, заменить байтом единиц.

4. Сложить два двухбайтовых числа.

5. Вычесть два числа в двоичном коде. Разность, меньшую нуля, заменить байтом нулей.

6. Умножить на два двухбайтовое содержимое регистров R9..R10 (меньшее 32 000).

7. Сложить два десятичных числа (байт) в двоично-десятичном коде.

8. Реализовать суммирующий двоично-десятичный счетчик.

9. Реализовать вычитающий двоично-десятичный счетчик.

2. Загрузить для отладки программу определения максимального элемента массива 8-разрядных чисел SRAM (программа **max_el_mass).**

Перед выполнением программы необходимо заполнить область памяти данных случайными числами, начиная с адреса 0x60. Количество элементов определяется числом, загружаемым в начале программы в r18.

Для обращения к памяти с адресом более 8 разрядов программа использует специально для этого предназначенный Z-регистр. Он состоит из пары регистров r31:r30. В начале выполнения программа копирует первый элемент массива в r16, а адрес этого элемента в X (регистровая пара r27:r26). После этого каждый элемент сравнивается со значением в r16, и в том случае, если элемент окажется больше содержимого r16, то он замещает собой предыдущее значение в r16, а также адрес нового наибольшего элемента копируется в пару X. Каждый раз при сравнении, из количества элементов вычитается единица. Как только содержимое r18 станет равным 0, выполнение программы можно прекращать в связи с перебором всех элементов. В итоге получим наибольший элемент в r16, его адрес в X.

;****** **Программа max_el_mass**

```
ldi r30,low($100)  ;Загрузка в регистр Z начального адреса
ldi r31,high($100) ;массива чисел
ldi r18,12         ;Загрузка количества элементов массива
ld r16,z           ;Загрузка в регистр r16 первого элемента массива
mov r26,r30        ;Загрузка в X адреса первого элемента
mov r27,r31
dec r18
```

```
m1:   inc r30          ;Увеличение Z на единицу для загрузки
                       ; следующего элемента
      ld r17,z         ;Загрузка элемента для сравнения
      cp r16,r17       ;Сравнение максимального (или первого) элемента
                       ;с только что загруженным
      brsh m2
      mov r16,r17      ;его перезапись в r16 в случае, если больше
      mov r26,r30      ;и сохранение адреса в X
      mov r27,r31
m2:   dec r18          ;Уменьшение счётчика количества элементов
      brbc 1,m1        ;Если не все элементы перебраны, переход на m1
m3:    jmp m3          ;Зацикливание по завершении
```

Модифицировать программу в соответствии со своим вариантом индивидуального задания:

1. Определить минимальный элемент массива SRAM.

2. Сформировать массив 3, элементы которого определяются как разность соответствующих элементов массивов 1 и 2. Отрицательную разность заменить нулем.

3. Сформировать массив 3, элементы которого определяются как сумма соответствующих элементов массивов 1 и 2. Сумму, большую 255, заменить байтом единиц.

4. Количество одинаковых элементов массивов 1 и 2 поместить в регистр R0.

5. Количество чисел массива 1, совпадающих с содержимым регистра R1, поместить в регистр R0.

6. Двоичные числа массива преобразовать в двоично-десятичные.

7. Двоично-десятичные числа массива (меньшие 99) преобразовать в двоичные.

8. Сформировать массив 2, элементы которого представляют дополнительный код восьмиразрядных чисел со знаком массива 1.

9. Отсортировать массив по возрастанию элементов.

3. Набрать и отладить прикладную программу **fairy**, позволяющую получить эффект бегущей 1 на линиях порта.

Ввести код программы в отладчик AVRStudio и проверить ее работу в пошаговом режиме. Проследить изменения, происхо-

дящие в регистрах SREG, PORTB и PINB по мере выполнения программы. Для чего в регистр DDRB заносится 0xFF? Чем отличается команда rol от команд lsl и asr?

;****** **Программа fairy**

```
.INCLUDE "C:\Program Files\Atmel\AVR
        Tools\AvrAssembler\Appnotes\m8def.inc"
        ; Подключение файла определения адресов

                ldi     r16,0xFF
                out     DDRB,r16
                sec
                clr      r16
m1:             rol      r16
                out     PORTB,r16
                rjmp    m1
```

Дополните программу подпрограммой задержки «Delay» таким образом, чтобы содержимое r17 определяло длительность паузы между сменой состояния на выводах порта.

Примечание: для вызова подпрограммы необходимо активировать стек, что происходит автоматически при указании его начала в паре регистров sph:spl.

Модифицировать программу в соответствии со своим вариантом индивидуального задания и произвести ее отладку:

1. Мультивибратор (тетрады порта D заполняются либо единицами, либо нулями в цикле, скорость миганий можно изменять с помощью порта B).
2. Бегущий огонек со сменой направления на линиях порта C.
3. Елочка 1 (линейка светодиодов, подключенная к линиям порта B, последовательно заполняется огнями и затем гаснет, после чего эффект периодически повторяется).
4. Бегущий огонек на линиях порта C должен сменить направление, если на пяти линиях порта B установлены логические единицы.

5. Елочка 2 (линейка светодиодов, подключенная к линиям порта, последовательно заполняется огнями и постоянно горит, а звезда – старший бит – моргает).

6. Если на выводы порта D пришло число с нечётным количеством единиц, то оно передаётся через порт B, если с чётным — через порт C.

7. «Бегущий огонёк» в одну сторону бежит по выводам порта B, в другую — порта B.

8. Одна тетрада поступающего на выводы порта D числа должна быть отправлена через порт B, другая через C.

9. Если поступившее на выводы порта D число чётное, то должны «мигать» выводы порта B, если нет – то порта C.

4. Загрузить для отладки программу генерации сигнала заданной частоты (программа **Generator)**.

```
;****** Программа Generator
; Генератор импульсов с программируемым периодом T=2n(1+X), где
; X – число в регистре OCR1A, n – коэффициент деления предделителя

.INCLUDE  "C:\Program Files\Atmel\AVR
          Tools\AvrAssembler\Appnotes\m8def.inc"
      ldi       r16,0x02          ; Линию PB1 на вывод
      out       DDRB,r16          ;
      ldi       r16,0b01000000    ; Режим CTC таймера T1, состояние
                                  ; вывода PB1 при сравнении меняется
      out       TCCR1A,r16        ; на противоположное
      ldi       r16,0b00001001    ; Режим CTC таймера T1 на частоте
      out       TCCR1B,r16        ; тактирования   (n=1)
      ldi       r16,99            ; Модуль счета X=99 для
      out       OCR1AL,r16        ; периода T=200 тактов
m1:   rjmp      m1
```

Изменить программу так, чтобы период генерируемых прямоугольных импульсов на выводе PB1 составил N секунд при использовании кварцевого резонатора на 16 МГц (N — вариант задания).

5. Составить комментарий к программе **PWM2.**

```
;****** Программа PWM2
```

```
.INCLUDE  "C:\Program Files\Atmel\AVR
           Tools\AvrAssembler\Appnotes\m8def.inc"
          ldi     r16,0x02
          out     DDRB,r16
          ldi     r16,0x7F
          out     OCR1AL,r16
          ldi     r16,0b11000001
          out     TCCR1A,r16
          ldi     r16,0b00000010
          out     TCCR1B,r16
  m1:     rjmp    m1
```

Ввести код программы в отладчик AVR Studio и проверить ее работу в пошаговом режиме. Какой режим ШИМ выбран в данном случае? Какова относительная длительность импульсов на выводе OC1A? С какой частотой относительно частоты кварца поступают импульсы синхронизации на таймер/счётчик?

Контрольные вопросы

- Чем ограничен размер массива?
- Перечислите все методы адресации памяти данных.
- Разрешены ли прерывания после системного сброса?
- Для чего нужен регистр Input capture (ICR)? Назовите ситуации, когда необходимо его использование. Назовите его альтернативные функции.
- Что должно произойти при достижении счётчиком значения 0x01FF, если в битах WGMn3:1 записано 0b0010? 0b0110? Как при этом поведёт себя OCnx?
- Что может выступать в качестве источника импульсов синхронизации таймеров/счётчиков?

Содержание отчета

Отчет должен содержать листинги отлаживаемых программ (в том числе и по индивидуальным заданиям), комментарии по ходу выполнения пунктов работы и рисунки, вставляемые в текст формата WORD, отображающие окна регистров и памяти, а также ответы на контрольные вопросы.

6 СРЕДСТВА РАЗРАБОТКИ ПРОГРАММЫ НА ЯЗЫКЕ СИ, КОМПИЛЯТОРЫ И СИМУЛЯТОРЫ

При проектировании микропроцессорного устройства вы определились с тем, что оно должно делать, нарисовали схему устройства, физически способную выполнить вашу задачу. Программу для МК удобно создавать с помощью специальных программных средств – компиляторов. Компилятор позволяет написать программу для МК на универсальном языке программирования Си (кстати, для МК требуется всего 3-5% всех его возможностей).

Текст программы, набранный в компиляторе, называют исходным кодом. Компилятор проверяет отсутствие ошибок в набранном исходнике и, если ошибок нет, преобразует исходник (компилирует его) в специальный файл обычно с расширением .hex – его называют "прошивка".

Прошивку с помощью программатора (для AVR это, например, пять проводков с параллельного порта ПК) помещают во FLASH-память программ МК и при необходимости частично в его EEPROM.

Очень трудно написать программу сразу правильно и без ошибок. Поэтому важнейшим этапом разработки электронного устройства является отладка программы МК.

Для отладки вы включаете устройство с прошитым МК, находите отклонения от требуемого алгоритма, выявляете ошибки, вносите соответствующие изменения в исходный текст программы и опять компилируете, прошиваете новый .hex в МК и так до победного конца – т.е. до тех пор, пока устройство не заработает так, как вам нужно.

Не всегда допустимо включить устройство, не зная наверняка, правильно ли работает программа МК – в некоторых случаях могут произойти серьезные и дорогостоящие повреждения обвязки МК и другой аппаратуры. Иногда требуется проверить работу МК, не имея вообще реально спаянной схемы и самого МК. В этих случаях рекомендуется использовать специальные программные средства – симуляторы.

Симулятор приблизительно моделирует на персональном компьютере (ПК) работу "прошитого" вашей программой МК и его обвязки – т.е. электронных компонентов, окружающих МК по схеме устройства.

Кроме того, симуляторы позволяют:

- останавливать программу, организуя точки останова;
- выполнять программу по шагам;
- видеть, как именно происходит выполнение программы;
- наблюдать и изменять значения в регистрах МК;
- наблюдать текущие значения переменных;
- использовать виртуальные измерительные приборы;
- симулировать работу МК с обвязкой, включающей различные электронные компоненты и устройства;
- виртуально обмениваться информацией с терминалом на ПК;
- делать еще много полезного!

Для начинающих наиболее удобен в работе компилятор Си для AVR CodeVisionAVR (или CVAVR). Компилятор содержит очень понятный и мощный генератор начального кода программы по вашим потребностям в конфигурации периферии МК AVR – называется он CodeWizardAVR (будем называть его мастером).

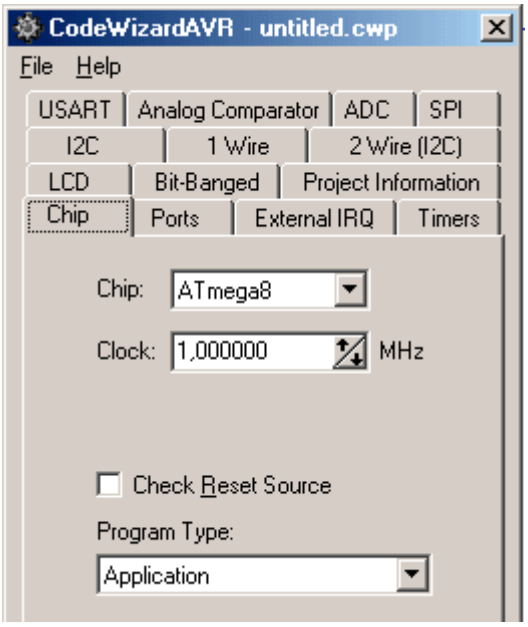


Рис. 6.1 – Окно мастера создания нового проекта

Вам нужно просто выбрать МК, частоту тактирования, затем открыть ярлыки тех устройств МК, которые будете использовать, и установить нужные параметры их работы.

Затем мастер создаст файл проекта .prj и файл исходного текста программы на языке Си с расширением .c – в нем уже будет содержаться код на Си, конфигурирующий МК по сделанному вами "заказу". Вам нужно будет добавить лишь код, реализующий нужный вам алгоритм работы устройства.

Используйте этот мастер и далее по ходу работы программы – точно так же как и в начале, но не генерируйте новые файлы, а просто откройте меню "Program Preview" и посмотрите нужный кусок программы на Си, возьмите, что вам нужно и вставьте в вашу программу.

