МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

“Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В.И.Ульянова (Ленина)” (СПбГЭТУ“ЛЭТИ”)

Учебное пособие по дисциплине
«Компьютерная и микропроцессорная техника»

Часть 1

Встраиваемые микроконтроллеры AVR-8

Бондаренко Д.Н.

Санкт-Петербург 2015

УДК 621.316.544.1 (035.5)
ББК 32.844.1_04я2

Автор: Бондаренко Д. Н.
Компьютерная и микропроцессорная техника. Часть 1. Встраиваемые микроконтроллеры AVR8: Электронное учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2015. – 229 с.

ISBN 978_5_94120_220_1

Микроконтроллеры для встраиваемых приложений являются базовым элементом современных распределенных систем управления, в том числе в электротехнологических комплексах. Сравнительная простота и дешевизна 8-разрядных микроконтроллеров делает их удобным иструментом как для начального освоения, так и для применения во многих практических задачах.

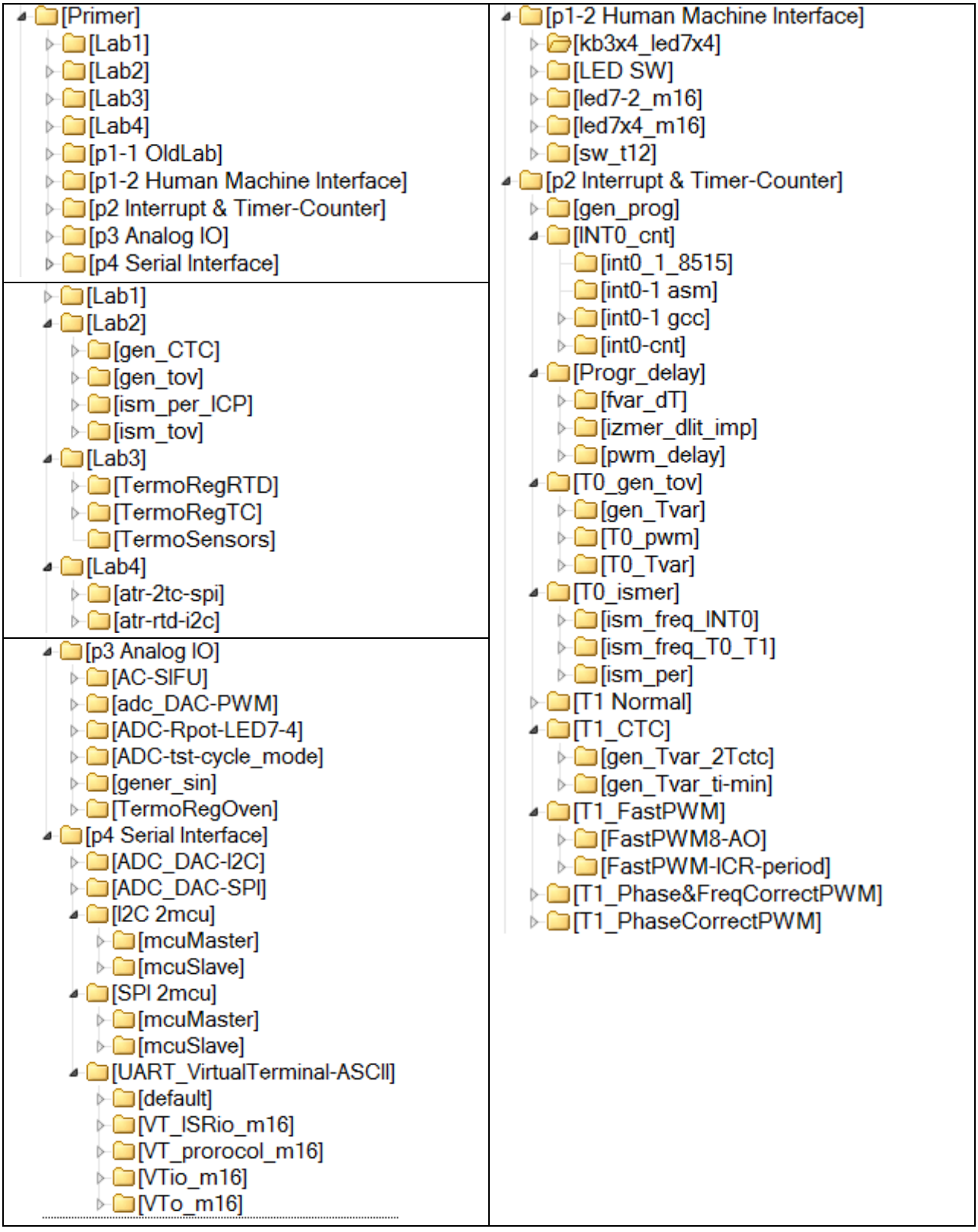
В пособии рассматриваются способы реализации алгоритмов управления с использованием ресурсов микроконтроллера и схемотехники внешних подключений. Знакомство с устройством ядра и встроенной периферии микроконроллеров семейства AVR (Atmel) закрепляется многочисленными примерами программ на языке С и схем подключения внешних устройств.

Особенностью данного курса является ориентация примеров на программные симуляторы: цифровые модели микроконтроллеров в отладчике IDE AVR-Studio и аналого-цифровые схемотехнические модели в системе моделирования Proteus VSM среды проектирования электронных устройств Labcenter Electronics. Примеры в тексте учебного пособия имеют ссылки на папки и файлы с готовыми проектами для AVR-Studio и Proteus VSM, они размещены на диске.

Пособие адресовано студентам, изучающим первую часть дисциплины «Компьютерная и микропроцессорная техника» плана подготовки бакалавров по направлению 13.03.02 «Электроэнергетика и электротехника» и профилю 13.03.02-11 «Электротехнологические установки и системы». Может быть полезным всем осваивающим азы программирования и схемотехники восьмиразрядных микроконтроллеров для встраиваемых приложений на примере семейства AVR.

Содержание

Дерево размещения папок с примерами на диске



Введение

Микропроцессорное устройство (МПУ) служит для выполнения арифметических, логических операций и операций управления, записанных в машинном коде. *Процессор* – исполнитель машинных инструкций, часть аппаратного обеспечения компьютера или программируемого логического контроллера; отвечает за выполнение операций, заданных программами. Приставка *микро-* означает – реализованный в виде одной микросхемы.

История микропроцессоров началась в 70-е годы 20 века с модели для построения калькулятора. Сейчас понятие микропроцессор стало очень широким, этим термином называют устройства разные как по своим техническим характеристикам, так и по области применения. Наиболее широкое применения микропроцессоры нашли в задачах управления, что отразилось в появлении еще одного термина – *микроконтроллер* (МК) – от английского *controller* – устройство управления. Первым широко распространенным микроконтроллером стала модель *MCS*-8048 фирмы *Intel*, следующая модель *MCS*-8051 стала «стандартом де-факто» и клонируется до сих пор.

Нас в первую очередь будут интересовать МПУ как устройства *промышленной автоматики*.

Цель автоматизации – повышение производительности труда, улучшение качества продукции, оптимизация управления, устранение человека от производств, опасных для здоровья. Автоматизация требует дополнительного применения датчиков (сенсоров), устройств ввода, управляющих устройств (контроллеров), исполнительных устройств, устройств вывода, использующих электронную технику и методы вычислений, иногда копирующие нервные и мыслительные функции человека.

Основными объектами в промышленной автоматике выступают *объект управления* (ОУ) и *система управления* (СУ). В качестве объекта управления нас интересует *технологическая установка* или *комплекс*. Система управления обеспечивает требуемый уровень автоматизации, то есть степень участия человека в процессе управления.

Процесс разработки любого устройства управления обязательно включает разработку *алгоритма*. Алгоритм (от имени учёного аль-Хорезми) – точный набор инструкций, описывающих *порядок действий для решения задачи за конечное время*.

Особенностью устройств управления на базе МПУ является *взаимодействие аппаратных узлов и программного обеспечения*, что усложняет разбиение алгоритма на элементы. Программная реализация алгоритма управления обеспечивается за счет циклического выполнения заданного набора команд (программы). Вариации поведения МПУ при изменении внешних условий обеспечиваются поведением данных (вычисление арифметических и логических выражений) и структурой программы (ветвления и частные циклы при проверке заданных условий).

Программная реализация обеспечивает высокую степень *гибкости*, так как одно и тоже МПУ путем замены управляющей программы может решать совершенно разные задачи, коррекция данных (настроек) обеспечивает адаптацию к изменению условий работы.

Недостатком программной реализации является повышенное время реакции по сравнению с чисто аппаратным решением. Это ограничивает использование принципа программного управления в задачах, требующих высокого быстродействия, хотя это

компенсируется непрерывным ростом частоты тактирования и производительности МПУ.

Фактор времени является очень важным в задачах управления – выполнение той или иной части алгоритма должно соответствовать достаточно жестким требованиям «не раньше и не позже».

Фактор времени и прогресс производства микропроцессоров привел к интеграции в в кристалл периферийных блоков, аппаратно реализующих элементы типовых задач управления, прежде всего задач ввода/вывода.

В соответствии с задачами микропроцессорное устройство управления кроме вычислительно-решающего устройства должно иметь:

- входы для получения сигналов с датчиков (дискретных и аналоговых),
- выходы для подачи управляющих сигналов (дискретных и аналоговых),
- средства поддержки человеко-машинного интерфейса от простейшего, состоящего из светодиодных индикаторов, кнопок, тумблеров, потенциометров, до цветных графических дисплеев с сенсорным вводом,
- узлы для аппаратной поддержки последовательных интерфейсов, обеспечивающих интеграцию подобных устройств управления в распределенную систему управления с регулярными и иерархическими связями.

Подобные МПУ, реализованные в одной микросхеме (в одном кристалле) называются *микроконтроллерами для встраиваемых приложений [microcontroller for embedded applications]*, они хорошо приспособлены для построения узлов или законченных систем управления, имеют небольшую цену, габариты и вес.

Наибольшее распространение среди них получили 8-разрядные МК. Это объясняется тем, что большинство задач управления не требует значительной арифметической производительности, большую часть задач составляют задачи логической обработки, задачи ввода/вывода и четкая координация выполнения задач во времени.

Использование микроконтроллеров для встраиваемых приложений предполагает обязательную разработку электрической схемы и конструкции печатной платы, что удлиняет время разработки устройства управления и ограничивает область применения встраиваемых МК в основном тиражируемыми устройствами.

Для автоматизации единичных и мелкосерийных установок более подходит обширный класс готовых устройств управления на базе МПУ:

- простейшие устройства «релейной» автоматики (таймеры, терморегуляторы),
- интеллектуальные датчики и исполнительные органы,
- программируемые логические контроллеры,
- операторские панели,
- панельные/промышленные компьютеры,
- разнообразные каналы связи (интерфейсы), позволяющие объединять перечисленные устройства между собой для построения распределенных систем управления.

Несмотря на значительные отличия в технических характеристиках, все перечисленные устройства имеют подобную организацию, схожие средства и способы программирования.

Базовой частью данного курса является достаточно близкое знакомство с семейством 8-разрядных микроконтроллеров *AVR* фирмы *Atmel*. Выбор данного семейства

обусловлен хорошим соотношением цена/качество, доступностью на отечественном рынке, богатым инструментарием по программированию и отладке и личным опытом автора данного пособия.

Основной упор в изучении МК сделан на освоение принципов программного управления, механизмов прерывания, использованию таймеров/счетчиков в задачах формирования и измерения временных интервалов, знакомству с задачами аналогового ввода/вывода и задачами организации обмена данными по последовательному интерфейсу.

Лабораторный практикум ориентирован на использование программ-симуляторов, имитирующих поведение МК под управлением программы. Каждая тема сопровождается многочисленными готовыми примерами программ с подробным описанием алгоритма и методики его тестирования в том или ином симуляторе. Для закрепления осваиваемого материала по каждой теме предлагаются наборы заданий разного уровня сложности. Простые требуют лишь незначительного изменения параметров или структуры алгоритма управления, более сложные требуют осмысленного объединения элементов алгоритмов из нескольких примеров.

Свободно распространяемая фирмой *Atmel* среда *AVR-Studio* поддерживает процесс разработки программ для МК на языке ассемблер и процесс отладки программ, написанных как на языке ассемблер, так и на языке *C* (*ANSI* C). Встроенный симулятор микроконтроллера реализует только цифровые функции ядра МК и большинства встроенных узлов ввода/вывода.

Свободно распространяемый компилятор языка C/C++ *AVR_GCC* поставляется с оболочкой *WinAVR,* При установке поверх среды *AVR-Studio,* последнюю можно использовать как общую надстройку, то есть работать только в среде *AVR-Studio* на языке *C/C++,* пользуясь всеми возможностями символьной отладки.

Данный набор программ является достаточным для освоения минимальных навыков программирования и отладки МК.

Для знакомства со схемотехникой подключения к МК цифровых и аналоговых задатчиков и измерителей сигналов, устройств индикации, кнопок, переключателей, потенциометров и пр. рекомендуется использовать пакет аналого-цифрового моделирования *Proteus VSM* из среды сквозного проектирования электронных устройств *Proteus* (программа *ISIS)* фирмы *Labcenter Electronics*. Наличие моделей МК управляемых программой пользователя и моделей стандартного набора компонентов электронной техники позволяет глубже понять алгоритмы работы примеров и расширяет возможности отладки не только программы, но и схемотехники примеров.