Компилятор CodeVisionAVR имеет и встроенный программный модуль для прошивки МК и конфигурационных ячеек (Fuse Bits - фьюзов) прямо в схеме. Фьюзы расположены в отдельном адресном пространстве, доступном только при программировании.

Для полного цикла разработки устройства достаточно только одного компилятора CodeVisionAVR. В компиляторах есть отличные примеры программ на Си для наиболее часто встречающихся задач на МК. В CodeVisionAVR примеры находятся в папке CVAVR\Examples. Примеры – это исходные тексты программ на Си для управления периферией МК и интерфейса с популярными внешними устройствами. Исходники снабжены подробными комментариями. Комментарии – это то, что написано после двух косых черточек // – в одну строку, либо между /* и */ – в одну или несколько строк.

Вы инсталлировали компилятор Си для AVR CodeVisionAVR и получили файл-прошивку для МК. Естественно, хотите узнать – будет ли прошивка, а значит ваша программа, работать в МК так, как вам нужно. Для этого удобно использовать специальные программы для ПК, называемые симуляторами. Вы можете проверить работу не только программы, загруженной в модель МК, но и работу модели целого электронного устройства!

Установите на ПК симулятор для AVR – VMLAB. В поставку VMLAB включено множество примеров программ и про-

шивок для немедленной симуляции-эмуляции устройства с МК. Примеры находятся в папках: Tutorial и AVR_demo. Есть пример работы VMLAB с CodeVisionAVR. Откройте файл проекта C:\VMLAB\AVR_demo\codevisi.prj и посимулируйте. При симуляции вы увидите движение по исходному тексту программы на Си, и можете расставлять точки останова программы, наблюдать за изменениями значений в регистрах МК, посмотреть осциллограммы сигналов на ножках МК и многое другое.

На симуляторе VMLAB мы будем проверять работу программы для разрабатываемого устройства. Файл-прошивку для МК (расширение .hex), созданный в компиляторе, будем прогонять в симуляторе МК с внешними компонентами и смотреть, что работает, что нет, и как работает. При необходимости будем корректировать исходный текст программы на Си, опять компилировать, и так по кругу до достижения правильной работы устройства. Этот процесс и называется отладкой программы.

7 ЯЗЫК СИ ДЛЯ МИКРОКОНТРОЛЛЕРОВ

Минимальная программа на Си может быть такой: **main(){}**. Эта программа не делает ничего полезного – но это уже программа и она показывает, что в программе на языке Си должна быть главная функция main – обязательно. Реальные программы на Си конечно больше.

Регистры МК в программе на Си имеют названия, как и в оригинальной технической документации фирмы ATMEL AVR Data Sheets (ДШ) и так как числа в большинстве из них можно менять – для программы регистры являются по сути переменными.

Чтобы поместить число в переменную (в регистр), в Си есть **оператор присваивания**. Это знак = (называемый в математике "равно"). В Си этот знак не означает равенство. Знак = в Си означает вычислить результат того, что справа от оператора присваивания и поместить этот результат в переменную, находящуюся левее оператора присваивания. Ниже приведены примеры команд на Си, использующие оператор присваивания.

PORTB = PINB + 57; /*Взять (прочитать, считать) значение переменной (регистра) PINB, затем прибавить к нему число 57 и поместить результат в переменную PORTB */

PORTB&=0x5A; /*Прочитать значение переменной PORTB, затем выполнить "поразрядное (побитное) логическое И" между прочитанным значением и числом 0x5A и записать результат в переменную PORTB */

PORTB = 0x23; /*Не читая содержимое переменной PORTB присвоить ей значение 0x23 */

Вместо & "И" могут быть и другие побитные логические операции: | "ИЛИ", ^ "Исключающее ИЛИ", ~ "инвертирование битов" и арифметические операции: + - * / %.

Запомните! Результатом поразрядных (побитных) логических операций (& | ^ ~) является число, которое может быть интерпретировано компилятором как "истина", если оно не ноль, и "ложь", если число ноль.

Целые числа в компиляторе могут быть записаны:
- в десятичной форме: 1234;
- в двоичной форме с префиксом 0b: 0b101001;
- в шестнадцатеричной форме с префиксом 0x: 0x5A;
- в восьмеричной форме с префиксом 0: 0775.

С оператором присваивания используются вот такие сокращения:

Длинная запись	Смысл	Сокращается до
x = x + 1;	добавить 1	x++; или ++x;
x = x - 1;	вычесть 1	x--; или --x;
x = x + y;	прибавить y	x += y;
x = x - y;	вычесть y	x -= y;
x = x * y;	умножить на y	x *= y;
x = x / y;	поделить на y	x /= y;
x = x % y;	остаток от деления	x %= y;
x--;	вычесть 1	x -= 1;
x++;	добавить 1	x += 1;

Есть в Си операции, которые изменяют значение переменной и без оператора присваивания:

PORTA++; /* Взять значение переменной PORTA, добавить к ней 1 и записать результат обратно в PORTA – инкрементировать регистр PORTA */

PORTC--; /* Эта строчка на Си означает обратное действие – декрементировать значение регистра PORTC */

Инкремент и декремент удобно использовать для изменения значения различных переменных-счетчиков. Важно помнить, что они имеют очень низкий приоритет. Поэтому, чтобы быть уверенным в порядке выполнения, желательно писать их отдельной строчкой программы.

Когда инкремент или декремент используется в выражении, то важно, где стоят два знака + или – (перед переменной или после переменной):

```
A=4;
B=7;
A=B++;  /* Взять значение переменной B, присвоить его переменной A, затем добавить 1 к переменной B и сохранить результат в B. Теперь A будет содержать число 7, B будет содержать число 8  */
A=4;
B=7;
A=++B;  /* Взять значение переменной B, затем добавить к нему 1 и сохранить результат в B и этот же результат присвоить переменной A. Теперь A будет содержать число 8 и B будет содержать число 8  */
```

Арифметические операции в Си:

```
x+y          //сложение
x-y          // вычитание
x * y        // умножение
x / y        /* деление. Если числа целые, результат – целое число с отброшенной дробной частью – не округленное! Т.е. если в результате деления на калькуляторе получается 6.23411 или 6.94, то результат будет просто целое число 6. Если числа с плавающей точкой, то есть float или double и записываются с точкой
```

и числом после точки, то и результат будет число с плавающей точкой */

```
x % y        // вычислить остаток от деления нацело
```

Примеры:

```
5 / 2        // даст  2
5 % 2        // даст 1
75 / 29      // даст 2
75 % 29      // даст 17
```

Операторы сравнения (или отношения) используются для сравнения переменных, чисел (констант) и выражений:

```
x < y        // x меньше y
x > y        // больше
x <= y       // меньше или равно
x >= y       // больше или равно
x == y       // равно
x != y       // не равно
```

Результат выполнения этих операторов: "истина" это "1" (точнее "не ноль"), "ложно" это "0". Значения, хранимые в переменных (в регистрах) x и y, не изменяются!

Логические операции:

```
||           // "ИЛИ"
&&           //  "И"
!            // "НЕ"
!(истина)  // дает "ложь"
!(ложь)    // дает "истина"
```

В результате логической операции вы получаете не число, а логическое значение "истина" или "ложь". Для логических операций && и || берутся результаты выражений слева и справа от знака операции, преобразованные в "истину" или "ложь", и определяется логический результат операции. Компилятор результат "истина" превращает в 1, а "ложь" в 0.

Ходовые конструкции на Си (в компиляторе CVAVR заготовки этих конструкций находятся под ярлыком "Code Templates" слева вверху. Вы можете выбирать нужные заготовки и вставлять их в свою программу):

if(){}else{}; идеальная конструкция, если вам нужно выполнить какую-то часть программы при наличии каких либо условий:

```
if (выражение) { /* делать этот код, если выражение
"истина" – т.е. результат его вычисления не ноль */
                }
else { /* делать этот код, если выражение "ложь" – т.е.
результат его вычисления равен нулю */
            };
```

} else { это не обязательный элемент конструкции:

```
if (выражение) { /* делать этот код, если выражение
"истина" – т.е. результат его вычисления не ноль */
                };
```

while(){}; условный цикл – используйте, если вам нужно выполнять какой то код программы, пока выполняется (существует, справедливо, не ноль) некоторое условие:

```
while (выражение) { /* делать этот код, если выражение "истина" – т.е. результат его вычисления не ноль. Пока выполняется этот код, выражение не проверяется на истинность. После выполнения кода происходит переход к строке while, чтобы снова проверять истинность выражения */
                };
```

Цикл while имеет вариант do – while, при котором код в { } выполняется по меньшей мере один раз независимо от истинности условия в скобках:

```
do { /* сделать этот код один раз, затем, если выражение есть
"истина" – т.е. результат его вычисления не ноль – опять делать
код с начала, и так до тех пор, пока выражение "истина"  */
}
while (выражение);
```

for(;;){}; – этот цикл позволяет выполнить часть программы нужное число раз:

```
char i; /* объявление переменной для for. Это обычная переменная и, значит, может иметь любое  допустимое имя по вашему желанию */
for (i=5;i<20;i+=4) {  /*  код цикла for. i=5 – это начальное выражение. Число 5 просто для примера, может быть таким, как позволяет объявление переменной i, в нашем случае от 0 до 255. i<20 – контрольное выражение. Может быть с разными операторами отношения, важно лишь, чтобы по ходу цикла оно становилось когда-то "ложью" – иначе цикл "зациклится", т.е. никогда не кончится. i+=4 – счетчик. Обычно это i++, т.е. к переменной добавляется 1 каждый "прогон" цикла. Но может быть таким, какое вам требуется, важно лишь достижение когда-либо условия, оговоренного выше! Иначе цикл станет бесконечным. Код цикла for будет первый раз выполнен для i=5, затем по выражению i+=4 i станет 9. Теперь будет проверено контрольное выражение i<20 и так как 9<20 код цикла for будет выполнен еще раз. Так будет происходить до тех пор, пока контрольное выражение "истинно". Когда оно станет "ложно" цикл for закончится и программа пойдет дальше.  */
        };
```

Начальным условием может быть любое допустимое в Си выражение, результатом которого является целое число. Контрольное выражение определяет, до каких пор будет выполняться цикл. Счетчик показывает, как изменяется начальное выражение перед каждым новым выполнением цикла.

Циклы for(;;) и while() часто используют вот так:

```
while(1);
for (;;);
/* Так написанные эти циклы означают: МК выполнять эту строчку пока есть питание, нет сброса и нет прерывания. Когда возникает прерывание, программа переходит на обработчик прерывания и (если в обработчике нет перехода в другое место программы) по завершении кода обработчика опять возвращается в такой цикл */
```

switch(){}; – оператор множественного выбора, позволяет сделать выбор из нескольких вариантов.

```
switch (выражение) {
case 7:
/* этот код будет выполняться, если результат вычисления выражения равен числу 7. На этом работа оператора switch закончится */
break;
case -28:
/* этот код будет выполняться, если результат вычисления выражения равен отрицательному числу -28. На этом работа оператора switch закончится */
break;
case 'G':
/* этот код будет выполняться, если результат вычисления выражения равен числу, соответствующему символу G в таблице ASCII. На этом работа оператора switch закончится */
break;
default:
/* этот код будет выполняться, если результат вычисления выражения не равен ни 7, ни -28, ни 'G'. А так же после выполнения кода, не имеющего в конце  break. На этом работа оператора switch закончится */
};
 /* switch закончен - выполняется дальнейший код программы */
```

case может быть столько, сколько вам нужно. Чтобы программа работала быстрее, старайтесь наиболее вероятные варианты располагать выше!

default - не обязателен. Его можно расположить и не в конце.

break; - если его не использовать, то, найдя нужный вариант, программа будет выполнять и следующие ниже условия case.

goto – оператор безусловного (немедленного) перехода.

```
mesto_5: /* сюда мы попадем после выполнения строки программы goto mesto_5   */
goto mesto_1; /* перейти в то место программы, где в начале строки написано mesto_1:     */
```

```
goto mesto_5; /* перейти в то место программы, где в начале строки написано mesto_5:     */
mesto_1: /* сюда мы попадем после выполнения строки программы goto mesto_1   */
```

goto существует наверно во всех языках и в ассемблере в том числе. Используйте его с осторожностью! Например: если вы покинете функцию-обработчик прерывания по goto, не завершив ее, то не произойдет автоматического включения прерываний глобально – т.е. не установится бит I в регистре SREG, Этот бит устанавливается автоматически после полного выполнения функции обработки прерывания и "естественного" выхода из неё.

Структура программы на языке Си

Программа на Си имеет определенную структуру:
1) заголовок;
2) включение необходимых внешних файлов;
3) ваши определения для удобства работы;
4) объявление глобальных переменных (глобальные переменные объявляются вне какой-либо функции, т.е. не после фигурной скобки {, доступны в любом месте программы, значит можно читать их значения и присваивать им значения там, где требуется);
5) описание функций-обработчиков прерываний;
6) описание других функций, используемых в программе;
7) функция main (это единственный обязательный пункт).