Контрольные вопросы по итогам лекционного материала и лабораторного практикума помогают проверить запоминание терминов, определений и понимание пройденного материала.

Часть 1. Архитектура микроконтроллеров

1.1. **МК для встраиваемых приложений и семейство МК *AVR*-8**

Микропроцессоры (МП) и микроконтроллеры (МП) для встраиваемых приложений *[embedded applications]* ориентированы на построение законченных систем управления и их узлов. Их отличают от других процессорных устройств такие существенные черты:

- развитые встроенные аппаратные узлы ввода/вывода, разгружающие процессор от рутинных действий и поддерживающие интерфейсы различных типов - дискретный, аналоговый, параллельный, последовательный;

- интегрированная на кристалл память программ и данных, в том числе, энергонезависимая;

- наличие большого числа разновидностей кристаллов, различающихся числом выводов и типом корпуса, объемами памяти, составом блоков ввода/вывода и пр.;

- встроенные узлы загрузки кодов программ и данных и узлы внутрисхемной отладки, работающие по последовательному интерфейсу;

- гибкая система тактирования и управления энергопотреблением, что актуально для мобильных устройств.

Основные различия между МК данной группы сводятся к производительности процессора, определяемой сочетанием следующих факторов: разрядность процессора (8/16/32), частота тактирования (десятки - сотни мегагерц), состав системы команд и режимов адресации. Рост разрядности сочетается с ростом частоты и снижением напряжения питания (с 5 В до 3.3 В, а ядра - до 1.2 В).

Несколько особняком стоят цифровые сигнальные процессоры, ориентированные на вычисления в реальном масштабе времени как с целыми числами, так и числами с плавающей точкой. Их характерные черты: наличие специальных команд (умножения с накоплением, организации кольцевых буферов и циклов и пр.), мультишинная организация внутренних связей, усовершенствованная двухпортовая память, встроенные быстродействующие АЦП и ЦАП либо высокоскоростной последовательный интерфейс для их подключения и др.

В данном курсе мы будем подробно знакомиться с 8-разрядными МК, доля производства которых в общем объеме процессорных микросхем составляет чуть меньше 30%. Их столь широкое распространение обусловлено низкой стоимостью, богатым набором встроенных блоков ввода/вывода различного типа и тем, что большая часть задач управления не требует серьезных вычислений, преобладают задачи логические, управления во времени и преобразования интерфейсов.

Первым в ряду 8-разрядных МК, получивших массовое распространение в конце 70-х годов 20 века была модель *MCS-8048* и ее развитие *MCS-8051/52* фирмы *Intel*, выполненные по *n*-МОП технологии. Они стали стандартом "де факто" для копирования и дальнейшего развития десятками фирм во всем мире, в том числе в СССР (КМ1816ВЕ48, КМ1816ВЕ51).

В последние два десятка лет многие фирмы на новом технологическом уровне разработали и выпускают более совершенные семейства МК, теперь уже преимущественно по К-МОП технологии.

На отечественном рынке 8-разрядные МК представлены прежде всего «клонами» и «потомками» модели *MCS*-8051, семействами *PIC12*/14/16/18 (*MicroChip*), семейством *AVR* (фирма *Atmel*) и пр. Эти МК ориентированы на решение задач управления с преобладанием

логических и несложных арифметических законов управления. Их максимальные тактовые частоты доходят сегодня до 16…25 МГц при питании 3…5 В. Уровень входных и выходных сигналов при этом имеет достаточный запас помехоустойчивости – примерно одна треть от напряжения питания, то есть не менее 1.7 В.

В частности, в данном курсе наибольшее внимание будет уделено семейству *AVR* фирмы *Atmel*, как одному из динамично развивающихся и доступному на отечественном рынке.

Семейством (или серией) МК называется ряд микросхем однородной разработки, имеющих идентичную архитектуру, но различающихся количеством ячеек памяти, выводов (ножек), конструкцией корпуса, составом и количественными характеристиками встроенных периферийных модулей.

Выпуск МК для встраиваемых применений в виде семейства позволяет производителям снизить затраты на создание богатой номенклатуры микросхем (при разработке новых кристаллов используются готовые проверенные в производстве модули кристалла), а пользователям МК сокращает затраты времени на освоение моделей, наиболее подходящих для решения конкретных задач.

История и особенности семейства AVR-8

В середине 90-х годов (20 века) фирма *Atmel* выпустила клон МК *MCS-8051* по технологии К-МОП – семейство *AT89xx*, в котором реализовала два существенных на тот момент новшества:

- интегрированная на кристалле память программ типа *FlashROM* и энергонезависимая память данных типа *EEPROM,*
- встроенный механизм загрузки/выгрузки содержимого памяти программ (*FlashROM*) и данных (*EEPROM*) с использованием последовательного интерфейса (типа *SPI*).

Сочетание этих новаций обеспечило существенное удешевление и упрощение - отпала необходимость в использовании дорого керамического корпуса с кварцевым окном для ультрафиолетового стирания, и отпала необходимость использования не дешевого (не менее 300$) параллельного программатора, в который при каждой операции записи приходилось помещать корпус с микросхемой памяти программ. Новая технология стала называться внутрисистемным программированием *[In-System Programming - ISP]*, существенно упростила и удешевила процедуру редактирования содержимого памяти программ (ПП) и данных (ПД) - теперь стало возможным выполнять перепрограммирование МК прямо "в системе", то есть на целевой плате.

Следующим шагом фирмы *Atmel* в конце 90-х годов стала разработка нового семейства 8-разрядных МК *AVR*-8 (*AT90xxxx*), в котором кроме новшеств, внедренных при разработке *AT89xx*, добавились следующие:

- интегрированные на кристалл узлы ввода аналоговых сигналов - аналоговый компаратор и аналогово-цифровой преобразователь;
- новая система команд *[RISC]* сразу разрабатывалась с расчетом на использование языка *C/C++*,
- процессор поддерживает 4-шаговый конвейер, что обеспечивает выполнение большинства команд (кроме команд перехода) за один машинный цикл,
- отказ от аккумулятора и использование достаточно большого файла регистров общего назначения (32*8 бит) снижает необходимость частых обращений к ПД (ОЗУ) с более

долгими или длинными режимами адресации; в этом же файле разместились 16-разрядные указатели на ПД и ПП;

- отказ от использования внешней ПП при наличии функции внутрисистемного программирования позволил использовать корпуса с малым числом ног (менее 40), что еще более снижает стоимость микросхемы. Дополнительным преимуществом отказа от внешней ПП стала повышенная степень защиты содержимого ПП от несанкционированного копирования.

Последовательность разработки подсемейств *AVR*-8: *AT90, ATmega, [FPSLIC], ATtiny, ATxmega.*

1.2. Структура и архитектура МК

Архитектурой называют образ, видимый пользователю, т. е. программисту или схемотехнику, работающему с данным МК. Основными элементами архитектуры являются:

• система команд и способы адресации;

• организация памяти и регистров;

• организация встроенных периферийных устройств и системы прерываний;

• типовые схемы включения с описанием назначения и расположения выводов и особенностей внутренней схемотехники выводов.

Знание архитектуры является минимально достаточным для понимания и практической работы с микроконтроллером. Сравнение однотипных черт и параметров архитектуры разных МК позволяет грамотно выбирать МК для решения конкретной задачи.

Структура МК отражает состав и связи его внутренних блоков в графическом виде. Знание структуры МК не является обязательным для работы с МК, но облегчает понимание и сравнение МК между собой за счет дополнительного визуального восприятия.

Разработчики МК снабжают каждую модель МК подробным техническим описанием *[DataSheets]*, в котором приводятся отличия в архитектуре, структурные схемы, технические параметры в виде текстов, таблиц и графиков, иногда с примерами программирования. Описания элементов архитектуры общих для всей серии (семейства) МК, прежде всего, описания системы команд и режимов адресации, размещаются в отдельных документах на сайте разработчика. Примеры применения МК для решения конкретных прикладных задач находятся там же в файлах описания приложений *[Application Notes]*.

Архитектура AVR

Система команд 8-разрядных МК серии AVR содержит от 90 до 133 инструкций, число которых зависит от модели. Как и для большинства других МК, этот набор можно условно разделить на три основные группы:

- арифметико-логические, включая сдвиги (собственно вычисления),
- передачи данных между ячейками памяти МК,
- ветвлений (переходов) в программе (формируют структуру программы).

По формату обрабатываемых данных различают команды, оперирующие с байтами

(8-битное слово) и с битами, отдельные команды работают с двухбайтными числами. Нередко группу команд работы с битами выделяют в самостоятельную четвертую группу.

Формат команды описывает структуру машинного слова команды. Часть бит выделяют под *код операции*, это обязательная часть любой команды. Остальные биты указывают *операнды*, то есть данные или адрес ячейки данных или адрес перехода. Различают команды безоперандные, одно- и двухоперандные. В двухоперандных командах обработки данных один из операндов называется источником *[source]*, его содержимое не изменяется, другой – приемником *[destination]*, его содержимое содержит результат операции.

Способы адресации определяют способы кодирования операндов, для кодирования используется двоичный код. Различают следующие виды адресации:

- *прямая* – адрес в виде константы, число бит *n* определяет число адресуемых ячеек $N = 2^n$,
- *непосредственная* (или абсолютная) – данное в виде константы,
- *косвенная* – адрес регистра-указателя (*X, Y, Z, PC, SP*), в котором находится адрес ячейки данных или перехода; разновидности косвенной адресации:
  - простая косвенная,
  - с преддекрементом – значение регистра указателя уменьшается на 1 до использования адреса,
  - с постинкрементом – значение регистра указателя увеличивается на 1 после использования адреса,
  - индексная – значение адреса вычисляется как сумма содержимого регистра-указателя и константы,
  - относительная – разновидность индексной с использованием счетчика команд (*PC*) как регистра указателя перехода.

Краткий перечень команд находится в Приложении 3 и разбит на подгруппы команд. В первой колонке находится *мнемоника* – имя команды из 3-4 латинских символов, далее идут *операнды* – адреса ячеек данных (или сами данные) или адреса перехода. Далее следуют описание действий команды словами и в виде условных значков. В колонке «Флаги» указаны имена битов регистра состояния *SREG*, значения которых меняются при выполнении данной команды. В колонке «цикл» указана длительность выполнения команды в машинных циклах. В колонке «слов» - длина команды в словах (1 слово – 16 бит). «Двоичный код команды» расписан по битам, для обозначения кодов адресов используются условные значки, расшифровка – ниже таблицы.

Более подробное рассмотрение системы команд будет дано в следующих разделах данной главы.

Память МК с так называемой *гарвардской* архитектурой разделяется на две принципиально различные области – память программ (ПП) *[Program Memory]* и память данных (ПД) *[Data Memory]*. Эти области образуют два непересекающихся адресных пространства, имеют разное назначение, размер и тип ячеек. Разделение адресных пространств обеспечивается системой команд и способами адресации (и системой шин).

Память программ служит для хранения кодов программы и констант. Физическим носителем ПП выступает энергонезависимое постоянное запоминающее устройство (ПЗУ) типа *FlashROM*.

Память программ располагается только в кристалле, представлена ячейками

размером 2 байта, количество которых варьируется от 1 до 128 (256) килобайт (КБ), что требует от 9 до 16 (17) разрядов адреса соответственно. Это и определяет разрядность регистра счетчика команд (СК) *[PC: Program Counter]*. При размещении в ПП констант предоставляется доступ к отдельному байту, что усложняет обращение по адресам больше $2^{16} – 1 = 65\,535$.

Память данных имеет байтовую организацию и адресное пространство объемом до 64 КБ (16 битный адрес). В этом пространстве находятся три области, различные по назначению и техническим характеристикам:

1) энергозависимая оперативная память статического типа *SRAM* для временного хранения данных (от 0 байт до 64 КБ);

2) пространство регистров ввода/вывода (РВВ) *[I/O Registers]*, иногда называют регистрами специальных функций (РСФ) *[Special Function Registers – SFR]*, служит для управления и наблюдения состояния процессора и встроенных периферийных устройств ввода/вывода; занимает 64 ячейки в младших моделях, в старших моделях требуется больше ячеек *[Ext. I/O Reg.]*, они занимают часть адресного пространства *SRAM*);

3) энергонезависимая память типа *EEPROM* для длительного хранения данных (до 4 КБ с доступом через регистры ввода/вывода).

Область оперативной памяти может иметь до трех подразделов:

1) обязательный файл регистров общего назначения (РОН) *[GPR]* объемом 32 байта – наиболее интенсивно используемая память для размещения данных и указателей (адресов),

2) резидентная (то есть размещенная на кристалле) память данных *[Internal SRAM]* объемом от 0 до 8 КБ,

2) внешняя память данных *[External SRAM]* объемом до 60 КБ, требует наличия 19 выводов для подключения.

Состав периферийных устройств ввода/вывода варьируется от модели к модели МК. В МК подсемейств *AT90, ATmega, ATtiny* в набор таких устройств входят:

1) Параллельные порты ввода / вывода – основной механизм ввода и вывода дискретных (цифровых) сигналов разрядностью 8 бит.

2) Параллельная магистраль (шина) для подключения внешних микросхем памяти данных (при наличии достаточного количества выводов – не менее 19).

3) Векторная маскируемая система прерываний с аппаратным распределением приоритетов и входы прерываний для быстрой реакции, асинхронной к основному циклу управляющей программы.

4) Таймеры/счетчики для формирования временных интервалов, генерации импульсов и измерения временных параметров внешних сигналов.

5) Контроллеры последовательного интерфейса для коммуникации в распределенной системе управления как в рамках одной установки, так и в составе распределенной системы управления.

6) Устройства аналогового интерфейса: аналоговый компаратор, аналогово-цифровой преобразователь, реже – цифро-аналоговый преобразователь.

В последней новации семейства *AVR*-8 – 8/16-разрядном подсемействе *ATxmega*, используются идентичные ядро и архитектура, усовершенствованные таймеры /счетчики и

устройства аналогового интерфейса и принципиально новые периферийные блоки:
1) Система управления событиями для разгрузки процессора от пересылки сигналов между блоками МК.
2) Контроллер прямого доступа к памяти - ориентирован на пересылку пакетов данных между блоками МК без участия процессора.

Типовая схема включения МК определяется числом и назначением выводов *[pin]* микроконтроллера. Подсемейство Classic имеет от 8 до 44 выводов, ATtiny – от 8 до 28 выводов, ATmega – от 40 до 100 выводов.

Следует выделить выводы особого назначения – питания *[VCC, GND, AGND* и пр.*]*, сброса *[RESET]*, тактирования *[XTAL1, 2]*, остальные используются как входы/выходы параллельных портов *[PB0, PD4, ...]*, либо выполняют одну или несколько альтернативных функций, это обозначается «косой чертой»: *PD2/INT0/RXD1/PCINT26*. Альтернативность означает возможность выбора только одной из функций для конкретного применения путем программной настройки.

Каждая модель МК выпускается в одном или нескольких типах корпусов, имеющих одинаковое или примерно одинаковое число выводов и разную конструкцию. Различают корпуса для сквозного монтажа (DIP-8, DIP-20, DIP-40) и для поверхностного монтажа (SOIC-8, SOIC-20, TQFP-44, TQFP-64, TQFP-100). В техническом описании модели МК приводится описание расположения (нумерация) выводов для каждого корпуса (цоколевка).

Конкретная схема включения показывает внешние связи МК с системой питания, элементами тактирования (если не используется внутреннее тактирование), устройством загрузки/выгрузки кода программы и схемными узлами аппаратной обработки входных и выходных сигналов. Примеры будут рассмотрены ниже, после более подробного знакомства с архитектурой МК *AVR*-8.

Структура AVR

Основные элементы структуры процессорных устройств: процессор, память программ и данных, устройства ввода/вывода и набор шин (проводников), объединяющих все элементы в систему.

Типовая структура микроконтроллера AVR-8 представлена на рис. 1.1.

Здесь показаны основные внутренние блоки, связи между ними и внешние выводы. Блоки, изображенные пунктирной линией, являются вариативной частью МК, остальные входят в состав любой модели.

Практически неизменным по составу является ядро МК, включающее процессор *[AVR CPU]*, память программ *[FlashROM]*, память данных *[Static RAM]* и *[EEPROM]*, набор бит конфигурации *[Fuse]* и защиты *[Lock]*, супервизор питания и сброса, систему тактирования, сторожевой таймер *[WatchDog Timer]*, модуль внутрисистемной загрузки/выгрузки *[In-System Programming]*. Здесь варьируется только число ячеек памяти, а также состав и назначение битов конфигурации.

В составе любой модели МК можно встретить такие блоки ввода/вывода, как модуль прерываний, параллельные порты ввода/вывода, не менее двух таймеров/счетчиков (*T/C0* и *T/C1*) и аналоговый компаратор *[Analog Comparator]*.

В моделях МК с расширенным составом можно встретить увеличенный состав

параллельных портов, таймеров/счетчиков и такие блоки ввода/вывода, как аналого-цифровой преобразователь (АЦП) *[Analog to Digital Converter]* с источником опорного напряжения (ИОН) *[Analog Reference]* и модули аппаратной поддержки последовательного интерфейса различных стандартов – от простейших байтовых асинхронного *[U(S)ART]* и синхронного типа *[SPI, I2C]*, до сложных и эффективных *CAN* и *USB*.

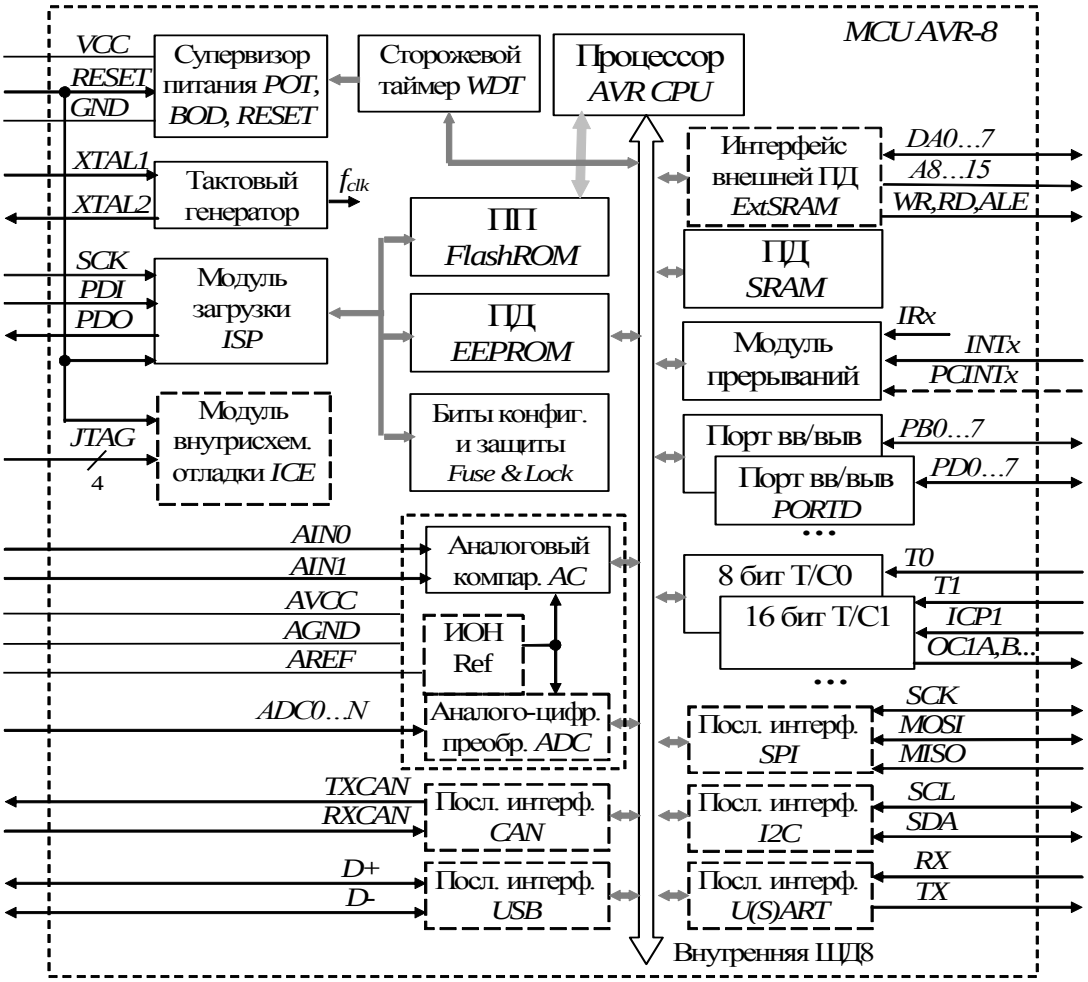


Рис. 1.1. Структура МК AVR-8

В моделях последних поколений можно встретить модуль поддержки процесса отладки по принципу внутрисхемной эмуляции *[In Circuit Emulator]*, реализуемой в МК серии *ATtiny* через вход сброса *RESET* по технологии однопроводного интерфейса, а в МК серии *ATmega* – через дополнительный последовательный интерфейс *JTAG*.

Память данных и блоки ввода вывода подключены к процессору по внутренней системной магистрали с 8-разрядной шиной данных, память программ – по отдельной магистрали с 16-разрядной шиной данных. Остальные внутренние связи между блоками показаны в усеченном виде: выход тактового генератора f_{clk}, интерфейс модуля загрузки/выгрузки с блоками энергонезависимой памяти и битами конфигурации и защиты.

Для подключения внешней памяти данных *[External Static RAM]* используется внешний интерфейс шинного типа, аналогичный внутренней магистрали, но с сокращенным набором проводников: мультиплексированная шина адреса и данных

DA0…7, шина старших разрядов адреса *A8…15,* сигналы управления ЗАПИСЬ *[WR]*, ЧТЕНИЕ *[RD]* и временного разделения адреса и данных *[ALE]*.

Внешние выводы *[pins]* можно условно разбить на группы.

Выводы *VCC* и *GND* служат для подачи питания, к выводу *VCC* подводится положительное напряжение питания (+5 В или ниже), вывод *GND* подключается к нулю питания. Эти выводы имеют самую большую токовую нагрузку (до нескольких сотен миллиампер), поэтому в корпусах с большим количеством выводов их дублируют для равномерного распределения нагрузки.

Вход сброса *RESET* и выводы системы тактирования *XTAL1, XTAL2* могут быть альтернативно использованы как входы/выходы параллельных портов (выбор только конфигурационными битами *Fuse*).

В МК серии *ATmega* для улучшения условий питания аналоговых модулей выделяют выводы питания *AVCC* и *AGND*, их потенциалы могут незначительно отличаться от потенциалов выводов цифрового питания (*VCC* и *GND*). Вывод *AREF* может использоваться как вход или выход опорного напряжения (если выбран при программной настройке аналогового компаратора или АЦП).

Все остальные выводы МК по умолчанию выполняют функции входов параллельных портов (*Px.y, x* = {*A, B, …*}, *y* = {*0, 1, ... 7*}) и программно могут быть перенастроены либо как выходы портов, либо для использования предусмотренной разработчиком альтернативной функции. Под этим понимается функции входов или выходов остальных блоков ввода/вывода: входов прерываний (*INTx, PCINTx*), таймеров/счетчиков (входы *Tx, ICPx*, выходы *OCyx*), аналогового компаратора (*AIN0, AIN1*) или АЦП (*ADC0, ADC1, …*), последовательного интерфейса, шинного интерфейса внешней памяти данных.

Подробнее о назначении выводов можно узнать из описания работы модулей ввода/вывода (в этом пособии) и технического описания конкретного микроконтроллера.

1.3. Тактирование, процессор и арифметико-логическая группа команд

Тактирование, энергопотребление и сброс

Работа всех цифровых узлов МК начиная с процессора и кончая блоками ввода/вывода, требует тактирования для синхронизации выполняемых действий.

Частота тактирования *fclk* для каждой конкретной модели имеет предельное верхнее значение *fclk_max*, значительное превышение которого ведет к перегреву кристалла и увеличивает вероятность сбоев. Работа на частоте равной или близкой к *fclk_max* обеспечивает максимальную производительность процессора и узлов ввода/вывода.

Значение *fclk_max* в общем случае зависит от диапазона напряжения питания МК. Например, в диапазоне 4.5...5.5 В для *AT90S8515* предельная частота 8 МГц, а для *ATmega164A* – 20 МГц. При снижении напряжения питания до 2.7 В предельная частота в два раза ниже – 4 и 10 МГц соответственно.

Выбор частоты влияет на энергопотребление МК – с ростом частоты растет ток потребления. Это определяется КМОП технологией МК: статический ток потребления почти отсутствует, присутствует емкостной ток переключения транзисторных ячеек: $I = U / Xc = Vcc * 2\pi fclk * C$, где C – эквивалентная емкость МК, зависящая от режима работы и

состава работающих периферийных блоков. Это актуально для мобильных приложений с питанием от химических источников (аккумулятор или "батарейка"). Данная формула не учитывает составляющую тока выводов МК, номинальные значения которых могут достигать 20 мА на один вывод.

В задачах управления некоторые узлы и функции предъявляют повышенные требования к стабильности и точности поддержания заданного значения частоты тактирования *fclk*. В первую очередь это относится к модулям таймера/счетчика и контроллера асинхронной последовательной связи типа *U(S)ART*.

Современные МК поддерживают несколько способов тактирования, смена способа выполняется с использованием устройства загрузки ("программатора") через биты конфигурации (*Fuse*-биты).

Как правило, при поставке эти биты обеспечивают выбор тактирования от встроенного *RC*-генератора. Недостатки такого способа заключаются в невысокой стабильности (влияние напряжения питания и температуры) и точности (разброс параметров *R* и *C*) частоты. Варианты МК с калиброванными значениями *R* и *C* обеспечивают разброс не хуже 1%.

Альтернативой встроенного тактирования являются способы с внешним тактированием, что требует использования одного или двух выводов МК. В МК подсемейства *ATtiny* с малым числом выводов (8, 20) приходится искать компромисс, так как не используемые для тактирования выводы можно использовать как входы/выходы общего назначения.