Функция имеет { "тело" } в фигурных скобках. Тело – это код на Си, определяющий то, что делает функция. Знак ; после функции не ставится.

Программа на Си начинает работу с функции main(), по необходимости из main() вызываются другие функции программы, по завершении работы функции программа возвращается в main() в то место, откуда функция была вызвана.

```
main(){
... какой то код программы ...
вызов функции_1; //программа перейдет в функцию_1
```

строка программы; // будет выполняться после возврата
... какой то код программы ...
}

Функции могут вызываться не только из main(), но и из других функций. Кроме того, описанный выше ход программы может нарушаться прерываниями.

Приведем пример программы на Си с описанной выше структурой (текст в рамке). По мере надобности программа будет разрываться обычным текстом, а затем продолжаться.

```
/*  Пункт 1. Заголовок программы
Он оформляется как комментарий, и обычно содержит информацию:
- о названии, назначении, версии и авторе программы;
- краткое описание алгоритма программы;
- пояснения о назначении выводов МК;
- другие сведения, которые вы считает полезным указать.
*/
// комментарий после двух косых черт пишут в одну строку!

// Пункт 2. Включение внешних файлов

#include <mega16.h> /* перед компиляцией, препроцессор компилятора вставит вместо этой строчки содержимое (текст) заголовочного файла mega16.h - этот файл содержит перечень регистров, имеющихся в МК ATmega16, и соответствие их названий их физическим адресам в МК. Посмотрите его содержание, вызвав CVAVR\inc\mega16.h    */

//delay functions
#include <delay.h>
/* перед компиляцией, препроцессор компилятора вставит вместо этой строчки текст "хидера" delay.h - этот файл содержит функции для создания пауз в программе.
Теперь чтобы сделать паузу вам нужно лишь написать:
 delay_us(N); // сделать паузу N (число) мкс
 delay_ms(x); // сделать паузу x мс
```

```
        x - может быть переменная или число от 0 до 65535
(тип unsigned int), например, delay_ms(peremennaya)*/

// Пункт 3. Определения пользователя
// AD7896 control signals PORTB bit allocation
#define ADC_BUSY PINB.0
#define NCONVST PORTB.1

/* после этих двух строк, перед компиляцией, препроцессор
компилятора заменит в тексте программы ADC_BUSY на
PINB.0 и NCONVST на PORTB.1. Таким образом, вместо то-
го, чтобы помнить, что вывод занятости AD7896 подключен
к ножке PB0, вы можете проверять значение осмысленного
понятия ADC_BUSY - "АЦП занят", а вместо управления аб-
страктной ножкой PB1 (через PORTB.1) можете управлять
"НьюКонвекшнСтат" - NCONVST - "стартовать новое АЦ
преобразование"
#define  –  Это  удобно, но вовсе не обязательно. */
```

Пункт 4. Объявление переменных

Перед использованием переменной в программе на Си её необходимо объявить, т.е. указать компилятору, какой тип данных она может хранить и как она называется.

Формат объявления переменной таков:

[<storage modifier>] <type definition> <identifier>;

[<storage modifier>] – необязательный элемент,
он нужен только в некоторых случаях и может быть:
extern – если переменная объявляется во внешнем файле, например, в хидере delay.h, приведенном выше;

volatile – ставьте, если нужно предотвратить возможность повреждения содержимого переменной в прерывании, и не позволить компилятору попытаться выкинуть её при оптимизации кода.

Пример:
volatile unsigned char x;

static – если переменная локальная, т.е. объявлена в какой либо функции и должна сохранять свое значение до следующего вызова этой функции.

eeprom – разместить переменную в EEPROM. Значение таких переменных сохраняется при выключении питания и при перезагрузке МК.
Пример:
eeprom unsigned int x;
Если это первая переменная в EEPROM, то её младший байт будет помещен в ячейку 1 EEPROM, а старший в ячейку 2. Необходимо помнить, что запись в EEPROM длительный процесс - 8500 тактов процессора.

Глобальные переменные объявляются до появления в тексте программы какой либо функции. Глобальные переменные доступны в любой функции программы.

Локальные переменные объявляются в самом начале функций, т.е. сразу после фигурной скобки { . Локальные переменные доступны только в той функции, где они объявлены!

<type definition> - тип данных, которые может хранить переменная.

Наиболе часто используемые типы данных:
unsigned char - хранит числа от 0 до 255 (байт);
unsigned int - хранит числа от 0 до 65535 (слово == 2 байта);
unsigned long int - хранит от 0 до 4294967295
(двойное слово == 4 байта).

Вместо **unsigned char** можно писать просто **char**, так как компилятор по умолчанию считает **char** беззнаковым байтом. А если вам нужен знаковый байт, то объявляйте его так:

signed char imya_peremennoi**;**

<identifier> – имя переменной - некоторый набор символов по вашему желанию, но не образующий зарезервированные слова

языка Си. Выше был уже пример идентификатора – имени переменной: imya_peremennoi.

Желательно давать осмысленные имена переменным и функциям, напоминающие вам об их назначении. Принято использовать маленькие буквы, а для отличия имен переменных от названия функций имена переменных можно, например, начинать с буквы, а названия функций (кроме main конечно) с двух символов подчеркивания.

Например, так**:** moya_peremennaya , __vasha_funkziya.

Глобальные переменные, а также локальные с модификатором static - при старте и рестарте программы равны 0, если вы не присвоили им (например, оператором =) иное значение при их объявлении или по ходу программы.

Вот несколько примеров объявления переменных:

```
unsigned char my_peremen = 34;
unsigned int big_peremen = 34034;
```

Пример массива, содержащего три числа или элемента массива.

```
char mas[3]={11,22,33};
```

Нумерация элементов начинается с 0, т.е. элементы данного массива называются mas[0], mas[1], mas[2] и в них хранятся десятичные числа 11, 22 и 33.

Где-то в программе вы можете написать: mas[1] = 120;

Теперь в mas[1] будет храниться число 120. Можно не присваивать значений элементам массива при объявлении, но только при объявлении вы можете присвоить значения всем элементам массива сразу. Потом это можно будет сделать только индивидуально для каждого элемента.

Строковая переменная или массив, содержащий строку символов.

```
char stroka[6]="Hello"; /*  Символов (букв) между кавычками  5 , но указан размер строки 6. Дело в том, что строки символов должны заканчиваться десятичным числом 0. Не путайте его с символом '0', которому соответствует десятичное число 48 по таблице ASCII, которая устанавливает каждому числу определенный символ  */
```

Например:

Элемент строки stroka[1] содержит число 101, которому по таблице ASCII соответствует символ 'e'.

Элемент stroka[4] содержит число 111, которому соответствует символ 'o'.

Элемент stroka[5] содержит число 0, которому соответствует символ 'NUL', его еще обозначают вот так '\0'.

Строковая переменная может быть "распечатана" или выведена в USART МК вот так: printf("%s\n", stroka);

flash и **const** ставятся перед объявлением констант, неизменяемых данных, хранящихся во flash-памяти программ. Они позволяют использовать не занятую программой память МК. Обычно для хранения строковых данных – различных информационных сообщений, либо чисел и массивов чисел.

Примеры:

```
flash int integer_constant=1234+5;
flash char char_constant=’a’;
flash long long_int_constant1=99L;
flash long long_int_constant2=0x10000000;
flash int integer_array1[ ]={1,2,3};
flash int integer_array2[10]={1,2};
flash char string_constant1[ ]=”This is a string constant”;
const char string_constant2[ ]=”This is also a string constant”.
```

```
// Пункт 5. Описание функций-обработчиков прерываний

/* мы будем использовать в этой программе только одно
прерывание и значит одну функцию-обработчик прерывания.
Программа будет переходить на неё при возникновении пре-
рывания: ADC_INT - по событию "окончание АЦ преобразо-
вания" */

interrupt [ADC_INT] void adc_isr(void)
{
PORTB=(unsigned char) ~(ADCW>>2);
/* отобразить горящими светодиодами, подключенными
```

от + питания МК через резисторы 560 Ом к ножкам порта B, старшие 8 бит результата аналого-цифрового преобразования.

Сделаем паузу 127 мс, чтобы в реальном устройстве можно было увидеть переключение светодиодов */
delay_ms(127);

/* В реальных программах старайтесь не делать пауз в прерываниях! Обработчик прерывания должен быть как можно короче и быстрее */

// начать новое АЦ преобразование
ADCSRA|=0x40;

} // закрывающая скобка обработчика прерывания

Функция обработчик прерывания может быть названа вами произвольно, как и любая функция, кроме main. Здесь она названа adc_isr. При каком прерывании ее вызывать компилятор узнает из строчки interrupt[ADC_INT]. По первому зарезервированному слову - interrupt - он узнаёт, что речь идет об обработчике прерывания, а номер вектора прерывания (адрес, куда физически, внутри МК, перескочит программа при возникновении прерывания) будет подставлен вместо ADC_INT препроцессором компилятора перед компиляцией - этот номер указан в подключенном нами ранее заголовочном файле ("хидере") описания "железа" МК - mega16.h - это число, сопоставленное слову ADC_INT.

Очень информативна следующая строка программы:
PORTB = (unsigned char) ~(ADCW>>2);

Нужно присвоить значение выражения справа от оператора присваивания той переменной, что указана слева от него. Значит, нужно вычислить выражение справа и поместить его в переменную PORTB. ADCW – это двухбайтовая величина (так она объ-

явлена в файле mega16.h, в котором CodeVisionAVR сохраняет 10-битный результат АЦП, а именно в битах 9_0 (биты с 9-го по 0-й), т.е. результат выровнен обычно – вправо.

VMLAB имеет только 8 светодиодов – значит нужно отобразить 8 старших бит результата - т.е. биты 9_2. Для этого мы сдвигаем все биты слова ADCW вправо на 2 позиции:
ADCW >> 2. Теперь старшие 8 бит результата АЦП переместились в биты 7_0 младшего байта (LowByte - LB) слова ADCW.

>> n означает сдвинуть все биты числа вправо на n позиций.
Это равносильно делению на 2 в степени n.
<< n означает сдвинуть все биты числа влево на n позиций.
Это равносильно умножению на 2 в степени n.

Светодиоды загораются (показывая "1") при "0" на соответствующем выводе МК – значит, нам нужно выводить в PORTB число, в котором "1" заменены "0" и наоборот. Это делает операция побитного инвертирования. Результатом выражения ~(**ADCW**>>2) будут инвертированные 8 старших бит результата АЦП, находящиеся в младшем байте двухбайтового слова **ADCW.** В Си в переменную можно помещать только тот тип данных, который она может хранить. Так как **PORTB** – это байт, а **ADCW** – это два байта, то прежде чем выполнить оператор присваивания (это знак =) нужно преобразовать **слово** (слово - word - значит два байта) **ADCW** в беззнаковый байт.

Пишем ...(**unsigned char**) ~(ADCW>>2). Результат этой строки – один байт и мы можем поместить его в **PORTB.** Если в регистре **DDRB** все биты равны "1" – т.е. все ножки порта_B выходы, мы безусловно увидим старшие 8 бит результата АЦП горящими светодиодами.

Разберем еще одну строчку:
ADCSRA|=0x40; /* результат поразрядного ИЛИ с маской 01000000 поместить обратно в регистр ADCSRA, т.е. установить бит 6. Обратите внимание на необходимость ставить в конце выражений точку с запятой – не забывайте! */

// **Пункт 6. Функции, используемые в программе**
/* их может быть столько, сколько вам нужно. У нас будет одна, кроме main и обработчика прерывания. Это будет функция, в которой описано начальное конфигурирование МК в соответствии с поставленной задачей. Удобно над функцией сделать заголовок, подробно поясняющий назначение функции!*/

(void)__init_mk(void) {
/* В начале любой функции объявляются локальные переменные – если, конечно, они вам нужны */
/* void - означает пусто. Перед названием функции - void – означает, что функция не возвращает никакого значения. А в скобках после названия означает, что при вызове в функцию не передаются никакие значения. */
// инициализация Port_B
DDRB=0xFF; // все ножки сделать выходами
PORTB=0xFF; // вывести на все ножки "1"

/* настройка АЦП производится записью определенного числа в регистр ADCSRA.
Нам нужно:
- включить модуль АЦП;
- установить допустимую частоту тактирования АЦП при частоте кварца 3.69 МГц. Мы выберем коэффициент деления 64 - это даст частоту такта для процессов в АЦП 57.656 кГц;
- включить прерывание по завершению АЦ преобразования.
По ДШ для этого нужно записать в регистр ADCSRA число 1000 1110 или 0x8E */
// ADC initialization w Oscillator=3.69MHz
// ADC Clock frequency: 57.656 kHz
// ADC Interrupts: On
ADCSRA=0x8E;

/* Теперь выбираем вход АЦП ADC0 (ножка PA0) и внешнее опорное напряжение (это напряжение, код АЦП которого бу-

```
дет 1023) с ножки AREF. Смотрим, что нужно записать для
этого в регистр мультиплексора (выбора входа) АЦП
ADMUX  */

// Нужно записать 0 (он там по умолчанию)
ADMUX=0;
/* Разрешаем глобально все прерывания, разрешенные инди-
видуально. Вы наверно поняли, что индивидуально мы раз-
решили лишь прерывание по завершении АЦП - вот оно то и
сможет возникать у нас.  */
#asm("sei")
} // скобка закрывающая для функции __init_mk()
```

Так делаются вставки ассемблерных инструкций: #asm("инструкция на ассемблере"). Обратите внимание – точки с запятой нет. На Си можно управлять всеми программно изменяемыми битами в регистрах МК, но часто используются такие строки:

```
#asm("sei") // Разрешить ГЛОБАЛЬНО все прерывания
#asm("cli") // Запретить ГЛОБАЛЬНО все прерывания
#asm("nop") // Пауза в 1 такт процессора
#asm("wdr") // Сбросить сторожевой таймер
```

```
/* Пункт 7.   Главная функция  main()  - обязательная!

Главная функция – программа начинает выполняться с нее */
void main(void){
/* В начале любой функции объявляются (если нужны)
ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ  */
__init_mk(); /*Вызываем функцию инициализации, настрой-
ки аппаратуры МК. Выполнив ее, программа вернется сюда
и будет выполнять следующую строку  */
// запускаем первое АЦ преобразование
ADCSRA|=0x40;
// бесконечный цикл в ожидании прерываний
while(1);}
```

```
/* Программа будет крутиться на этой строчке, постоян-
но проверяя, истинно ли условие в скобках после while, а
так как там константа 1 - то условие будет истинно всегда!*/
  // функция main закончена
```

Теперь программа будет работать так. По завершении цикла АЦП будет возникать прерывание и программа будет перескакивать в функцию обработчик прерывания **adc_isr**().

При этом будут автоматически запрещены все прерывания. В конце **adc_isr**() запускается новое АЦ преобразование и при выходе из обработчика прерывания снова разрешаются глобально прерывания, а программа возвращается опять в бесконечный цикл **while**(1). Светодиоды будут высвечивать 8-ми битный код АЦ преобразования напряжения на ножке PA0.

8 ЗАГРУЗКА ПРОГРАММЫ В МИКРОКОНТРОЛЛЕР

Прошивать микроконтроллер можно прямо из программатора, встроенного в компилятор CodeVisionAVR через простейший адаптер (буквально "пять проводков"), соединяющих принтерный порт ПК с прошиваемым микроконтроллером AVR. Результат написания и компиляции программы – файл-прошивку с расширением .hex (и возможно файл с содержимым для EEPROM МК) нужно записать ("зашить") в МК. МК AVR многократно программируются прямо в устройстве, в котором будут работать. Такое программирование называют ISP.

Для этого установите на плате вашего устройства 6 контактов, а лучше 6-ти штырьковый разъем для ISP.

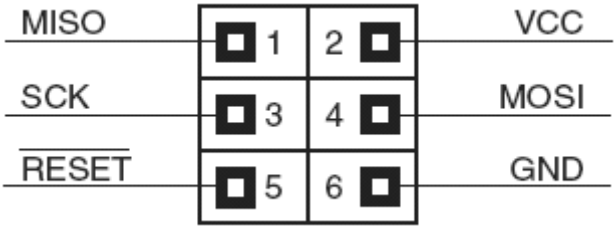


Рис. 8.1 – Разъем для внутрисхемного программирования

Вывод 2 нужно подключить к плюсу питания МК, если вы собираетесь использовать программатор, питающийся от вашего же устройства, например фирменный ISP AVR. Для "5 проводов" этот вывод не подключается. Для программирования достаточно 5 контактов. Соответственно и разъем, который вы будете использовать, может быть любым удобным для размещения на плате и имеющим минимум 5 контактов. Все контакты ISP разъема подсоединяются к ножкам МК в соответствии с названиями.

Будем пользоваться интерфейсом программирования, встроенным в компилятор CodeVisionAVR, и в нем разрабатывать программу для МК.

Вы можете в компиляторе CodeVisionAVR открыть меню "Project -> Configure -> After Make" и отметить чек бокс "Program the chip", затем ОК.

Еще нужно в меню "Settings -> Programmer" выбрать ваш адаптер для программирования.

После безошибочной компиляции программы вам будет доступна кнопка "Program" - нажмите на нее и произойдет программирование МК - т.е. файл .hex будет загружен в память программ МК. Затем МК будет "сброшен" (на ножку RESET будет подан лог. 0, а затем опять "1") и начнет выполнять только что прошитую (загруженную в него) программу.

Вам даже не нужно будет отсоединять адаптер программирования от вашего устройства, если вы не используете в устройстве последовательный интерфейс SPI.

В диалоге настройки программирования не трогайте галочки установки фьюзов МК, если не разобрались четко, что они делают! Иначе вы можете отключить режим ISP или внутренний RC-генератор и для следующего программирования вам понадобится ставить кварц с конденсаторами.

В ATmega с завода включен внутренний RC-генератор на частоте 1 МГц (уточните это по ДШ и его возможные частоты). Если вам нужна другая частота или нужно включить внешний кварцевый или керамический резонатор – вам нужно запрограммировать некоторые фьюзы по таблицам из ДШ. Незапрограммированный фьюз – 1, запрограммированный – 0.

Пример: чтобы включить в ATmega16 внешний кварцевый резонатор с частотой от 3 до 8 МГц с конденсаторами (по схеме рис. 12 ДШ) найдите в ДШ раздел "System Clock". В таблице 2 указаны комбинации фьюзов для разных источников тактового сигнала. Далее написано, что с завода МК поставляется с такой комбинацией фьюзов

CKSEL 0001 SUT 10 CKOPT 1

По таблице 4 находим: для кварца с частотой от 3 до 8 МГц нужны конденсаторы от 12 до 22 пФ и вот такая комбинация фьюзов :

CKSEL 1111 SUT 10 CKOPT 1

Установка фьюзов в программаторе компилятора CVAVR показана на рисунке. Сняв галочку "Program Fuse Bit(s)" вы сможете не менять установку фьюзов!

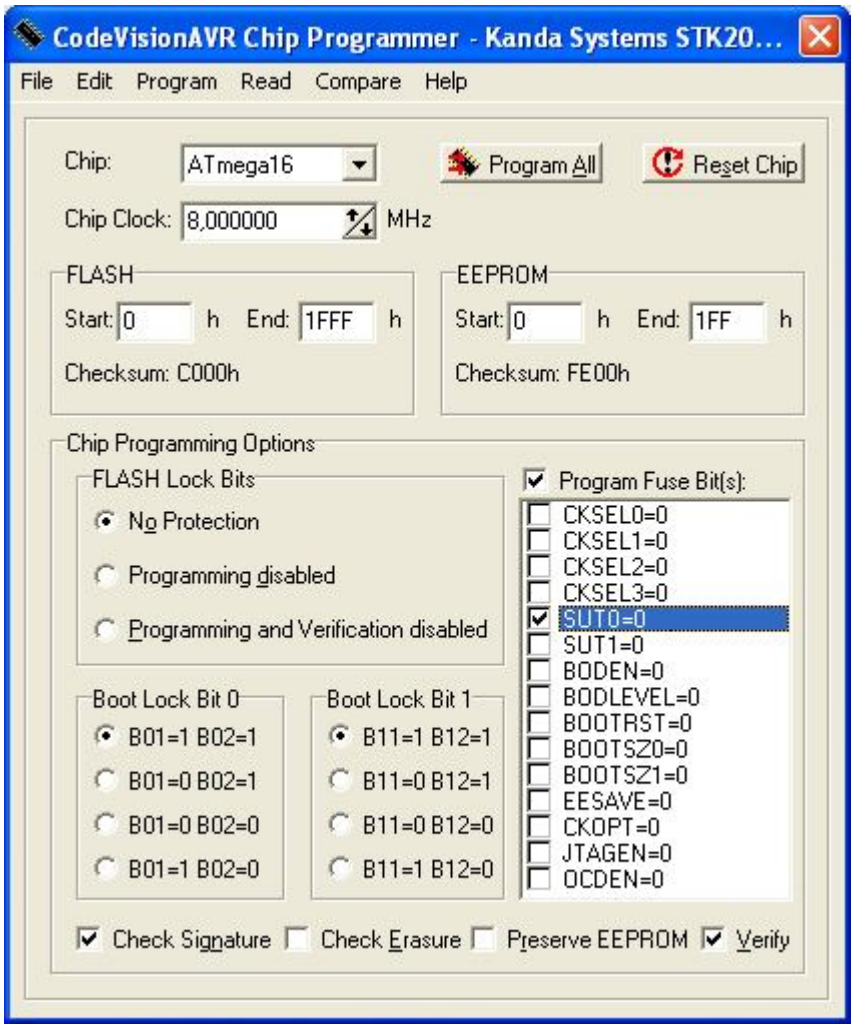


Рис. 8.2 – Окно программатора

Для прошивания МК нажмите кнопку "Program All". Для использования ATmega16 с внешним кварцевым или керамическим резонатором на частотах выше 8 МГц вам нужно установить фьюзы как в примере выше, но запрограммировать CKOPT – значит сделать его "0". Т.е. вам нужна такая комбинация:
CKSEL 1111 SUT 10 CKOPT 0

Приведенный ниже адаптер "5-проводков" (источник информации указан на рисунке) прекрасно работает с компилятором CodeVision. Для изготовления адаптера лучше взять "принтерный" шнур – он длинный и экранированный, а неэкранированные проводки не стоит делать более 10-15 см.

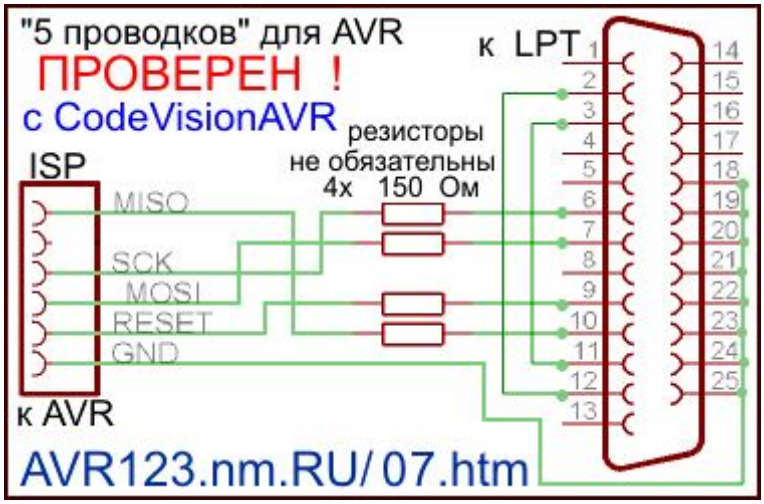


Рис. 8.3 – Адаптер связи AVR с персональным компьютером

Подробно работа с программатором описана в его help. В help CVAVR и VMLAB вы можете найти информацию по работе с компилятором и симулятором, а также ДШ для любого типа микроконтроллера AVR. Естественно, эта информация приводится на английском языке. На русском языке эти вопросы хорошо представлены в работе [9], на базе которой построены разделы [5-9] настоящего учебного пособия и которую можно рекомендовать для более детального изучения дисциплины.

9 МОДЕЛИРОВАНИЕ РАБОТЫ МИКРОКОНТРОЛЛЕРА AVR С ПОМОЩЬЮ СИМУЛЯТОРА VMLAB (лабораторная работа №5)

Цель работы. Целью лабораторной работы является отладка прикладных программ на языке Си для микроконтроллера AVR с помощью компилятора CVAVR и симулятора VMLAB.

Программа работы

1. Установите в директорию **C:\CVAVR** свободную версию компилятора **CodeVisionAVR.** В директории C:\CVAVR создайте папку **z1** (задача 1) для файлов первого проекта.

Запустите компилятор. Для создания файла проекта нажимайте: **Файл -> новый -> проект -> ОК -> No**

- перейдите в созданную для проекта папку **z1** и введите в поле "имя файла": z1
- нажмите "сохранить" - откроется окно конфигурации проекта
- перейдите на закладку "C compiler"
- выберите МК (Chip) ATmega16
- установите частоту тактирования МК (Clock) 4.0 МГц
- нажмите **ОК**.

Перед вами появится открытый текстовый файл **Project Notes - z1.prj**, в котором вы можете записывать свои замечания и мысли по проекту.

Теперь нужно создать главный для нас текстовый файл для набора исходного текста на Си - его расширение .c

- нажимайте:

Файл -> New -> Source -> ОК

появился файл **untitled.c**

- нажимайте:

Файл - Сохранить как

- введите в поле "имя файла": **z1.c** и нажмите **Сохранить**.

Нужно добавить созданный файл z1.c в список файлов проекта - откройте меню конфигурирования проекта: **Project -> Configure**.