Самый распространенный вариант внешнего тактирования – использование кварцевого резонатора, требует двух выводов *XTAL1, XTAL2* (см. рис. 1.3).

Варианты с использованием одного входа тактирования *XTAL1* сводятся к использованию внешней *RC*-цепи или внешнего тактового генератора. Стоимость такого генератора сравнима со стоимостью «младших» моделей МК.

Энергопотребление современных МК (8-разрядных) достаточно мало – мощность *P* = 5 В * (10...300) мА = 50...1500 мВт при максимальной частоте тактирования (16…20 МГц). Для работы в мобильных приложениях, а также при реализации функции часов реального времени актуальны режимы пониженного энергопотребления. Они основаны на программном отключении ряда или большинства блоков МК (режим "сна") и последующей активации по внутреннему или внешнему сигналу (прерывания). Это позволяет снизить потребляемый ток на 1...3 порядка и иметь весьма малое время пробуждения.

Не следует забывать и о снижении частоты тактирования для снижения энергопотребления. Современные модели МК кроме выбора способа тактирования битами конфигурации позволяют программно менять коэффициент деления частоты или переключаться на альтернативные источники тактирования – *RC*- генератор сторожевого таймера (около 1 МГц) или генератор часов реального времени (обычно 32768 Гц).

Сброс МК – это процедура при старте или рестарте (повторном старте), состоящая из трех автоматически выполняемых этапов:

1) этап пуска и установления работы тактового генератора; время определяется состоянием специальных *Fuse*-битов от единиц микросекунд (для внешнего тактового

генератора) до десятков миллисекунд (для тактирования с использованием *RC* или кварцевого резонатора);

2) этап предустановки регистров процессора и РВВ (РСФ) в заданное начальное состояние (несколько машинных тактов); это обеспечивает строго определенное состояние устройств ввода/вывода;

3) счетчик команд [Program Counter – PC] устанавливается на нулевой адрес памяти программ и процессор переходит к выполнению программы.

Напомним, что содержимое энергозависимой памяти данных (ОЗУ и РОН) при сбросе не инициализируется – это забота программиста.

Процедура сброса в *AVR* МК выполняется автоматически в следующих случаях:

1) штатное включение питания – по нарастанию напряжения питания выше заданного уровня *[Power-On Reset];*

2) переход уровня напряжения на входе *RESET* из НИЗКОГО в ВЫСОКИЙ; НИЗКИЙ уровень используется для подключения программатора/отладчика, к данному входу можно подключить кнопку сброса;

3) переполнение сторожевого таймера *[WatchDog]* – используется для борьбы с зависанием, то есть с непредусмотренным зацикливанием программы, механизм может быть запрещен специальным *Fuse*-битом или программой;

4) снижение напряжения питания ниже заданного уровня *[Brown-Out Detector],* данный механизм выбирается *Fuse*-битом;

5) сброс по интерфейсу *JTAG*, данный механизм выбирается *Fuse*-битом.

Процессор

Процессор с точки зрения программиста представлен рядом явно адресуемых регистров (файл 8-битных регистров общего назначения *R0…R31*, регистр состояния *Status REGister* и регистр указатель стека *Stack Pointer*) и неявно используемого регистра счетчик команд *[Program Counter]*.

Старшие шесть РОН *R26…R31* используются как три 16 битных указателя: *X = R26*:*R27, Y = R28*:*R29, Z = R30*:*R31*, здесь «:» - знак конкатенации, обозначающий для указателя *X*, что в *R26* хранится старшая, а в *R27* – младшая часть адреса. Все три указателя используются при адресации памяти данных *SRAM*, для адресации к памяти программ *FlashROM* используется только указатель *Z.*

Регистр состояния *SREG* содержит битовые флаги (признаки), отражающие результаты арифметико-логических операций и влияющие на ход выполнения программ:

SREG.0 = C [Carry] – перенос в *C* из старшего 7-го разряда при сложении или заем из *C* при вычитании;

SREG.1 = Z [Zero] – нулевой результат;

SREG.2 = N [Negative] – отрицательный результат;

SREG. 3 = V [oVerflow] – переполнение дополнительного кода (для чисел со знаком) и т. д.

Указатель стека *SP [Stack pointer]* используется в механизме вызова подпрограмм для хранения адреса возврата и, при необходимости, данных. Детально этот механизм будет рассмотрен в следующих разделах.

Содержимое регистра счетчика команд *PC* служит для адресации очередной

команды в ПП. Оно автоматически увеличивается на единицу или двойку при выполнении команд арифметико-логической группы и группы передачи данных (число 1 или 2 определяется по длине самой команды). Если же встречается команда перехода (ветвления), содержимое счетчика команд изменяется в соответствии с указанием самой команды.

На рис. 1.2 приведена структура ядра МК, включающая память программ *FlashROM*, память данных *SRAM*, процессор *AVR CPU* и внутреннюю магистраль адресного пространства памяти данных с 8-разрядной шиной данных ШД8.

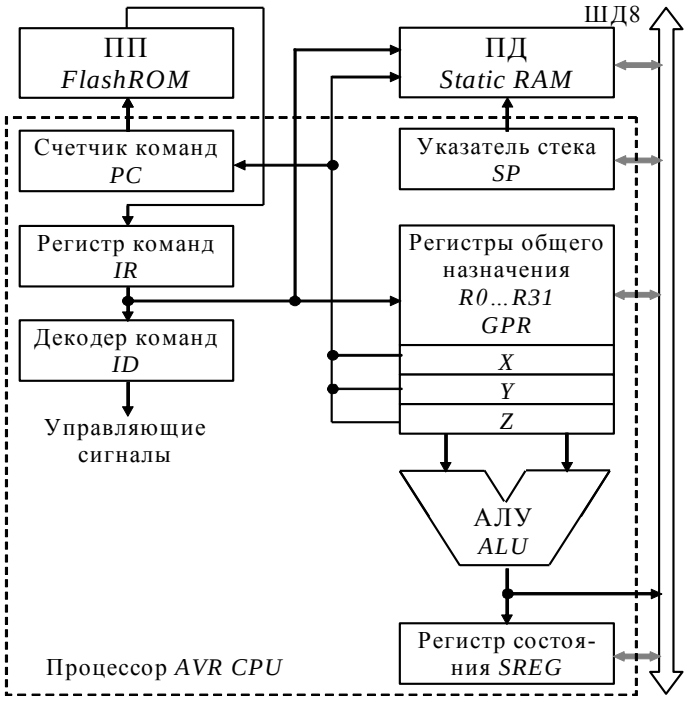


Рис. 1.2. Ядро и процессор

В состав процессора кроме перечисленных выше программно доступных элементов входят:

- арифметико-логическое устройство АЛУ - выполняет команды обработки данных, то есть собственно вычисления, поэтому определяет вычислительную производительность;

- регистр команд *[Instruction Register]* используется для размещения кода исполняемой команды; декодер команды *[Instruction Decoder]* анализирует формат команды, выделяет из нее код операции (КОП) и группы бит адресации данных или адресации в памяти программ. Адресация данных требует 4 или 5 бит для выбора РОН, 6 бит для области РВВ, от 9 до 16 бит в области *SRAM*;

- устройство управления (УУ) под управлением тактовых импульсов *fclk* реализует последовательное выполнение команд, в современных МП - конвейерное выполнение нескольких команд.

Выполнение каждой команды требует выполнения последовательности действий, называемой машинным циклом.

1. Выборка команды по адресу указателя команды и прибавление 1 или 2 к значению указателя *PC + 1/2* → *PC*.

2. Декодирование кода команды.

3. Выполнение команды, включающее возможную выборку данных и/или запись результата в указанную ячейку, а для команд перехода – модификация указателя команды *PC*.

В МП с конвейерным принципом выполнения команд под временем цикла понимают темп выполнения команд. Например, при 4-х этапном конвейере (AVR), время выполнения короткой команды составляет 4 цикла, но темп выполнения программы составляет одну команду за цикл. Исключение составляют команды перехода, нарушающие последовательное наращивание адреса команды. Темп их выполнения "сбивает" работу конвейера и увеличивает темп выполнения до 4…5 циклов соответственно.

Арифметико-логическая группа команд

Системы счисления и форматы данных

В 8-разрядных МК основным форматом данных является байт *[byte]*, состоящий из 8 бит *[bit]* данных. Часть регистров представляют объединение двух байт в одно слово *[word]*, еще реже встречаются адресные указатели из трех байт.

Как известно, один бит (двоичный разряд) может принимать только два значения: 0 (или "НИЗКИЙ", то есть сигнал с напряжением близким к нулю питания *GND*) и 1 (или "ВЫСОКИЙ", то есть сигнал с напряжением близким к напряжению питания *+VCC*).

При использовании данных в формате байта диапазон представляемых целых чисел зависит от интерпретации двоичного эквивалента. Если подразумеваются целые числа без знака, то диапазон представлен от 0 до 255 = 2^8 - 1. Для чисел со знаком диапазон от -128 = -2^7 до +127 = 2^7 - 1, старший (7-ой разряд) выступает как знаковый (0 - положительное число, 1 - отрицательное), отрицательные числа представлены в двоично-дополнительном коде. Такой способ представления удобен тем, что для выполнения операций сложения и вычитания как над числами со знаком, так и для чисел без знака подходят одинаковые машинные команды. Разница выявляется через анализ битов-признаков в SREG

Для операций умножения и деления беззнаковых и знаковых чисел требуются разные команды или алгоритмы.

Трансляторы языка *C/C++* используют для работы с байтом тип данных *char*, со словом - тип *int*, с двойным словом (4 байта) - тип *long*. Для обозначения беззнаковых чисел добавляется прилагательное *unsigned*, для знаковых - *signed.*

Для представления констант в исходных текстах программ используются как десятичные числа (по умолчанию), так и их двоичные (префикс *0b*) или шестнадцатеричные (префиксы *0x* или *$*, или суффикс *h*) эквиваленты. Например, 12 = *0x0C = $0C = 0Ch = 0b00001100*. Использование двоичного и шестнадцатеричного форматов удобно при работе с отдельными двоичными разрядами и при работе с адресами.

В Приложении 1 подробно рассмотрены системы счисления и форматы представления чисел.

Особенностью арифметико-логических группы команд в МК *AVR* является то, что в качестве операндов используются только регистры общего назначения. Это упрощает адресацию и позволяет выполнять почти все команды за один машинный цикл.

Большинство команд этой группы выполняются за 1 машинный цикл, имеют формат

1 слово (2 байта) и изменяют биты регистра состояния *SREG*.

Код команды (16 двоичных разрядов – слово *[word]*) состоит из кода операции и поля адресации одного или двух операндов. Для адресации одного РОН (*R0…R31*) достаточно 5 двоичных разрядов (2^5 = 32). В некоторых командах для экономии поля адресации используется только 4 разряда, что позволяет адресовать только 16 старших РОН *R16 … R31*. Под константу в коде команды отводится 6 (2^6 = 64) или 8 разрядов (2^8 = 256).

Арифметико-логические команды выполняют различные операции над знаковыми и беззнаковыми числами форматом 1 байт (8 бит). Исключение составляют команды, в мнемонику которых входит буква *W* – они оперируют с двухбайтными операндами, размещаемыми в смежных регистрах, например, *r25:24*. Все операции производятся только над регистрами общего назначения (т. е. используется так называемая регистровая адресация) и/или константой (абсолютная адресация).

Подгруппа арифметических команд.

Подгруппа команд арифметических операций выполняет такие базовые действия, как сложение (*ADD, ADC, ADIW*), вычитание (*SUB, SUBI, SBC, SBCI, SBIW*), инкремент (прибавление единицы – *INC*), и декремент (вычитание единицы – *DEC*).

Команда сложения ADD Rd, Rr выполняет сложение содержимого регистра-источника Rr с регистром-приемником Rd, сумма размещается в Rd. В мнемонической записи команды, как и в других двухоперандных командах первым указывается регистр-приемник, затем – регистр-источник.

Команды сложения и вычитания могут работать как с числами без знака (0...255), так и с числами со знаком в двоично-дополнительном формате (-128...+127) – это одни и те же команды, разница заключается лишь в интерпретации результата, для чего дополнительно используются биты-признаки регистра состояния *SREG*.

Пример операции сложения.

До операции *r16 = 0xfa., r17 = 0x08.*

add r16, r17 ; r16 + r17 → r16

После *r16 = 0x02., r17 = 0x08,* устанавливаются биты признаков *C = 1 и H = 1*.

Для беззнаковых чисел исходное значение *r16 = 0xfa* = 250 и r17 = *0x08* = 8, результат будет *r16* = 250 + 8 = 258 = 256 + 2 = *0x0102 = 0b1'0000'0010*. Он не помещается в один байт, поэтому в регистре-приемнике будет число *0x02*, единица из 8-го бита результата разместится в бите *C [Carry]* регистра *SREG*, выполняя роль признака переноса в старший байт. Если результат сложения использует старший байт, то при Carry = 1 следует прибавить 1 к содержимому старшего байта. Если старший байт не используется, имеем ошибку в результате вычислений.

Для чисел со знаком тому же двоичному числу *0xfa = 0b1111'1010* соответствует отрицательное десятичное -6, результат сложения -6 + 8 = 2, *C* = 1 означает заем из старшего байта. Старший байт надо уменьшить на 1.

Сложение и вычитание многобайтных чисел выполняются побайтно, начиная с младших байтов.

Например, пусть надо сложить число A1:A0 с числом B1:B0. Знак ‘:’ конкатенации означает объединение байтов в одно слово, то есть A0, B0 – младшие байты, A1, B1 – старшие байты складываемых чисел. Алгоритм сложения состоит из двух команд сложения. Сначала получаем сумму младших байтов

командой сложения *(add)* по схеме A0 + B0 → A0. Затем получаем сумму старших байтов A1 + B1 + Carry → A1 командой сложения с учетом переноса (*adc*), который мог возникнуть при выполнении предыдущей команды сложения.

Для AVR, пусть A0=R0, A1=R1, B0=R2, B1=R3.

```
ADD R0, R2      ; A0 + B0 → A0
ADC R1, R3      ; A1 + B1 + Carry → A1
```

Ангалогично, вычитание двухбайтного числа B1:B0 (R4:R3) из трехбайтного A2:A1:A0 (R2:R1:R0).

```
SUB R0, R3      ; A0 – B0 → A0
SBC R1, R4      ; A1 – B1 – Carry → A1
CLR R5          ; B2 ← 0
SBC R2, R5      ; A2 – B2 – Carry → A2
```

Команда деления отсутствует во всех МК серии *AVR*, умножения – в подгруппах *Tiny* и *Classic*.

Микроконтроллеры подсемейства *MEGA* выполняют операции умножения (*MUL, MULS, MULSU, FMUL, FMULS, FMULSU*) целых и дробных (*F*) чисел форматом один байт с учетом (*S*) и без учета знака (*U*). Умножение знаковых и беззнаковых чисел требует разных команд, отдельную группу составляют команды умножения дробных чисел.

В отсутствии команд умножения, для умножения многобайтных чисел и для выполнения операций деления используются алгоритмы с командами арифметических сдвигов (подробнее рассмотрены через несколько страниц). Команды сдвига вправо (*LSR, ROR, ASR*) эквивалентны делению на 2, команды сдвига влево (*LSL, ROL*) эквивалентны умножению на 2. Различия между сдвигами логическими *(LSx)* и циклическими *(ROx)* проявляются только в обработке крайних бит (нулевого и седьмого) и взаимодействии с битом *SREG.C*.

В языке C/C++ операции сдвига влево обозначаются ‘<<’, вправо – ‘>>’, детали операции зависят от формата данных. Операция ‘<<=’ – сдвиг влево с присвоением – эквивалентна команде *LSL*.

Листинг трансляции в первых двух строках показывает текст на языке C, в следующих трех – эквивалентный набор машинных команд:

```
11:         unsigned char pred=1;
12:         pred <<= 1;
+0000004D:   91800060    LDS       R24,0x0060     Load direct from data space
+0000004F:   0F88        LSL       R24            Logical Shift Left
+00000050:   93800060    STS       0x0060,R24     Store direct to data space
```

Например, для умножения беззнакового целого в формате байта A на константу 15 представим это число как сумму чисел $1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$: 15 = 8 + 4 + 2 + 1. Алгоритм состоит из однородных операций – умножения на 2 (сдвиг влево) и сложения. Рассмотрим (сначала) случай, когда результат умножения не превышает одного байта, то есть A ≤ 17. Пусть множимое A размещается в R16.

```
MOV     R17, R16        ; 1A
LSL     R16             ; 2A
ADD     R17, R16        ; 1A + 2A = 3A
LSL     R16             ; 4A
ADD     R17, R16        ; 3A + 4A = 7A
LSL     R16             ; 8A
ADD     R17, R16        ; 8A + 7A = 15A – результат умножения в R17
```

Если первую операцию (MOV) заменить обнулением R17 (CLR R17) и сложением (ADD R17, R16), то весь алгоритм состоит из повторения одинаковых пар операций, число повторений очевидно определяется числом разрядов множителя. При использовании другого множителя суммирование должно выполняться

только для разрядов множителя, содержащего 1.

Если результат может занимать два байта, операция умножения на два должна выполняться над двумя байтами, причем для переноса из младшего байта в старший используется SREG.C.

```
LSL     R16     ; младший байт множимого, седьмой бит заносится в SREG.C
ROL     R17     ; старший байт множимого, в нулевой бит заносится SREG.C
```

Деление байта на константу 2^x сводится к x сдвигам делимого вправо, например, при делении на 8 требуется три сдвига.

Если делимое A занимает один байт и находится в R16:

```
LSR     R16     ; A / 2 → R16
LSR     R16     ; A / 4 → R16
LSR     R16     ; A / 8 → R16
```

Если делимое A1:A0 занимает два байта и находится в R16:R17, то сдвигать надо пару регистров через бит SREG.C:

```
LSR     R16     ; старший байт À1 / 2
ROR     R17     ; младший байт À0 / 2
LSR     R16     ; старший байт À1 / 4
ROR     R17     ; младший байт À0 / 4
LSR     R16     ; старший байт À1 / 8
ROR     R17     ; младший байт À0 / 8
```

800 / 8 = 100, 800 = 0x320

Алгоритм деления на числа не кратные степеням двойки гораздо сложнее, он строится по приципам деления «в столбик» в двоичном виде и состоит из нескольких циклов.

1. Сдвигаем делитель влево до тех пор, пока он меньше делимого и считаем число сдвигов.
2. Далее вычитаем модифицированный делитель из делимого, записываем 1 в разряд, номер которого равен числу сдвигов.
3. Если остаток вычитания не равен нулю, сдвигаем делитель вправо до тех пор, пока он не станет меньше остатка, но не более зафиксированного числа сдвигов.
4. Переходим к п. 2

Например, 251 разделить на 10. Представим делимое и делитель в двоичном виде: 251 = 0b11111011, 10 = 0b1010. Ясно, что 251 / 10 = 25 и 1 в остатке.

Реализация такого алгоритма требует использования команд сравнения и условных ветвлений (переходов), сам алгоритм удобно оформить в виде подпрограммы. (Надо дать ссылки на готовые подпрограммы из AppNote)

Тестирование ASR: -4 / 2 = -2: -4 = 0xFC, 0xFE = -2; -100 / 2 = -50, -100 = 0x9C, 0xCE = -50!!!

Подгруппа логических команд.

AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR,

Первую подгруппу образуют однооперандные команды, изменяющие содержание указанного РОН:

- *COM* инвертирует каждый бит, эквивалент операции *C "~",* например, из числа *0x01 = 0b00000001* получим *0xfe = 0b11111110*;

- *NEG* меняет знак числа в двоично-дополнительном коде, например, из числа *0x01* = 1получим *0xff* = -1.

- *CBR* обнуляет все биты, например, из числа *0x01* получим *0x00*;

- *SBR* устанавливает в «1» все биты, например, из числа *0x01* получим *0xff*.

Вторая подгруппа образована двухоперандными командами поразрядной обработки. В командах с суффиксом *xxxI* второй операнд представлен 8-битной константой в теле команды, в остальных оба операнда находятся в РОН:

- *AND* и *ANDI* выполняют операцию «поразрядное логическое И»: результат «1» только если оба бита «1», эквивалент операции *C "&="*; например, *R0 = 0b01101001*, *R1* =

0b00001111, после команды *AND R0, R1* результат *R0 = 0b00001001*;

```
Пример сброса бита 2 PBB PORTB с использованием команды ANDI:
in      R16, PORTB  ;Чтение PBB PORTB в POH R16
andi    R16, ~(1<<2);Сброс бита 2 R16
out     PORTB, R16  ;Запись POH R16 в PBB PORTD
; 3 команды (3 слова ПП), 3 м.ц., 1 POH
```

- *OR* и *ORI* выполняют операцию «поразрядное логическое ИЛИ»: результат «1», если хотя бы один бит «1», эквивалент операции *C "|="*; например, *R0 = 0b01101001*, *R1 = 0b00001111*, после команды *OR R0, R1* результат *R0 = 0b01101111*;

```
Пример установки бита 2 PBB PORTB с использованием команды ORI:
in      R16, PORTB  ;Чтение PBB PORTB в POH R16
ori     R16, (1<<2) ;Установка бита 2 R16
out     PORTB, R16  ;Запись POH R16 в PBB PORTD
; 3 команды (3 слова ПП), 3 м.ц., 1 POH
```

- *EOR* выполняет операцию «поразрядное логическое исключающее ИЛИ» (иначе «сумма по модулю 2»): результат «1» если оба бита не совпадают, эквивалент операции *C "¬="*; например, *R0 = 0b01101001*, *R1 = 0b00001111*, после команды *EOR R0, R1* результат *R0 = 0b01100110*.

```
Пример инверсии бита 0 PBB PORTD с использованием команды EOR:
ldi     r17, (1<<0) ;Загрузка в POH R17 константы для инверсии бита 0
in      R16, PORTD  ;Чтение PBB PORTD в POH R16
eor     R16, R17    ;Инверсия 0-бита R16
out     PORTD, R16  ;Запись POH R16 в PBB PORTD
; 4 команды (4 слова ПП), 4 м.ц., 2 POH
```

Подгруппа команд сдвига.

ASR, LSL, LSR, ROL, ROR, SWAP

При сдвигах содержимое двоичных разрядов адресуемого POH сдвигается на один разряд:

- вправо [*Right*], *Rd*[*n*+1] → *Rd*[*n*], *n* = 0…6, команды *ASR, LSR, ROR,*

- или влево [*Left*], *Rd*[*n*] → *Rd*[*n*+1], *n* = 6…0, команды *LSL, ROL*.

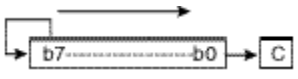
По порядку сдвигов 7-го и 0-го битов различают:

- циклический сдвиг [*ROx - ROtate*], когда содержимое POH сдвигается «по кругу» через 9-ый бит *C* [*SREG.Carry*], *ROL*: *C* → *Rd*[0], *Rd*[7] → *C*, *ROR*: *C* → *Rd*[7], *Rd*[0] → *C*,

если перед операцией R16 = 0b10101100, SREG.C = 1, то после двух команд ROL R16 содержимое R16 = 0b10110011, SREG.C = 0;

- логический сдвиг [*LSx – Logic Shift*], когда выдвигаемый бит помещается в *SREG*.C, а вдвигаемый равен нулю, эквивалент операции *C "<<"*или *">>";*

- арифметический сдвиг вправо *ASR [Arithmetic Shift Right]*, который отличается от *LSR* тем, что копируется (сохраняется) содержимое старшего знакового разряда, что удобно при выполнении деления на 2 чисел со знаком.



Особняком стоит команда *SWAP*, при выполнении которой меняются местами тетрады (старшие и младшие 4 бита).

```
R(7:4) ← Rd(3:0), R(3:0) ← Rd(7:4)
```

1.4. Структура и адресация памяти программ. Ветвления, циклы, подпрограммы, и группа команд передачи управления

Адресация в цифровых устройствах использует двоичное кодирование. Для адресации двух ячеек достаточно одной линии (адреса 0, 1), четырех ячеек – двух линий (адреса 0, 1, 2, 3). В общем случае n линий адресуют до $N = 2^n$ ячеек, для адресации N ячеек необходимо число линий равное ближайшему большему целому $n = log_2(N)$.

8 разрядов адресуют лишь 256 ячеек, поэтому в 8-разрядных МК для адресации памяти программ (нередко и для памяти данных) используют 16-разрядные регистры, позволяющие адресовать до 65536 = 64 К ячеек. Здесь и далее следует отличать 1 к = 1000 и 1 К = 1024 = 2^{10}.

Режимы адресации описывают способы кодирования (в машинной команде) адреса данных или перехода в программе. Различают прямую и непрямую (косвенную) адресацию.

При прямой адресации адрес является константой и входит в код команды. При адресации до 64 К адрес занимает 2 байта, поэтому длина команды получается 3-4 байта. Разделение большого адресного пространства на подобласти уменьшает длину команды, например, адресация только РОН, регистров ввода/вывода и пр.

Многочисленные варианты косвенной адресации используют регистры-указатели, содержимое которых формируется до выполнения команды с этим типом адресации или модифицируется во время выполнения. Косвенная адресация позволяет работать с вычисляемыми, а не только константными адресами, без чего работа с массивами была бы затруднительна.

Области памяти

Как было отмечено, память МК разделяется на память программ (ПП) *[Program Memory]* и память данных (ПД) *[Data Memory]*.

Память программ обычно энергонезависима (постоянное запоминающее устройство – ПЗУ), так как управляющая программа редко модифицируется. Память данных состоит из энергозависимой части (оперативное запоминающее устройство – ОЗУ) – собственно рабочей ПД, и энергонезависимой (перепрограммируемое ПЗУ) для хранения редко изменяемых констант (настроек прибора).

Области адресации ПП и ПД разделены за счет выделения для ПП собственной магистрали выборки. Это позволяет производить одновременную выборку команд и данных.

Память программ

Память программ служит для хранения кодов программы и констант. Физическим носителем ПП обычно выступает один из видов энергонезависимого *[nonvolatile]* постоянного запоминающего устройства (ПЗУ) *[ROM: Read Only Memory]*.