В открывшемся диалоге, нужно выбрать ярлык "Files" и нажать кнопку "**Add**". В новом диалоге выберите файл "**z1.c**" и нажмите "**Открыть**". Теперь файл включен в проект.
- нажимайте: **ОК**
- максимизируйте (разверните) окно файла - **z1.c**

Теперь все готово к собственно программированию, т.е. к созданию текста программы на языке Си. Ниже в таблице подготовлен текст программы к задаче 1, реализующей следующее техническое задание: Разработать устройство на микроконтроллере ATmega16, которое будет отображать в двоичном виде горящими светодиодами 8-ми битное число, начиная с 0 и с постоянным увеличением на 1. Устройство питается постоянным стабилизированным напряжением от 4 до 5.5 вольт. Тактирование МК осуществляется от кварцевого резонатора с частотой 4 МГц. Всего подключено 8 светодиодов от ножек порта **A** через токоограничительные резисторы к питанию МК. Переключение светодиодов должно производиться с паузами в 65 мс.

```
#include <mega16.h> /* Вставить вместо этой строки текст
файла mega16.h, содержащий описание регистров МК */
#define PA_OUT DDRA = 0xFF
/* Заменить везде в тексте программы
PA_OUT    на      DDRA = 0xFF */
 // ++++ функция инициализации МК ++++
void initialization(void){
PA_OUT;//сделать весь PORTA выходом
TCCR0 = 0x05;/* таймер включить считать, делая один от-
счет каждые 1024 колебания на ножке XTAL1 */
}
Char per=0;
// ++++  Главная функция  ++++
void main (void){
initialization(); /* Вызвать функцию инициализации МК -
т.е. настройки нужных нам устройств МК в соответствии с
поставленной задачей */
//Бесконечный цикл
while (1){ //Делать всегда
```

```
PORTA=~(per++);
while (!(TIFR&0x01));
// ждем установки флага переполнения timer0
TIFR = 0x01;
// очистить флаг переполнения timer0
      }; //цикл закончен
} //скобка для main()
```

Запишите (без комментариев) программу в окно исходного текста программы. Сохраните изменения: **файл** -> **Save All**.

Для компиляции программы нажмите кнопочку "**Make the project**".

Загляните в папку нашего проекта - z1. В результате компиляции там появилось много новых файлов. Главные для нас:

z1.hex - файл-прошивка для "загрузки" в МК;

z1__.c - копия файла z1.c для симуляторов;

z1.cof - информация, связывающая содержимое файлов z1__.c и z1.hex. Эта информация позволяет при симуляции в VMLAB наблюдать движение программы прямо по коду на языке Си. Указанные файлы будем использовать в симуляторе VMLAB. Необходимым для реального МК является лишь файл прошивки.

Следующие четыре файла содержат нашу программу, написанную на стандартном ассемблере для AVR с привязкой к тексту на Си: z1.asm, z1.lst, z1.vec, z1.inc. Остальные файлы практически не интересны.

2. Запустите VMLAB и откройте созданный проект:

Project -> Open Project File

Перейдите в папку задачи 1 C:\CVAVR\z1\ и наберите имя файла z1_vm.prj проекта для VMLAB. После появления фразы, что такой файл не существует, VMLAB предложит создать его, с чем вы соглашайтесь. В появившемся окне запишите без комментариев приведенный ниже в таблице текстовый файл.

```
; Файл-проект z1_vm.prj  для симуляции по задаче 1.
; Комментарии пишутся  в  VMLAB только в одну строчку
; после точки с запятой

; МК как бы "прошит" файлом - z1.hex. После включения МК
; горящие светодиоды показывают в двоичном виде числа от 0
; до 255 и далее опять с нуля и так по кругу...

; светодиоды подключены к порту_А МК

.MICRO "ATmega16" ; симулируемый МК
.TOOLCHAIN "GENERIC"
.TARGET "z1.hex" ; что "прошито" в МК
.COFF "z1.cof"
.SOURCE "z1__.c"

.POWER VDD=5 VSS=0 ; Питание +5 вольт
; VSS это GND МК - "общий" провод схемы
; Относительно него измеряются напряжения

.CLOCK 4meg ; частота кварца 4 МГц
; Точнее это частота тактирования МК

;  Ввод схемы устройства по задаче 1
; 8 светодиодов подключаются катодами через резисторы
: номиналом 560 Ом к ножкам МК с 33 до 40

; резистор R1 подключить к узлу D1_NODE  и к выводу PA0 МК
; анод светодиода к цепи +5 В. Остальные 7 светодиодов
; подключаются аналогично

D1 VDD D1_NODE
R1 D1_NODE PA0 560

D2 VDD D2_NODE
R2 D2_NODE PA1 560
```

```
D3 VDD D3_NODE
R3 D3_NODE PA2 560

D4 VDD D4_NODE
R4 D4_NODE PA3 560

D5 VDD D5_NODE
R5 D5_NODE PA4 560

D6 VDD D6_NODE
R6 D6_NODE PA5 560

D7 VDD D7_NODE
R7 D7_NODE PA6 560

D8 VDD D8_NODE
R8 D8_NODE PA7 560

; Сигналы на ножках PA0 PA1 PA2
; будем наблюдать в окне виртуального  осциллографа - "Scope"

.PLOT V(PA0) V(PA1) V(PA2)
; Рисовать графики напряжения в перечисленных узлах схемы
```

В меню Project запустите **Re-Build all** ...

Через меню View откройте два компонента: SCOPE – это виртуальный запоминающий осциллограф симулятора и Control Panel – это панель, на которой содержатся нужные нам светодиоды и многое другое, пока нам не нужное.

Через меню Window откройте (обычно оно открывается сразу при открытии проекта) окно Code – в этом окне вы увидите текст симулируемой программы.

Обратите внимание на окно Messages – в нем появляются служебные сообщения симулятора по ходу работы. В окне Messages должно появиться сообщение об успехе и что все готово к запуску (Success! All ready to run). Кроме того, на панели ин-

струментов загорится зеленый светофор – это кнопка, которой можно запускать симуляцию.

Нажатие зеленого светофора эквивалентно подаче "1" на вывод RESET МК при включенном питании, но еще не выполнявшем программу.

В окне Scope появились три графика для сигналов, которые мы будем наблюдать. Установите масштаб по вертикали 2 вольта на деление, а по горизонтали 50 мс.

В окне Code появилось серое поле слева и зеленые квадратики напротив исполняемых строк кода программы на Си – кликнув по такому квадратику мы можем поставить точку останова программы.

Разместите три окна и Control Panel на экране компьютера так, чтобы видеть их все.

Нажмите "светофор" для запуска симуляции программы.

Программа запустится и остановится – в окне Messages появится сообщение. Опять нажимаем на "светофор". Симулятор опять останавливается и сообщает, что произошел сброс от "сторожевого таймера МК" - мы не указали симулятору, что не используем его. Опять нажимаем на "светофор" – теперь программа будет работать непрерывно, пока мы ее не остановим.

Пусть программа симулирует, а вы понаблюдайте за тем, что происходит в указанных выше окнах. Что отображается в окне Control Panel кроме светодиодов?

Понаблюдайте за окнами **SCOPE** и **Code** и за светодиодами. В окне **Code** при симуляции возникают и растут желтые полосы, подсвечивающие строки исполняемой программы. Длины этих подсветок пропорциональны времени, в течение которого программа выполняет код этих строк.

Какой ток потребляется микроконтроллером от источника питания? Остановите симуляцию, нажав красный восьмиугольник «Стоп» и измерьте длительность периода импульсов на ножке PA2 МК. Насколько соответствует она расчетной величине? Для измерения временного промежутка в окне SCOPE симулятора VMLAB нужно установить вертикальные курсоры 1 и 2 на границах измеряемого интервала и в поле Cursor delta time появится значение времени между двумя курсорами.

При измерении коротких повторяющихся интервалов можно мерить время сразу нескольких, а результат поделить затем на число таких интервалов между измерительными курсорами.

Перезапустите МК, кликнув по кнопке с **круговой темно-синей стрелкой**. Вы как бы отключаете и затем снова подаете питание на МК, но создаете "0" на ножке RESET МК – вследствие чего программа не стартует!

Какую функцию выполняет команда **PORTA=~(per++);** ?

Приведите в отчете схему подключения светодиодов к МК.

3. Модифицируйте программу. Переключите светодиоды к порту С. Время паузы между переключениями светодиодов уменьшить в 2 раза.

Для изменения Си кода программы просто запустите компилятор CodeVisionAVR (VMLAB выключать не нужно!) и внесите нужные изменения, затем откомпилируйте проект. Далее перейдите в VMLAB, сделайте **глубокий рестарт** и затем **Re-buid all.** Все! Изменения внесены и все опять готово к симуляции. Таким образом, компилятор и симулятор работают одновременно в одной папке проекта и не мешают, а помогают друг другу. В отчет включите файлы z1.c и z1_vm.prj модифицированного проекта.

4. В следующем проекте будем выводить данные на символьный LCD дисплей (жидко-кристаллический индикатор). Схема его подключения к порту А микроконтроллера приведена на рис. 9.1 (там же указан источник информации, в котором вы можете более подробно ознакомиться с решаемой задачей).

Запустите компилятор **CodeVisionAVR**, затем генератор начального кода **"CodeWizardAVR"** - кликнув **серую шестеренку** слева от красного жучка... Выберите ATmega16 и частоту кварца 4 МГц. Перейдите к закладке LCD и укажите PORTA и 16 символов.

Выполнив **Файл -> Generate, Save and Exit**, создайте в директории C:\CVAVR папку **z2** (задача 2) для файлов нового проекта. Сохраните, нажимая три раза z2, файлы z2.c, z2.prj и

z2.cwp. Посмотрите сгенерированный мастером файл начального кода программы **z2.c**. Какими командами проводится инициализация LCD дисплея? Можно ли удалить из программы команды, реализующие инициализацию периферийных устройств, не используемых в данной задаче?

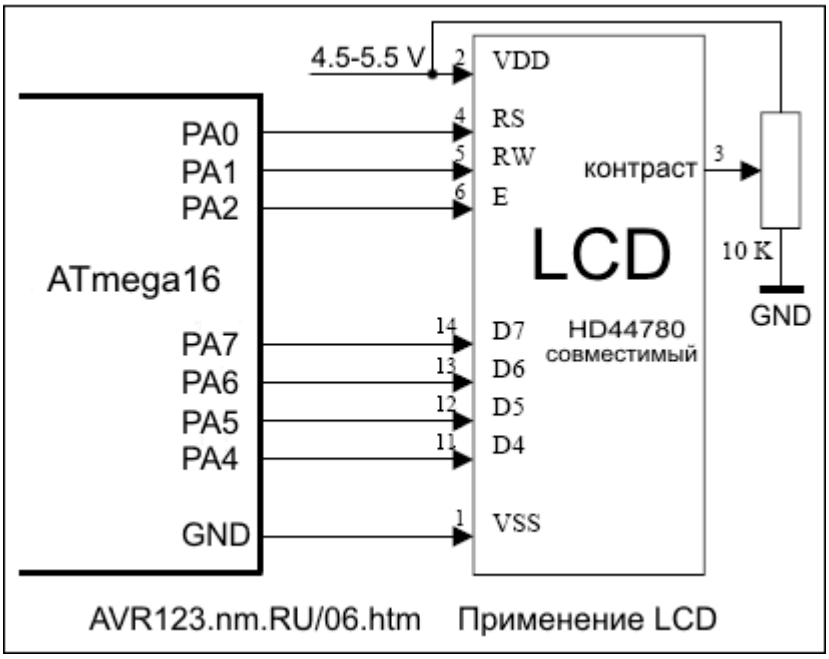


Рис. 9.1 – Типовая схема включения LCD дисплея

После команды
lcd_init(16); // LCD 16 символов на строку
добавьте две строчки:
lcd_gotoxy(5,0); // вывод символов с 6-й позиции в первой строке
lcd_putsf("Hello!"); // счет строк и символов начинается с нуля!
Сохраните (File -> Save All) и откомпилируйте программу.

Не закрывая компилятор, откройте **VMLAB**. В окне **Open Project File** впишите имя файла z2_vm и откройте файл проекта для симулятора **z2_vm.prj**. Впишите в него приведенный ниже в рамке текст и запустите **Re-build all** ... Загоревшийся светофор говорит о том, что программа готова к симуляции. Откройте окно Control Panel и, трижды нажав светофор, добейтесь непрерывной симуляции. Долгожданная надпись на экране LCD появится не

сразу (процесс инициализации LCD продолжается достаточно долго). Почему через некоторое время загорается светофор?

```
; файл z2_vm.prj
.MICRO "ATmega16"
.TOOLCHAIN "GENERIC"
.TARGET "z2.hex"
.COFF "z2.cof"
.CLOCK 4meg
Xdisp LCD(16 2 250K) PA0 PA1 PA2  PA7 PA6 PA5 PA4  nc3 nc2 nc1 nc0
```

Не закрывая **VMLAB** вернитесь в компилятор **CVAVR**. После команды **#include <mega16.h>** добавьте команду

#include <delay.h> // функции организации задержек

После команды **lcd_putsf("Hello!");** добавьте команды:

```
delay_ms(200);
lcd_clear(); // очистка экрана LCD
delay_ms(200);
lcd_gotoxy(5,1);
lcd_putsf("FINISH!");
```

В последнем цикле программы перед комментарием **// Place your code here** добавьте команду **#asm("wdr")** и перекомпилируйте проект.