В МК семейства *AVR* память программ представлена ячейками размером 16 бит = 2 байта = 1 слово *[word]*, количество которых варьируется от 512 до 65 536 слов (1 … 128 килобайт), что требует от 9 до 16 разрядов адреса соответственно. Это и определяет разрядность счетчика команд (СК) *[PC: Program Counter]*.

При размещении в ПП констант адресация идет не к словам, а к байтам, здесь 16 бит адреса недостаточно при обращении по адресам больше 65 535. Физическим носителем

ПП является *резидентное* (встроенное в МК) электрически стираемое и программируемое постоянное запоминающее устройство (ЭСППЗУ) *[FlashROM]*.

Команды передачи управления (ветвлений)

Эти команды (в отличии от других) могут изменять последовательную выборку команд из памяти (PC + 1 → PC) и служат для организации ветвлений (условных и безусловных) и механизма подпрограмм. Их выполнение нарушает работу конвейера процессора AVR, поэтому они имеет большее время выполнения, по сравнению с командами других групп (2…4 цикла).

Ветвления безусловные представлены командами перехода и различаются лишь способом адресации перехода *RJMP* (относительная 12 бит), *IJMP* (косвенная Z-указатель 16 бит), *JMP* (прямая 16 бит).

Относительная адресация в команде *RJMP ra12* работает по схеме PC + ra12 + 1 → PC, занимает в ПП одно слово, но ее область действия ограничена объемом памяти программ 2048 слов. При программировании на ассемблере в качестве операнда указывается символьная метка, если ее адрес отстоит от текущего значения PC более чем на 2048 слов, траслятор выдаст сообщение об ошибке.

Прямая адресация в *JMP a16 (a16 → PC)* адресует до 64 килослов, но занимает в памяти 2 слова и выполняется на один цикл дольше (3 цикла). Здесь также обычно используется символьная метка, некоторые трансляторы сами выбирают какую из команд использовать – JMP или RJMP.

Косвенная адресация в *IJMP (Z → PC)* требует предварительной загрузки адреса перехода в регистр-указатель Z (две команды *LDI*), ее удобно использовать при вычисляемом адресе (таблица подпрограмм).

Механизм подпрограмм обеспечивает многократное выполнение отдельного алгоритма или его части при экономии памяти программ. Использование механизма подпрограмм наряду с механизмом макросов (см. Приложение 4, с 150) развивает модульность программы и упрощает процесс отладки и развития программы.

Команда вызова подпрограммы выполняет переход по адресу первой команды подпрограммы, но предварительно сохраняет в стеке адрес команды, следующей за ней. Команды *CALL, RCALL, ICALL* различаются только способами адресации перехода, аналогично командам перехода *JMP*.

Тело подпрограммы (п/п – *subroutine*) состоит из одной и более машинных команд. Адрес расположения первой команды называется точкой входа, для удобства она обозначается символьной меткой. В этом случае команда вызова п/п в качестве операнда использует эту метку для вычисления адреса перехода.

Завершает текст подпрограммы команда возврата из п/п (*RET*). Она использует для перехода адрес из стека, что обеспечивает попадание на команду, следующую за командой вызова (при условии, что данные в стеке не были нарушены при выполнении подпрограммы).

Стек занимает часть памяти данных (SRAM), в AVR МК для него отводят «верхние» адреса. Настройка стека на ассемблере является заботой программиста. До вызова первой подпрограммы требуется загрузить в регистр указателя стека SP [Stack Pointer] адрес начала стека:

```
LDI     R16, HIGH(RAMEND) ; RAMEND – адрес последней ячейки RAM
OUT     SPH, R16
```

```
    LDI    R16, LOW(RAMEND)
    OUT    SPL, R16
```

Применение команд вызова п/п и возврата из п/п приводит к некоторым затратам времени (7–10 м. ц.) и памяти программ (1–2 слова на вызов и 1 слово на возврат из п/п), поэтому при длине п/п менее 5–8 команд эффективнее использовать механизм макрокоманд.

При необходимости передавать в п/п параметры и/или получать из п/п некоторые данные (результат) используются различные механизмы. Один из самых простых и распространенных – через РОН: п/п «знает» номера регистров, через которые получает входные параметры и по которым должна расположить возвращаемые значения.

На языке C/C++ механизм вызова подпрограмм является составной частью функций, за настройку стека «отвечает» транслятор.

Ветвления условные можно разделить на подгруппу ветвлений по состоянию флагов состояния процессора и подгруппу пропуска следующей команды.

Первая из указанных подгрупп представлена командами *BRxx s, ra7* (от *BRanch* – ветвление) с относительным механизмом адресации 7 бит. Эти команды проверяют заданное состояние указанного флага состояния процессора SREG.s, если соответствует, выполняется переход PC + ra7 → PC, иначе выполняется следующая команда PC + 1 → PC.

Состояние флагов процессора изменяется при выполнении арифметико-логических команд, а также команд сравнения *TST, CP, CPC, CPI.*

Например, при организации цикла на 10 повторений команда `BRNE (Branch if Not Equal)` выполняет переход по метке, если результат предыдущей операции – вычитания единицы из R16 не равен нулю, иначе цикл завершается. Равенство нулю фиксируется в бите SREG.Z, более длинный синтаксис – BRBC Zero, CYCLE:

```
    LDI    R16, 10     ; Загрузка в РОН числа повторений
CYCLE:                 ; Метка начала цикла
    ;ТЕЛО ЦИКЛА
    DEC    R16         ; Уменьшение на 1 счетчика повторений
    BRNE   CYCLE       ; Проверка результата уменьшения, если не ноль, повтор
```

Команды пропуска (*skip*) проверяют:

- состояние бита РОН (*SBRS Rr, b* – пропустить, если бит *b* регистра *Rr* установлен, *SBRC Rr, b* - ... сброшен),
- либо бита РВВ (*SBIS P, b, SBIC P, b*),
- либо сравнивает содержимое двух РОН (*CPSE Rd, Rr* – сравнить и пропустить, если *Rd == Rr*),

при выполнении условия пропускают следующую команду (*PC* + 2/3 → *PC – длина перехода зависит от длины следующей команды*), иначе выполняют следующую (*PC* + 1 → *PC*). В этих командах адреса перехода предопределены, поэтому нет операнда адреса перехода.

Например, для инверсии бита 0 РВВ PORTD команда sbic анализирует его текущее состояние, если бит в нуле, то пропускается команда перехода по метке AA1 и выполняется команда установки бита (sbi), иначе выполняется команда сброса бита (cbi).

```
      sbic  PORTD, 0    ;
      rjmp  AA1
      sbi         PORTD, 0    ;
      rjmp  AA2
AA1:  cbi         PORTD, 0    ;
AA2:  ; 5 команд (5 слов ПП), время 5/6 м.ц., РОН не исп.
```

1.5. **Структура и адресация памяти данных. Группа команд передачи данных**

Память данных МК семейства *AVR* состоит из двух физически различных устройств:
1) резидентного энергозависимого *[volatile]* статического оперативного запоминающего устройства (СОЗУ) *[SRAM: Static Random Access Memory]*, разрядностью 8 бит (1 байт) и количеством ячеек от 96 до 4096 = 4 Кбайт; в некоторых моделях МК возможно наращивание ОЗУ внешними микросхемами с общим объемом не более 64 Кбайт;
2) резидентного энергонезависимого *[nonvolatile]* электрически стираемого и записываемого ($t_{ЗП}$ = 2…4 мс) постоянного запоминающего устройства (ЭСППЗУ) *[EEPROM: Electrically Erasable Programmable Read-Only Memory]*, разрядностью 8 бит (1 байт) и количеством ячеек от 64 до 4096 = 4 Кбайт; целесообразно использовать как память настроек (параметров).

Адресация памяти. Все три перечисленные физические области памяти (*FlashROM, SRAM, EEPROM*) имеют раздельные адресные пространства (Табл. 1.1). Разделение обеспечивается системой команд и режимов адресации. Кроме того, в области *SRAM* с помощью режимов адресации выделяются две подобласти:
1) файл регистров общего назначения (РОН) *R0 ... R31* занимает адреса 0 … 31 (= *0x1f*);
2) файл регистров ввода/вывода (РВВ) *[I/O: Input/Output]* состоит из 64 ячеек и занимает адреса 32 … 95 *(0x20 ... 0x5f).*

Таблица 1.1. Карта памяти МК семейства AVR

<table>
<tr><td>Память программ</td><td colspan="4">Память данных (ячейка – 1 байт)</td></tr>
<tr><td rowspan="5">Встроенная FlashROM
Объем: от 1024 байт
до 256 КБ
Ячейка:
2 байта – для программы
1 байт – для данных</td><td>Адреса</td><td>Энергозависимая</td><td></td><td>Энергонезависимая</td></tr>
<tr><td>0x00
…
0x1f</td><td rowspan="2">РОН (Registers)</td><td>0
…
31</td><td rowspan="4">Встроенная EEPROM
64 Б … 4 КБ,
Адреса:
0x00
...
E_END</td></tr>
<tr><td>0x20
…
0x5f</td><td>0
…
63</td></tr>
<tr><td>0x60
…
S_END</td><td>Встроенная SRAM
(Data)
0…8 КБ</td><td></td></tr>
<tr><td>S_END+1
…
0xff</td><td>Внешняя SRAM
0…64 КБ</td><td></td></tr>
</table>

Область РОН является наиболее интенсивно используемой областью памяти данных (ПД): только в ней располагаются операнды арифметико-логической группы команд и реализуются сложные режимы адресации. Пары регистров *R27:R26=X, R29:R28=Y, R31:R30=Z* могут использоваться как 16-разрядные для адресации памяти данных – *X, Y, Z* и памяти программ – *Z*. Остальные области памяти доступны только командам передачи данных.

Регистры ввода/вывода (РВВ), иначе называемые регистрами специальных функций (РСФ) *[SFR: Special Function Register]*, позволяют следить за состоянием и управлять аппаратными ресурсами МК: процессора, *EEPROM*, системы прерываний, параллельных портов и других встроенных периферийных устройств. К этой подобласти относятся регистр состояния *SREG* и 16-разрядный регистр-указатель стека *SP [Stack Pointer]*

(состоит из двух 8-разрядных регистров *SPH* и *SPL*).

Стеком называется область памяти данных, работающая по принципу «последним записан – первым считан» *[LIFO: Last Input – First Output]*. Используется при выполнении подпрограмм. При вызове подпрограммы в стеке аппаратно сохраняется адрес следующей за вызовом команды, при возврате из подпрограммы этот адрес восстанавливается из стека.

Также стек может использоваться программистом или транслятором для передачи параметров в подпрограммы или из подпрограммы. В подпрограммах обслуживания прерываний стек используется для сохранения и восстановления контекста.

В данной серии МК стек «растет вниз», начиная со старших адресов *SRAM*, поэтому при помещении данных в стек используется *постдекремент* указателя стека, т. е. уменьшение значения *SP* на 1 после записи, обозначается (*SP*)–, при извлечении из стека – *прединкремент*, т. е. увеличение значения *SP* на 1 перед записью, обозначается +(*SP*).

Группу команд передачи данных можно разделить на подгруппы, в зависимости от обслуживаемых областей памяти:

1. РОН ↔ константа: `LDI`.

   Пример: `LDI Rd*,K8 ;` Загрузка 8-разрядной константы в РОН Rd*=R16…R31.

2. РОН ↔ РОН: `MOV`.

   Пример: `MOV R0, R16 ;` Копирование содержимого РОН R16 в R0.

3. РОН ↔ РВВ: `IN, OUT`.

   Пример: `IN Rd, P ;` Копирование содержимого РВВ P=0…63 в РОН Rd=R0…R31.

   Пример: `OUT P, Rd ;` Копирование содержимого РОН Rd=R0…R31 в РВВ P=0…63.

4. РОН ↔ память данных (включая подобласти РОН и РВВ):
   `LD, LDD, LDS, ST, STD, STS, PUSH, POP`.

   Пример: `LD Rd, X ;` Загрузка содержимого байта SRAM *по адресу (R26:R27)=(X) в РОН Rd.*

   Пример: `STS A16, Rr ;` Сохранить содержимое РОН Rr в SRAM по адресу (A16).

   Пример: `PUSH Rr ;` Протолкнуть содержимое РОН Rr в стек.

   Пример: `POP Rd ;` Извлечь содержимое байта из стека в РОН Rd.

5. РОН ← память программ (константа): `LPM, ELPM`.

   Пример: `LPM ;` Загрузка содержимого байта FlashROM по адресу (R30:R31)=(Z) в РОН R0 (адрес < 65536).

   Пример: `ELPM Rd,Z ;`Загрузка содержимого байта FlashROM по адресу (Z) в РОН Rd (адрес > 65535).

Режимы адресации данных в группе команд передачи данных имеют наибольшее разнообразие:

1. Абсолютная адресация – код данных (константа) входит в код команды, подгруппа 1.

   Пример: `K8` – 8-разрядная константа (0…255).

2. Прямая (непосредственная) адресация – код адреса входит в код команды, подгруппы: 1 – 5.