Вернитесь в VMLAB. Сделайте глубокий рестарт и запустите **Re-build all** ... Как теперь выводится информация на табло дисплея? Почему не загорается светофор после запуска непрерывной симуляции?

5. Проведите исследование работы АЦП. В папке C:\CVAVR \z3 с помощью компилятора создайте файлы проекта задачи 3 на базе программы **z3.c**, текст которой приведен ниже в рамке (он подробно прокомментирован в разделе 7).

```
// файл z3.c
#include <mega16.h>
#include <delay.h>

interrupt [ADC_INT] void adc_isr(void) {
PORTB=(char)~(ADCW>>2);
delay_ms(20);
ADCSRA|=0x40; }

void main(void) {
PORTB=0xFF;
DDRB=0xFF;
ADCSRA=0x8E;
asm("sei")
ADMUX=0;
ADCSRA|=0x40;
while (1); }
```

Затем с помощью симулятора запишите файл Z3_vm.prj.

```
; файл Z3_vm.prj
.MICRO "ATmega16"
.TOOLCHAIN "GENERIC"
.TARGET "z3.hex" ; эмулируемая прошивка МК
.COFF "z3.cof" ; файл содержит привязку
; содержимого [.hex] к коду в [__.c]
.SOURCE "z3__.c" ; исходник на Си, на который сориентирован
файл [.cof].
; это CodeVision добавляет '__' при компиляции
.TRACE ; выводить отладочную информацию в окне
            ; SCOPE - розовым (см. HELP эмулятора)
.CLOCK 4meg ; частота используемого кварца

; Обозначения точек МК, к которым можно
;"подключить" эмулятор: RESET, AREF, PA0-PA7, PB0-PB7,
PC0-PC7, PD0-PD7, ACO, TIM1OVF
```

```
; Для использования АЦП МК нужно подать опорное напряжение
на вывод AREF - мы подадим 5 вольт питания МК. Но в VMLAB
; нельзя соединить два узла напрямую. Берем резистор на 1 Ом.

R1 VDD AREF 1 ; резистор R1 подключен к
; узлам VDD и AREF через сопротивление 1 Ом

; опорное напряжение Vref у нас 5 вольт -
; значит при подаче 5 вольт на вход АЦП
; мы получим результат: 1111111111 (АЦП 10-ти разрядный)

; Вход0 АЦП (это вывод PA0 МК) мы подключим к
; подвижному контакту переменного резистора
; (Slider 1 в окне "Control Panel") -
; чтобы при эмуляции менять напряжение на входе АЦП.

V1 PA0 VSS SLIDER_1(0 5)
; на концах переменного резистора 0 и 5 вольт

; Эмулятор имеет 8 светодиодов -
; подключаем их к выводам порта B

D1 VDD PB0
D2 VDD PB1
D3 VDD PB2
D4 VDD PB3
D5 VDD PB4
D6 VDD PB5
D7 VDD PB6
D8 VDD PB7

; Эмулятор допускает прямое подключение светодиодов к
; плюсу питания и выводам МК - в действительности необходим
; токоограничительный резистор 430-910 Ом
; последовательно с каждым светодиодом!
.PLOT V(PA0) ; на экран осциллографа (окно "SCOPE")
                    ; выведем напряжение на движке потенциометра
```

Запустив проект на симуляцию, понаблюдайте за светодиодами и осциллографом, изменяя положение движка потенциометра. Какое напряжение соответствует единице младшего разряда АЦП? Раскройте окно Peripherals и понаблюдайте за регистрами АЦП при изменении положения движка потенциометра S1. Сравните показания светодиодов и содержимое регистров ADCH и ADCL.

Просмотрите содержимое памяти программ и текст программы на ассемблере. Сколько ячеек занимает программа? По какому адресу расположен вектор прерывания по завершению процесса аналого-цифрового преобразования?

6. Запустите на симуляцию проект, подготовленный в папке **z4** (задача 4). Проект реализован на МК ATmega16.

В окне **SCOPE** (это виртуальный осциллограф) можно увидеть изменения напряжений на ножках МК, указанных в файле проекта - **vmlab.prj**. Верхняя осциллограмма – это сигнал на ножке **TXD (PD1)** последовательного порта **USART**, по которой МК передает данные на COM порт ПК через интерфейс RS232- что передает МК мы видим в виртуальном терминале **TTY** панели **Control Panel**. Там выводится значение **ШИМ** (**PWM**) сигнала, создаваемого на ножке PD5. Сам сигнал виден в окне **SCOPE** – посмотрите, как он меняется в соответствии с сообщаемыми числовыми значениями. На ножке **PD4** формируются импульсы той же частоты с неизменной длительностью.

В файле проекта **vmlab.prj** к ножке **PD5** подключен простейший фильтр нижних частот (ФНЧ) из резистора и конденсатора – он преобразует **ШИМ-сигнал** в постоянное напряжение, которое можно увидеть в окне **SCOPE** (сигнал **DAC**).

Формат передачи данных в примере – 8N1 (это формат по умолчанию для ПК). В таком формате передача байта начинается со "старт-бита" – это лог. "0" на ножке TXD для USART МК и +5...+15 В для COM порта ПК. Затем на ножку TXD выводятся все 8 бит передаваемого байта, начиная с нулевого. За время передачи бита приемник должен определить и запомнить этот уровень. Далее идет "стоп-бит" – это лог. "1" на ножке TXD для USART МК и -5...-15 В для COM порта ПК. Для согласования уровней между МК и ПК включают адаптер MAX232.

7. Протестируйте работу программы, текст которой приведен ниже. Разработайте программу, реализующую световой эффект бегущего огонька без использования ассемблерных вставок.

```
#include <mega16.h>
#include <delay.h>
void main(void){
DDRB=0xFF;
#asm ("ldi r20,1")
while(1){
delay_ms(10);
#asm ("lsl r20")
#asm ("out 0x18,r20")
if (PORTB==0){
PORTB++;
#asm ("ldi r20,1")};
};}
```

Контрольные вопросы

- Назовите нагрузочную способность линий портов AVR.
- Какими ассемблерными вставками можно разрешать и запрещать глобально прерывания в программе для AVR на языке Си?
- Запишите результат выполнения арифметических операций: 245/37 и 245%37.
- Какими командами можно организовать задержку в одну секунду в программе для AVR на языке Си?
- Дать комментарий к команде PORTA=~(per++);
- Объявите переменную mnogo, если она может принимать значения от нуля до миллиона.
- Прокомментировать результат выполнения команды ADCSRA|=0x40;

Содержание отчета

Отчет должен содержать тексты отлаживаемых программ с конкретной датой их компиляции, комментарии по ходу выполнения пунктов программы работы и рисунки, вставляемые в текст формата WORD, отображающие окна **SCOPE** , **Control Panel** и т.д. с результатами моделирования, а также ответы на контрольные вопросы.

10 МОДЕЛИРОВАНИЕ РАБОТЫ МИКРОКОНТРОЛЛЕРА AVR С ПОМОЩЬЮ СИМУЛЯТОРА PROTEUS VSM

Proteus VSM – программа-симулятор микропроцессорных устройств. Поддерживает МК: PIC, 8051, AVR, HC11 и другие распространенные процессоры. PROTEUS содержит огромную библиотеку электронных компонентов: более 6000 популярных аналоговых и цифровых моделей устройств. PROTEUS VSM позволяет очень достоверно моделировать и отлаживать достаточно сложные устройства, в которых может содержаться несколько МК одновременно и даже разных семейств в одном устройстве. PROTEUS VSM великолепно работает с компилятором CodeVisionAVR.

Proteus VSM является средой сквозного проектирования. Это означает создание устройства, начиная с его принципиальной схемы и заканчивая изготовлением печатной платы. Достаточный набор инструментов и функций, среди которых вольтметр, амперметр, осциллограф, всевозможные генераторы, способность отлаживать программное обеспечение микроконтроллеров, делают Proteus VSM хорошим помощником разработчика электронных устройств.

Proteus VSM состоит из двух самостоятельных программ: ISIS и ARES. ARES – это трассировщик печатных плат. Основной программой является ISIS, в ней предусмотрена горячая связь с ARES для разводки платы.

При запуске программы появляется основное окно (рис.10.1). Самое большое место отведено под окно редактирования. Именно в нем происходят все основные процессы создания, редактирования и отладки схемы устройства. В самом низу основного окна расположена панель управления активной симуляцией (ПУСК-ПОШАГОВЫЙ РЕЖИМ-ПАУЗА-СТОП).

На рис. 10.1 приведена схема моделирования цифрового термометра на микроконтроллере ATmega16 с термодатчиками фирмы DALLAS SEMICONDUCTOR и выводом информации на LCD-дисплей. Микросхема DS18S20 обеспечивает 9-битные температурные измерения по шкале Цельсия. Микросхема DS18S20 подключается через 1-проводную шину, которая по определению требует только одной линии данных (а также общей) для взаимо-

действия с центральным процессором. Она имеет рабочий температурный диапазон от -55°C до +125°C и точность ±0.5°C в диапазоне от -10°C до +85°C.

Модель термометра DS18S20 позволяет задавать температуру термодатчика (в данном примере +100.0 и -32.0 °C). При указании свойств микроконтроллера подключается файл прикладной программы с расширением .hex, подготовленный компилятором CVAVR.

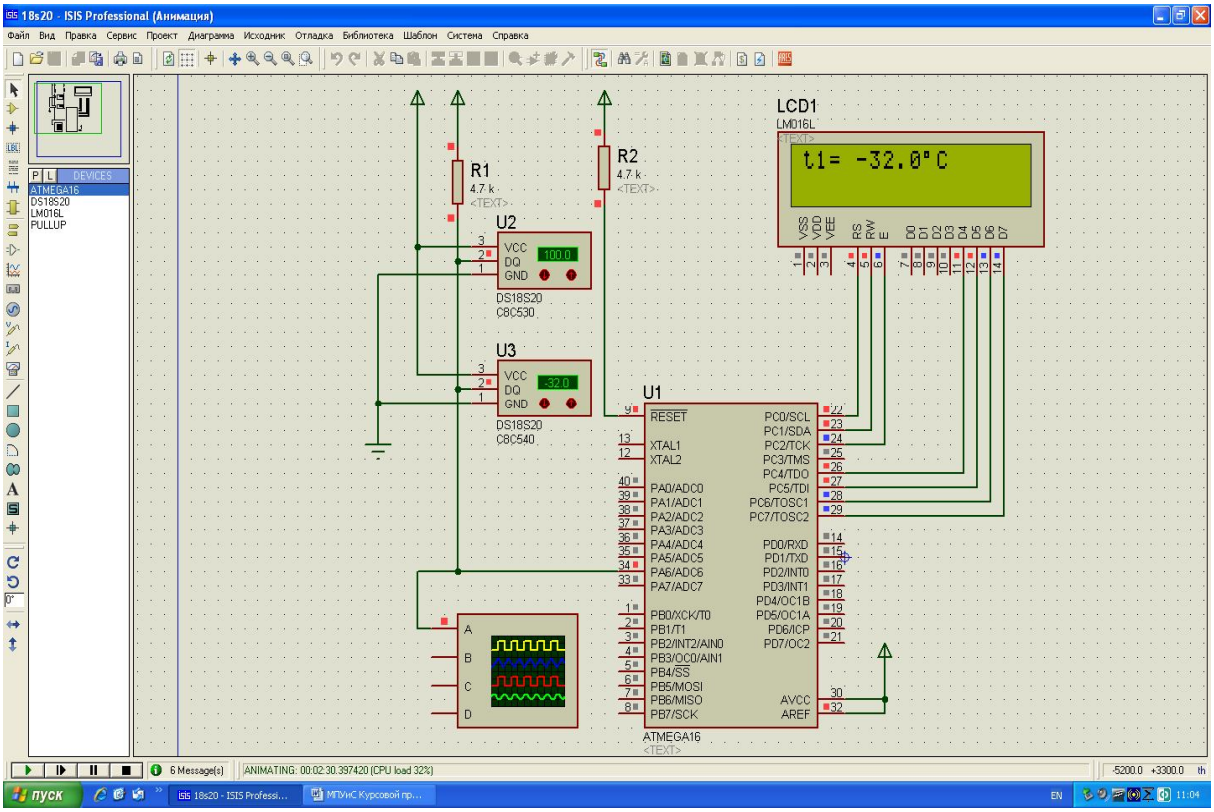


Рисунок 10.1 – Моделирование цифрового термометра для измерения комнатной и наружной температуры

С помощью встроенного в среду моделирования Proteus осциллографа можно снять управляющие сигналы микроконтроллера и информационный сигнал датчика. Временные диаграммы сигналов (протокол 1-Wire) представлены на рисунках 10.2 и 10.3 соответственно. Уровни напряжения можно видеть на всех выводах компонентов непосредственно (лог.1 - красный цвет, лог.0 - синий, неподключенные выводы - серый цвет). В данном примере моделирование позволило показать работоспособность устройства при измерении очень высоких и низких температур.

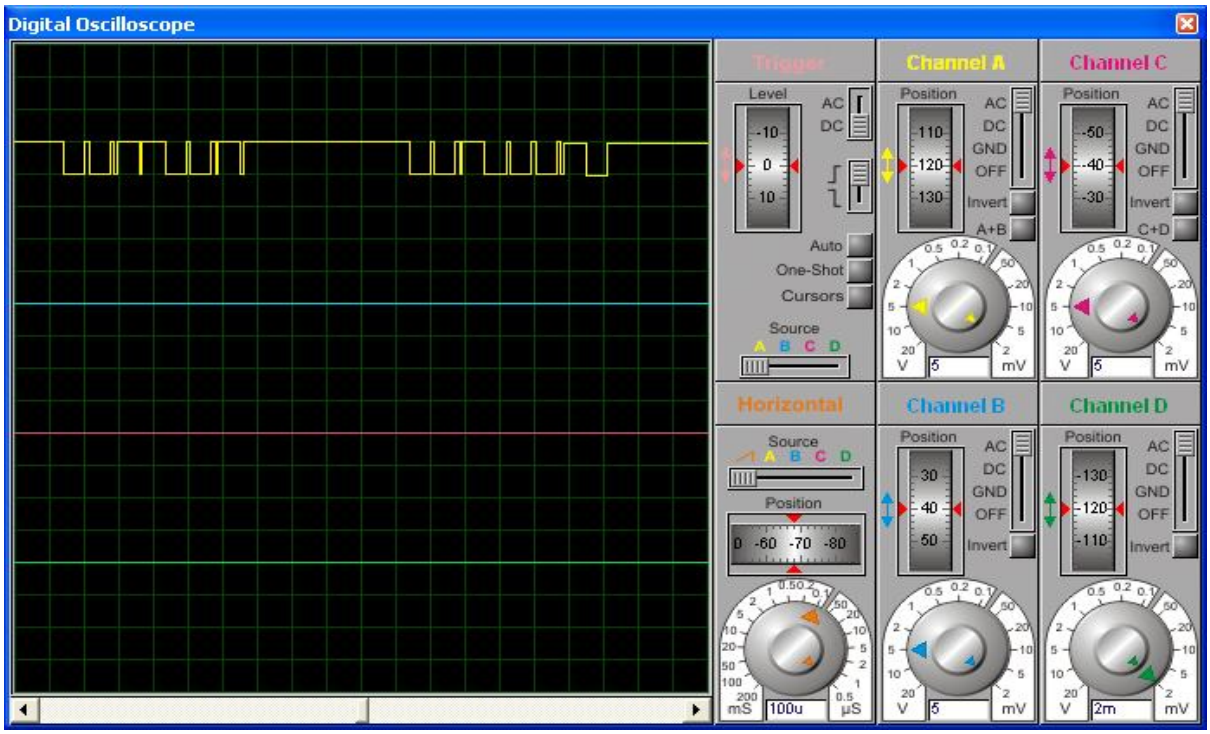


Рисунок 10.2 – Осциллограмма управляющих сигналов микроконтроллера

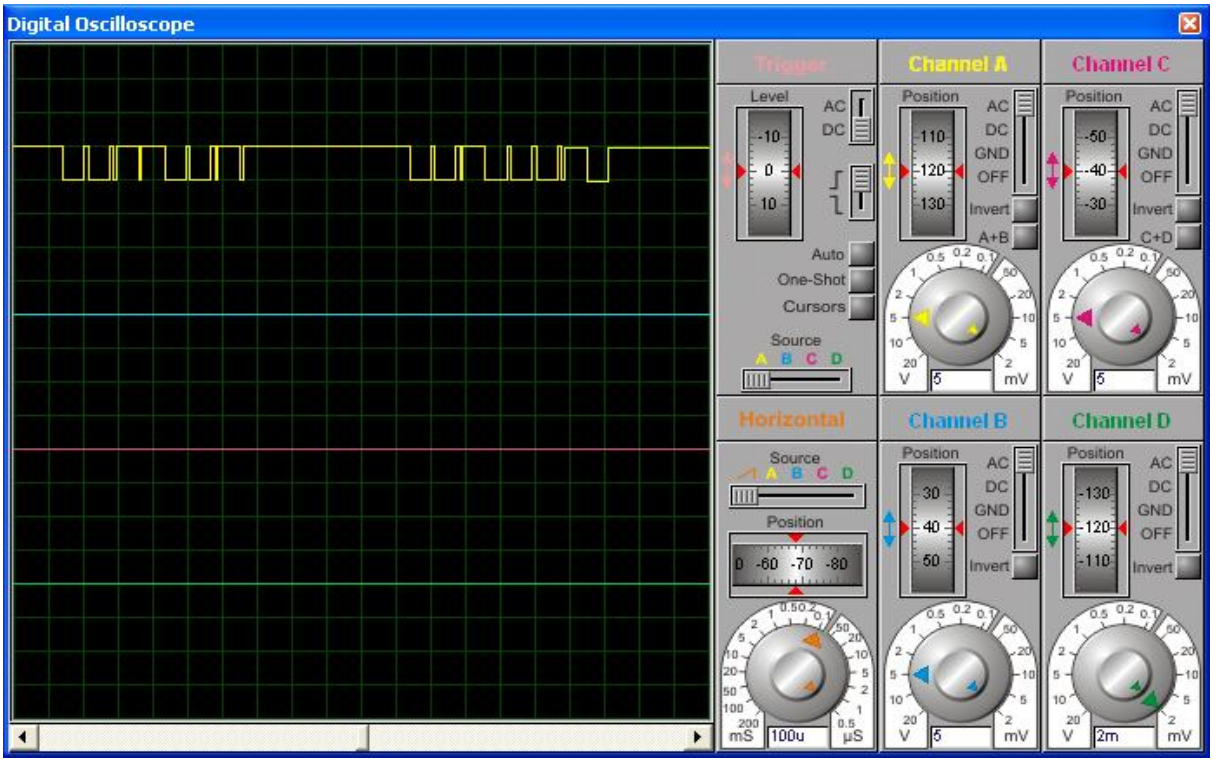


Рисунок 10.3 – Осциллограмма информационного сигнала датчика

11 ИЗМЕРИТЕЛЬ ЧАСТОТЫ СЕТИ

Техническое задание. Спроектировать цифровой измеритель частоты сети. На цифровые индикаторы выводятся три цифры (например, 51.7), т.е. значение частоты с точностью до десятых долей герца. Диапазон измеряемых частот от 20 до 99 Гц. Время измерения – не более 1 с.

Обоснование алгоритма решения задачи

Для получения требуемой точности можно подсчитать количество импульсов сетевого напряжения за 10 с. Такой алгоритм не проходит по быстродействию.

Другой способ основан на измерении периода сетевого напряжения и оценке частоты по формуле $f = 1/T$. Если период измерять в секундах, то получаем значение частоты в герцах. Точность определения частоты зависит от точности измерения периода и точности выполнения операции деления. При использовании микроконтроллеров следует ориентироваться на работу с целыми числами. Типичные погрешности, учитываемые при этом – погрешность дискретизации и погрешность округления.

Будем измерять период сети в микросекундах. Для типового значения частоты 50 Гц период T составляет 20000 мкс. С помощью 16-разрядного таймера, работающего на частоте 1 МГц, можно измерить период до 65 мс, что соответствует частоте порядка 17 Гц. При частоте 99 Гц период равен 10101 мкс, т.е. таймер позволит измерить его с высокой точностью (погрешность дискретизации порядка 0,01%).

Основное рабочее соотношение $f = \frac{10^6}{T} = \frac{1000000}{T}$.

Для того чтобы работать с целыми числами и гарантировать точность цифры десятых долей герца после выполнения целочисленного деления и округления будем рассчитывать значение частоты по формуле

$$f = \left(10^8 \Big/ T + 5 \right) \Big/ 10$$

и после вычисления f в виде трехразрядного десятичного числа добавлять точку перед последней цифрой при выводе информации на табло. Например, при T=20500 мкс получаем $\frac{10^8}{T} = 4878$ и $\frac{10^8}{T} + 5 = 4883$. На цифровых индикаторах нужно зажечь значение частоты 48.8 Гц.

Выбираем микроконтроллер ATmega16 с кварцем на 8 МГц (время выполнения простейших команд 1/8 мкс). Функциональная схема устройства с распределением функций портов показана на рис.9. Порт PA используется для вывода информации на индикаторную панель. Порт PB используется для программирования микроконтроллера (например, с помощью интерфейса «пять проводков» он подключается к COM-порту персонального компьютера).

Прямоугольные импульсы с частотой сети, формируемые на выходе триггера Шмитта, будем подавать на вход INT0 (вторая линия порта PD). Запустив в основной программе таймер/счетчик T1 в требуемый режим работы, поручим основные функции по решению задачи подпрограмме внешнего аппаратного прерывания по нарастающему фронту INT0:

- Останов T1
- Чтение периода (TCNT1L+TCNT1H*256)
- Сброс T1
- Новый запуск T1
- Вычисление значения частоты
- Вывод данных на индикаторную панель

Полагаем, что на реализацию указанных действий будет затрачено время не более 10 мс и к началу нового периода микроконтроллер будет готов повторить описанный алгоритм в новой подпрограмме прерываний.

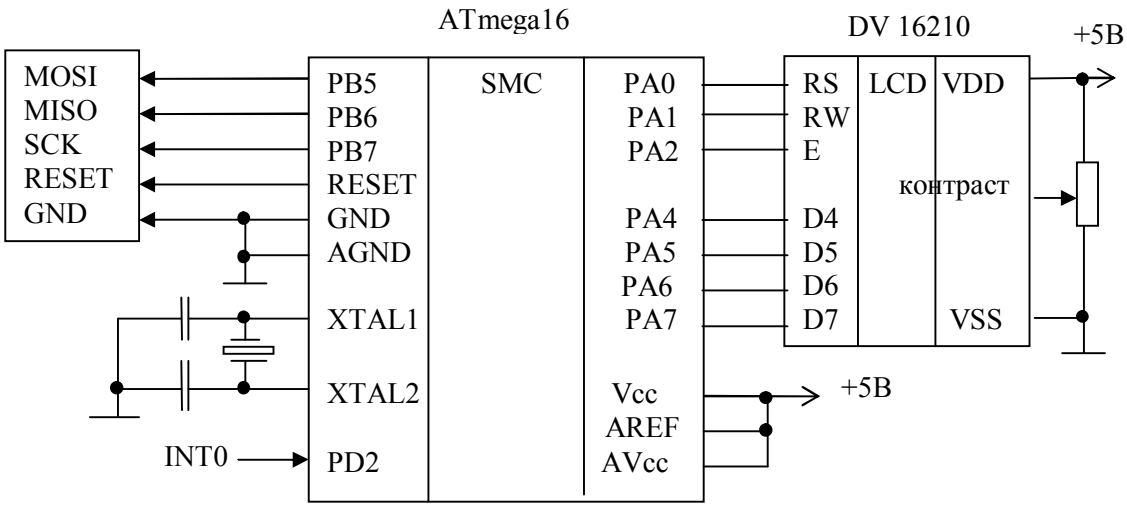


Рисунок 11.1 – Функциональная схема измерителя частоты сети

Разработка прикладной программы

Последовательность шагов по разработке и отладке программы может быть следующей (проект формируем в папке z10):

1. Создаем новый проект выбирая:
File→New→Select Project

2. Выбираем использование CodeWizardAVR:
Use the CodeWizard?→Yes

3. В окнах CodeWizardAVR фиксируем тип МК и рабочую частоту:
Chip→Chip: ATmega16→Clock: 8MHz

4. Конфигурируем LCD-дисплей: LCD→PORTA→16

5. Конфигурируем Timer1: Timers→Timer1→
Clock Value: 1000kHz→Interrupt off: Timer1 Overflow→Val: 0xFFFF

6. Настраиваем прерывания: External IRQ→INT0 Enabled→Rising Edge

7. Генерируем файлы C source, C project и CodeWizardAVR project выбирая:
File|Generate, Save and Exit→
Create new directory: C:\cvavr\z10→

Save: z10.c→Save: z10.prj →Save: z10.cwp

8. Редактируем код C source (основная творческая часть по реализации алгоритма)

9. Смотрим или модифицируем конфигурацию проекта выбирая Project→Configure→After Make→Program the Chip

10. Компилируем программу выбирая:
Project→Make

11. Автоматически программируем ATmega16 на STK500:
Apply power→Information→Program.