   Пример: `Rd` – 5-разрядный адрес РОН приемника R0…R31. Разновидность прямой адресации – регистровая. `Rd*` – 4-разрядный адрес РОН R16…R31.

   Пример: `P` – 6-разрядный адрес РВВ = 0…63. Разновидность прямой адресации – регистровая.

   Пример: `A16` – 16-разрядный адрес SRAM = 0…65535. Увеличивает формат команды на 1 слово (2 байта).

3. Косвенная адресация – код адреса (или его часть) находится в одном из регистров-указателей Z, Y, X, с разновидностями:

3.1. Простая косвенная – содержимое регистра указателя не изменяется во время выполнения команды, подгруппы 4, 5.

Пример: `Y=R28:R29` – 16-разрядный адрес SRAM.

Пример: `Z=R30:R31` – 16-разрядный адрес FlashROM.

3.2. Относительная косвенная (индексная) – то же, что и для простой, но исполнительный адрес образуется суммированием содержимого регистра-указателя и константы (индекса) в коде команды (6 разрядов, 0…63), подгруппа 4.

Пример: `LDD Rd, Z+i6` ;Загрузить в Rd содержимое байта SRAM по адресу `(R30:R31)+(i6), i6=0…63.`

3.3. Косвенная с преддекрементом – содержимое регистра-указателя сначала уменьшается на 1, а затем производится обращение по полученному адресу, подгруппа 4.

Пример: `LD Rd, -Y ; Y ← Y–1, Rd ← (Y).`

Пример: `PUSH Rr ; (SP)← Rr, SP← SP–1.` Постдекремент!

3.4. Косвенная с постинкрементом – сначала обращение по адресу регистра-указателя, а затем содержимое регистра-указателя увеличивается на 1, подгруппа 4.

Пример: `ST X+, Rr ; (X) ← Rr, X ← X+1.`

Пример: `POP Rd ; SP ← SP+1, Rd ← (SP).` Прединкремент!

Общие черты группы команд передачи данных:

– не изменяют состояние флагов в регистре состояния *SREG*;

– формат команд: для большинства – 1 слово, 2 слова – для команд с непосредственной адресацией;

– время выполнения зависит от области действия: 1 машинный цикл – для команд 1–3-й подгрупп (РОН, РВВ и константы), 2 м. ц. – для 4-й подгруппы (*SRAM*) и 3 м. ц. – для 5-ой подгруппы (*FlashROM*).

Команды работы с битами

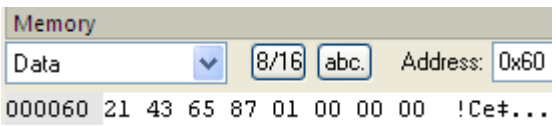
Система команд содержит ряд команд, обеспечивающих изменение состояния отдельных битов и анализ состояния отдельных битов:

– *SBI [Set Bit], CBI [Clear BIt]* – установить, очистить указанный бит РВВ;

– *SBIS [Skip if Bit Input/output register Set], SBIC [Skip if Bit Input/output register Clear]* – пропустить следующую команду, если указанный бит РВВ установлен, сброшен.

В тех случаях, когда последовательность действий, требуемая при заданном состояния бита РВВ, не укладывается в одну машинную команду, сразу после команды *SBIS / SBIC* ставится команда безусловного перехода *RJMP* с адресом перехода к действиям, выполняемым при невыполнении указанного в предыдущей команде условия.

При размещении в памяти данных многобайтных чисел в МК семейства AVR используется принцип размещения младшего байта по меньшему адресу, а старшего – по большему адресу.



Например, unsigned long ll = 0x87654321L;

Подобный порядок используют регистры-указатели, размещаемые по старшим адресам РОН. Например *X = XH:XL = R27:R26*, в этой записи знак “:” называют знаком *конкатенации*, то есть объединения регистров в 16-битный указатель.

Доступ к области EEPROM обеспечивается через группу регистров ввода/вывода:
– *EEAR [EEPROM Address Register]* - регистр адреса, при объеме *EEPROM* более 255 байт состоит из двух 8-разрядных регистров *EEARH:EEARL*;
– *EEDR [EEPROM Data Register]* - регистр данных;
– *EECR [EEPROM Control Register]* - регистр управления, состоящий из следующих битов:

– *EERE*(0) *[Read Enable]* – программная установка в «1» инициирует чтение, по окончании чтения аппаратно сбрасывается в «0»;

– *EEWE*(1) *[Write Enable]* – программная установка в «1» инициирует запись при *EEMWE=1,* сброс в «0» осуществляется аппаратно по окончании операции записи (2…4 мс);

– *EEMWE*(2) *[Master Write Enable]* – предварительное разрешение записи по «1», аппаратный сброс через 4 м. ц. после установки;

– *EERIE*(3) *[Ready Interrupt Enable]* – разрешение прерывания («1») по готовности, т. е. окончанию записи.

Группа команд работы с битами

Значительная доля задач управления оперирует не с числами, а с битами. Поэтому в современных МК всегда присутствует набор команд для этой цели.

В МК AVR к таким командам относят, как уже рассмотренные ранее команды:

- сдвиги LSL, LSR, ROL, ROR, ASR, SWAP,
- управление битами PBB SBI, CBI,

так и не рассмотренные:

- сохранение / восстановление бита РОН в SREG.T: BLD Rd, b / BST Rr, b,
- установка / очистка бита регистра состояния SREG.S: BSET S / BCLR S.

Особняком стоят три команды, которые есть в любой системе команд.

NOP – нет операции, используется для формирования фиксированных задержек, 1 м.ц.

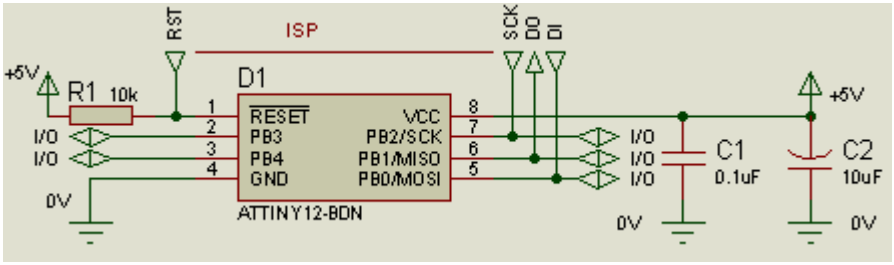
SLEEP – «спать», то есть перевод МК в режим пониженного энергопотребления. Это актуально для мобильных устройств с ограниченной энергией химического источника питания. Параметры режима задаются предварительно битами PBB MCUCR / MCUCSR.

WDR – сброс сторожевого таймера. Сторожевой таймер используется для борьбы с зависаниями МК в основном цикле, вызванными попаданием в частный цикл опроса без выхода.

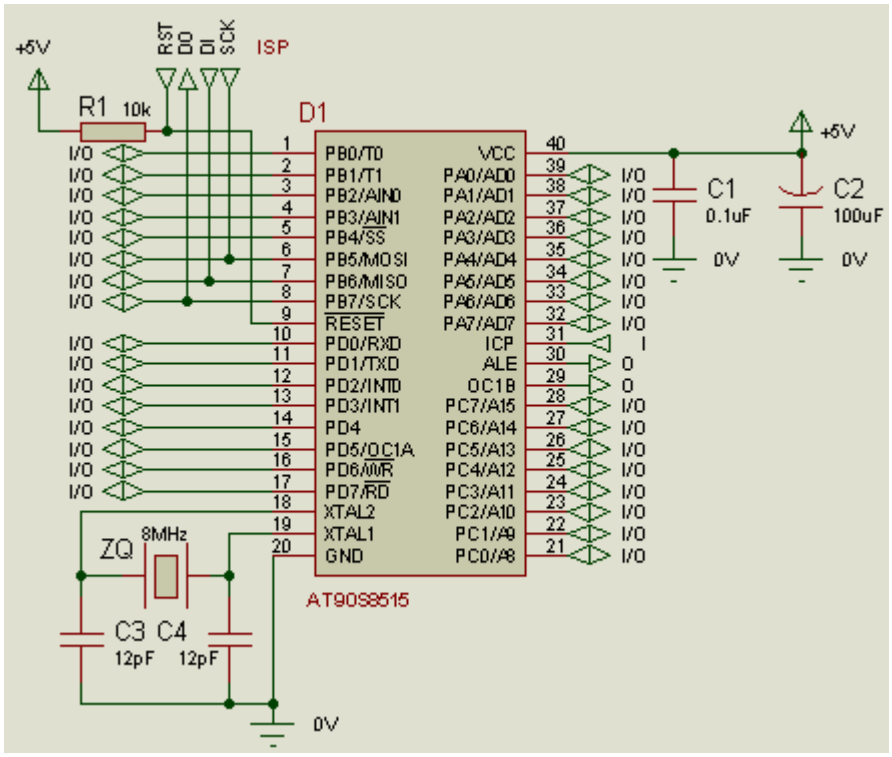
1.6. Порты ввода/вывода. Типовая схема включения МК. Структура управляющей программы

Типовые схемы включения и паспортные данные МК

На рис. 1.3 приведены типовые электрические схемы включения МК семейства *AVR* (обозначен D1), на рис. 1.3, а – для модели с самым малым числом выводов (ATTINY12 – 8 ног), на рис. 1.3, б – для модели с числом выводов 40 (AT90S8515).



а



б

Рис. 1.3

Напомним, что любая электрическая схема состоит из условных графических обозначений компонентов, линий элетрических связей и дополнительных буквенно-цифровых надписей. Каждый компонент имеет определенное буквенное обозначение с порядковым номером: C1…C4 – конденсаторы, D1 – микросхема МК, R1 – резистор, ZQ – кварцевый резонатор. Рядом находятся необязательные обозначения типов компонентов (AT90KS8515) или их номиналов (10k – 10 килоОм, 0.1uF – 0.1 микроФарада).

Числа возле выводов обозначают номер вывода, жестко привязанный к расплложению вывода на корпусе микросхемы, порядок нумерации – против часовой стрелки, расположение первой ножки зависит от типа корпуса. Буквенные обозначения выводов («внутри» прямоугольника) позволяют понять назначение данного вывода, косая черта разделяет альтернативные функции вывода (ножки).

Выводы, обозначенные *VCC* и *GND* используются для подачи питания (+5 В и 0 В соответственно). Для фильтрации импульсного тока потребления используются конденсаторы – электролитический *C1* емкостью десятки-сотни микрофарад и керамический *C2* емкостью 0.1 мкФ. Последний обладает меньшим значениями паразитных параметров – последовательного сопротивления и индуктивности, при разработке печатной платы его следует располагать как можно ближе к выводам питания МК.

Для экономии места на электрических схемах нередко цепи питания не показывают, заменяя текстовыми комментариями: «Вывод 40 D1 подключить к +5 В, выод 20 D1 – к 0 В».

ВЫСОКИЙ уровень напряжения на входе *RESET* необходим для штатного выполнения программы, обеспечивается притягивающим резистором *R1*. НИЗКИЙ уровень входа *RESET* останавливает нормальную работу МК, переводит его в состояние

СБРОС. Это состояние позволяет использовать специальные режимы – программирования или отладки.

К выводам *XTAL1* и *XTAL2* (на рис. 1.3, б) подключен кварцевый резонатор *ZQ* - он задает частоту тактирования МК f_{CLK}. Керамические конденсаторы *C3, C4* по 12 - 22 пФ необходимы для частотной коррекции (выбора требуемой гармоники кварца). В современных моделях МК встроен тактовый RC-генератор. Это позволяет освободить два вывода (XTAL1, 2) и использовать их как входы/выходы общего назначения, что весьма актуально в МК с малым числом выводов.

Остальные выводы МК могут использоваться как входы/выходы портов или других устройств ввода/вывода, то есть входы/выходы общего назначения, на рис. 1.3, а их 5, на рис. 1.3, б – 32 + 3. Латинские буквы i (input) обозначают вход, o (output) – выход, i/o – вход/выход (эти обозначения не являются стандартными).

Функция последовательного внутрисхемного программирования (*ISP*) использует еще 3 вывода МК: *SCK* - вход тактов, *MOSI (DI)* - вход данных, *MISO* (*DO*) - выход данных. Программирование выполняется при НИЗКОМ уровне на входе *RESET*.

Паспортные данные МК, как и любых других электронных компонент, состоят из таблицы предельных эксплуатационных параметров, таблицы электрических параметров, таблицы динамических (временных и частотных) параметров, разнообразных диаграмм (зависимостей параметров от напряжения питания, частоты тактирования, температуры кристалла и пр.), схем расположения выводов в конкретном корпусе (цоколевка), тестовых схем и чертежей корпусов микросхемы.

Найти паспортные данные конкретной модели МК можно на сайте разработчика-производителя. Для семейства *AVR* это *www.atmel.com* в разделе *Products* → *Microcontrollers* → *AVR…* → *DataSheets* в файлах формата *pdf* (читаются программой *Acrobat Reader*).

Предельные эксплуатационные параметры *[Absolute Maximum Ratings]* определяются технологией производства и одинаковы для всех МК данного семейства (кроме оговариваемых особо исключений). Здесь определены диапазоны допустимых значений температуры корпуса, напряжения и токи выводов. Нарушение любого из значений параметров может привести к выходу микросхемы из строя.