Ниже приведен текст программы на языке Си. Для обеспечения возможности выводить на LCD-дисплей русские буквы в программу добавлена таблица кодов и функция void putchar(char c). Большую часть времени МК находится в цикле основной программы.

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.7 beta 5 Standard
Automatic Program Generator
© Copyright 1998-2007 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project : Измеритель частоты сети
Version : от 17 до 99 Гц
Date    : 10.11.2008
Author  : Шарапов А.В.
Company : ПрЭ
Comments: курсовой проект

Chip type           : ATmega16
Program type        : Application
Clock frequency     : 1,000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

#include <mega16.h>

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x1B ;PORTA
#endasm
#include <lcd.h>
```

```
#include <stdio.h>

flash char Decode2Rus[255-192+1]= {

0x41,0xA0,0x42,0xA1,0xE0,0x45,0xA3,0xA4,
0xA5,0xA6,0x4B,0xA7,0x4D,0x48,0x4F,0xA8,
0x50,0x43,0x54,0xA9,0xAA,0x58,0xE1,0xAB,
0xAC,0xE2,0xAD,0xAE,0xAD,0xAF,0xB0,0xB1,
0x61,0xB2,0xB3,0xB4,0xE3,0x65,0xB6,0xB7,
0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0x6F,0xBE,
0x70,0x63,0xBF,0x79,0xE4,0x78,0xE5,0xC0,
0xC1,0xE6,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7

};

#define _ALTERNATE_PUTCHAR_
void putchar(char c)
{
if(c>=192) lcd_putchar(Decode2Rus[c-192]); else
lcd_putchar(c);
}

// Declare your global variables here
unsigned int period=20000;
unsigned int f=500;
char lcd_buffer[33];// Буфер в ОЗУ для LCD-дисплея

// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Place your code here

TCCR1B=0; //останов таймера
period=TCNT1L+256*TCNT1H;// Вычисление периода сети
TCNT1H=0;//обнуление таймера
TCNT1L=0;
TCCR1B=0x02;//запуск измерения нового периода

f=(100000000/period+5)/10;// Вычисление частоты сети
sprintf(lcd_buffer," power circuit frequency
%u.%uHz",f/10,f%10);// Подготовка строки для LCD-дисплея
lcd_gotoxy(1,0);// Вывод строки со второй позиции в 1 строке
lcd_puts(lcd_buffer);
}
```

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

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 1000,000 kHz
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x02;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Falling Edge
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x02;
MCUCSR=0x00;
GIFR=0x40;

// LCD module initialization
lcd_init(16);

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

while (1)
      {
      // Place your code here
      };}
```

Моделирование работы устройства с помощью VMLAB

Ниже приведен файл проекта для симулятора. Для моделирования работы измерителя частоты сети к входу INT0 подключается генератор прямоугольных импульсов, период которого можно задавать программно. К порту PA подключен LCD-дисплей (две строки по 16 символов).

```
; файл z10_vm.prj
.MICRO "ATmega16"
.TOOLCHAIN "GENERIC"
.TARGET "z.hex"
.COFF "z.cof"
.SOURCE "z__.c"
.CLOCK 8meg
.POWER VDD = 5  VSS = 0

;V[inst_name]  node VSS PULSE(v_initial v_final t_delay t_rise
;t_fall t_width ;t_period)
V1 PD2 VSS PULSE(0 5 0 0 0 5m 21m)

;X[inst_name]  LCD(chars lines oscil_freq) RS RW E D7 D6 D5 D4 D3 D2 D1D0
Xdisp LCD(16 2 250K) PA0 PA1 PA2 PA7 PA6 PA5 PA4 nc3 nc2 nc1 nc0
```

Цель моделирования – проверка работоспособности выбранного алгоритма и оценка времени, которое микроконтроллер затрачивает на обработку информации. При проведении эксперимента раскрываем рабочие окна: Peripherals (окно периферийных устройств, чтобы пронаблюдать значение таймера T1), I/O Ports (окно портов ввода/вывода, чтобы увидеть подачу импульсов на линию PD2), Control Panel (экран жидкокристаллического индикатора).

При моделировании подключался файл Z10.hex, сформированный при трансляции программы, подготовленной на языке Си, с выводом информации на русском языке (`частота сети 47.6 Гц`).

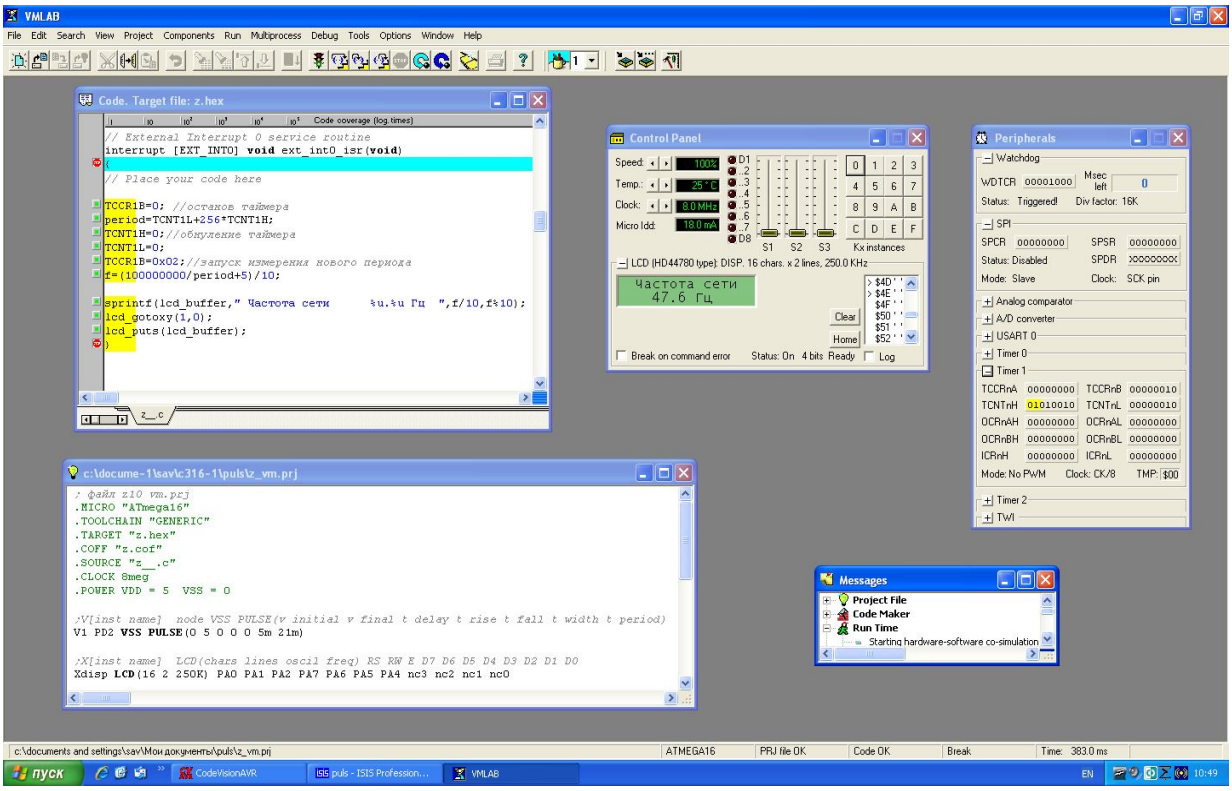


Рисунок 11.2 – Рабочие окна симулятора VMLAB

В окне Code (текст программы на языке Си) ставим точки останова перед открывающей и закрывающей скобкой подпрограммы прерывания и, проведя цикл измерения частоты (для этого дважды нажимаем зеленую кнопку светофора), определяем по данным окна Messages время выполнения подпрограммы прерывания. Оно составило 3,81 мс, что говорит о работоспособности выбранного алгоритма определения частоты. Практически каждый период сетевого напряжения обновляется значение рассчитываемой частоты.

В окне Program Memory можно увидеть машинные коды выполняемых команд и соответствующие им команды на языке ассемблера микроконтроллера. Можно реализовать пошаговое выполнение команд, нажимая кнопочку правее светофора.

Моделирование работы устройства с помощью симулятора PROTEUS VSM

При моделировании на вход прерывания INT0 подавался сигнал с генератора прямоугольных импульсов, частоту колебаний которого можно задавать с помощью соответствующих регулировок (сигнал униполярный амплитудой 5 В). Убеждаемся, что сразу после изменения частоты в диапазоне 17-120 Гц аналогичные изменения происходят на табло LCD-дисплея. При моделировании не требуется подключение кварцевого резонатора и источников питания микроконтроллера и LCD-дисплея.

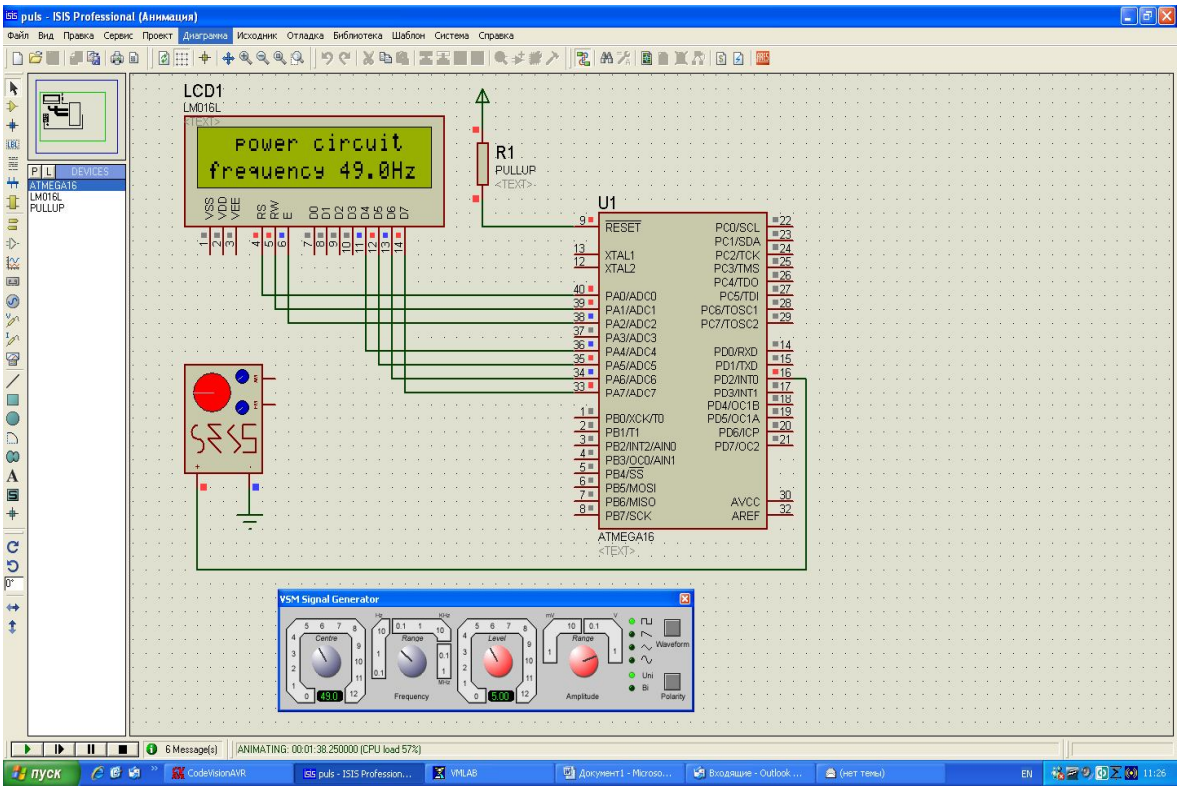


Рисунок 11.3 – Рабочие окна симулятора Proteus VSM

При моделировании подключался файл Z10.hex, сформированный при трансляции программы, подготовленной на языке Си, с выводом информации на английском языке `(power circuit frequency 49.0 Hz)`.

ЛИТЕРАТУРА

1. Шарапов А.В. Цифровые и микропроцессорные устройства: Учебное пособие. - Томск: ТМЦ ДО, 2003. - 166 с.

2. Кривченко И.В. Микроконтроллеры общего назначения для встраиваемых приложений производства Atmel Corp. // Электронные компоненты. - 2002. - №5. - С. 69–73.

3. Кривченко И.В. Преобразование двоичных чисел в двоично-десятичные // ООО «ЭФО». - 1999.

4. Гребнев В.В. Микроконтроллеры семейства AT89 фирмы Atmel // ООО «ЭФО». - 2000.

5. Гребнев В.В. Микроконтроллеры семейства AVR фирмы Atmel. - М.: ИП Радиософт, 2002.

6. Кривченко И.В. Система команд и программная модель AVR // ООО «ЭФО». - 1999.

7. www.atmel.com.

8. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL. - М.: Издательский дом «Додэка-XXI», 2005. -560 с.

9. http://avr123.nm.ru. Краткий курс. Микроконтроллеры AVR. Начинающим "с нуля".