Значения напряжений определяются по отношению к выводу *GND* (общий питания и сигналов) и нормируются по отношению к уровню напряжения питания, предельное значение которого составляет 6 В (здесь и далее – численные значения для МК AVR). Напряжение на всех выводах не должно превышать напряжение питания на 0.5 В, кроме входа RESET на котором допускается напряжение до +13 В, что необходимо в режиме высоковольтного программирования.

Токи через входы/выходы общего назначения (как втекающие, так и вытекающие) не должны превышать 40 мА. Через выводы питания текут как токи собственно потребления внутренних узлов МК, так и суммарные токи входов/выходов общего назначения, поэтому их предельные значения существенно больше (100 – 400 мА в зависимости от модели).

Остальные характеристики описывают свойства и «поведение» МК при условии соблюдения запаса по всем предельным параметрам, а также дополнительно оговариваемых условиях (напряжение питания, температура и пр.).

Важно отметить общие черты формы описания этих параметров. Кроме краткого

символьного обозначения, развернутого наименования и размерности, обязательно существует графа условий измерения данного параметра, а графа значений параметра разбита на три подграфы с минимальным, типовым и максимальным значениями. Это связано со значительным разбросом свойств полупроводниковой структуры и взаимовлиянием параметров. Допускается указывать хотя бы одно значение – «наихудшее», например, максимальное значение напряжения НИЗКОГО уровня, минимальное – ВЫСОКОГО уровня.

Статические электрические параметры *[DC Characteristics]* описывают значения напряжений и токов выводов в определенных (длительных) состояниях.

Система символьных обозначений использует букву *V* для обозначения напряжений, *I* - для токов, подстрочные буквы: $_I$ - вывод в режиме входа *[input]*, $_O$ - вывод в режиме выхода [*output*], $_L$ - режим НИЗКОГО уровня сигнала *[low]*, $_H$ - режим ВЫСОКОГО уровня сигнала *[high]*.

Логика КМОП и ток потребления МК

Базовым элементом построения МК AVR является КМОП-инвертор, схема которого представлена на рис. 1.3, снизу. Она состоит из двух МОП-транзисторов, вверху *p*-канальный (*VT1*), внизу *n*-канальный (*VT2*). У этих транзисторов объединены затворы (вход инвертора) и стоки (выход инвертора), исток *VT1* подключен к *Vcc*, исток *VT2* - к *GND*.

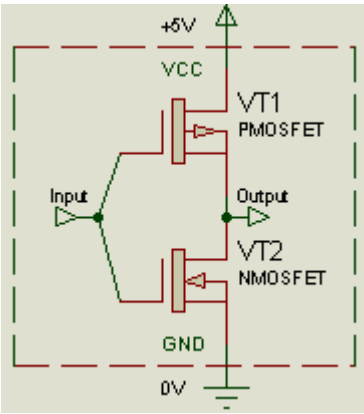


Рис. 1.4

Если на вход (затворы) подать ВЫСОКИЙ уровень, то откроется *n*-канальный *VT2*, напряжение на выходе будет стремиться к *GND* (НИЗКИЙ). Если подключить нагрузку между выходом и *Vcc*, через открытый канал *VT2* потечет втекающий ток *Iol*.

Если на вход (затворы) подать НИЗКИЙ уровень, то откроется p-канальный *VT1*, напряжение на выходе будет стремиться к *Vcc* (ВЫСОКИЙ). Если подключить нагрузку между выходом и *GND*, через открытый канал *VT1* потечет вытекающий ток *Ioh*.

Следовательно, в статике открыт только один из двух транзисторов, пути протекания тока от *Vcc* к *GND* нет, поэтому КМОП каскады, нагруженные друг на друга не потребляют постоянного тока *Icc dc* --> 0.

При переключении необходимо заряжать и разряжать входные емкости затворов, кроме того переключение транзисторов происходит не мгновенно, в какой-то момент оба канала приоткрыты. Все это приводит к импульсному потреблению тока. С ростом частоты тактирования МК амплитуда этих импульсов не растет, но эффективное значение растет практически по линейному закону *Icc ac = fclk * k*.

Следует принять во внимание, что при подключении к выходу подобного каскада значительной омической нагрузки ее ток добавляется к току питания. Эта часть тока определяется только сопротивлением нагрузки и формой тока нагрузки. Вытекающий ток нагрузки втекает в МК через вывод *Vcc*, втекающий – через вывод *GND*.

Параллельный 8-разрядный порт

Основной элемент подсистемы ввода/вывода МК. Состоит из внешних выводов (ножка = *[pin]*, до 8 в одном порту), которые могут использоваться как для ввода, так и для вывода цифровой информации и трех программно-доступных регистров (РСФ), обеспечивающих индивидуальную настройку каждой из 8 ножек:

–*DDRx [Data Direction Register x]* – регистр направления порта x, доступен по записи и чтению, обеспечивает настройку (конфигурирование) вывода как входа (0) или выхода (1);

–*PORTx* – регистр данных порта x, доступен по записи и чтению, определяет уровень выводимого сигнала при операциях вывода, при операциях ввода определяет тип входа – высокоомное *Hi-Z* (третье) состояние (0) или подключение к ножке внутреннего подтягивающего *[pull-up]* (к положительному полюсу питания) резистора (1);

–*PINx* – регистр состояния вывода (ножки), доступен только по чтению, связан с ножкой через триггер Шмита; здесь x – одно из порядковых имен портов (*A, B, C, D*, …).

Сами внешние выводы (ножки) имеют наименование *Pxy*, где *x* – имя порта (*A, B, C, D*, …), *y* – номер вывода в порту (0, 1, … 7), отдельные разряды регистра *DDRx* именуются *DDxy*, регистра *PORTx – Pxy*, регистра *PINx – PINxy*.

Таблица 1.2. Состояния порта ввода/вывода

DDRx	PORTx	PINx	Состояние
0	0	Внешняя схема	Вход высокоомный
0	1	Внешняя схема и Rpu	Вход с притяжкой к VCC
1	0	Низкий уровень с учетом нагрузки	Выход = 0
1	1	Высокий уровень с учетом нагрузки	Выход = 1

При старте МК аппаратно обнуляются все разряды регистров *DDRx* и *PORTx*, это соответствует настройке всех выводов портов в режим ввода в высокоомном состоянии (Табл. 1.2). Выбор такой настройки по умолчанию не случаен – он гарантирует наименьшую верятность повреждения выводов порта.

На этапе инициализации выполняется запись «единиц» в определенные разряды регистра/ов *DDRx, PORTx* для конфигурирования выходов (начальное состояний 0 или 1) и входов (*Hi-Z* состояние, то есть «подтягивание к плюсу»).

В ходе работы МК программа может изменять как состояние выходов, так и саму настройку каждого вывода, при этом следует учитывать ограничения внешней схемотехники. Изменение состояния одного выхода *Pxy* порта *PORTx* производится командой *CBI* или *SBI PORTx, Pxy*, но ее применение ограничено половиной адресного пространства РВВ (32 из 64). Изменение состояния сразу всех разрядов выходного порта производится командой *OUT PORTx, Rr*. Чтение входов производится командой *IN Rd, PINx* для всех восьми разрядов. Для поразрядного чтения нет отдельной команды, но есть пара уже упоминавшихся команд, осуществляющих пропуск следующей команды, если в

указанном РВВ указанный бит находится в установленном (1) или сброшенном (0) состоянии – *SBIS* или *SBIC PINx, PINxy*.

Табл. 1.3. Примеры операций с портами на Ассемблере и Си

Настроить выводы порта A PA3…PA0 на ввод с притяжкой (к VCC)	
`ldi R16, (1<<3)|(1<<2)|(1<<1)|(1<<0);` `out PORTA, R16`	`PORTA = (1<<3)|(1<<2)|(1<<1)|(1<<0);`
Настроить PB0, PB1 на вывод, задать PB0=1, PB1=0	

`ldi R16, (1<<1)|(1<<0)`
`out DDRB, R16`
`sbi PORTB, 0`
	`DDRB = (1<<1)|(1<<0);`
	`PORTB = (1<<0);`

Чтение входов PA3…PA0	
`in R16, PINA` `ANDI R16, 0x0f`	`in = PINA & 0x0f; //=0b00001111;` `//0x0f = (1<<3)|(1<<2)|(1<<1)|(1<<0)`
Обнулить выход PB0	

`sbi PORTB, 0; Если к рег.нет доступа:`
`in R16, PORTB`
`sbr R16`
`out PORTB, R16`
	`PORTB &= ~(1<<0);`

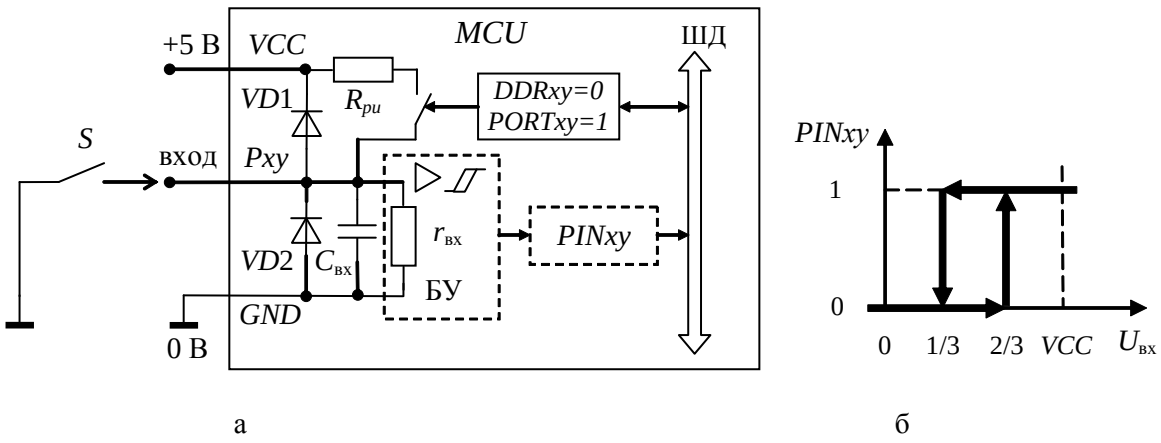
Инверсия выхода PB1	

`in R16, PORTB`
`ldi R17, 1<<1`
`eor R16, R17`
`out PORTB, R16`
	`PORTB ^= (1<<1);//Исключ.ИЛИ с присв.`

Если вход PA3 = 1, установить выход PB0	
`sbic PINA, 3` `sbi PORTB, 0`	`if(PINA | (1<<3)) PORTB |= (1<<0);`

Схемотехника портов подробно представлена в технических описаниях *[datasheets]*. В структуре порта можно выделить блок ввода, блок вывода и блоки обслуживания альтернативных функций данного вывода *[pin]*.

Схема порта в режиме ввода представлена на рис. 1.5, а. Она состоит из входного буферного усилителя (БУ) по схеме триггера Шмитта, стробируемого при чтении соответствующего разряда *y* регистра *PINx*, защитных диодов *VD1*, *VD2*. Передаточная функция триггер Шмитта $PINx.y = f(U_{вх})$ (рис. 1.5, б) имеет гистерезис, что исключает возможность попадания в состояние с "запрещенным" уровнем входного напряжения (0.3…0.6)*Vcc*.



а б

Рис. 1.5. Порт в режиме ввода

Защитные диоды ограничивают напряжение на входе усилителя на уровне *Vcc* + 0.6 В (*VD1*) и -0.6 В (*VD2*), но рабочий ток этих диодов не должен превышать 1..3 мА. Это определяет номинал резистора, включаемого последовательно на входе порта при вероятном выходе напряжения источника сигнала за допустимые пределы.

Если между входом Pxy и нулем (GND) подключен знакопеременный источник напряжения e(t), положительная амплитуда которого E+ > VCC + 0.6 В или положительная амплитуда которого E- < VCC + 0.6 В, то последовательно с источником следует поставить резистор сопротивление котрого удовлетворяет условиям: ро ро

R ≥ (E+ – VCC – 0.6) / 1 мА, R ≥ (E- – 0.6) / 1 мА.

На схеме также показано сопротивление $r_{вх}$, моделирующее входной импеданс усилителя. Его приблизительное значение может быть оценено по паспортным данным $r_{вх} = U_{вх} / I_{вх}$ = 5 В / 8 мкА = 0.625 МОм.

Резистор *Rpu* (*[pull up]* - то есть подтягивающий к *Vcc*) подключается к выводу входа при *DDRxy* = 0 *PORTxy* = 1, его сопротивление 20…50 кОм, он задает ВЫСОКИЙ уровень свободного входа (то есть, не подключенного к определенному потенциалу). Такая настройка удобна, когда ко входу МК подключен однополюсный ключ *S*, второй вывод ключа занулен. В замкнутом состоянии ключ обеспечивает потенциал равный *GND*, в разомкнутом без резистора R_{pu} потенциал не определен, что может привести к чтению «шума» – произвольной последовательности «нулей» и «единиц».

Схема порта в режиме вывода (выхода) представлена на рис. 1.6, а. Выходной каскад порта представляет собой инвертор и состоит из комплиментарной пары МОП транзисторов *p*-канального *VT1* и *n*-канального *VT2,* затворы которых соединены и управляются регистром *PORTxy*.

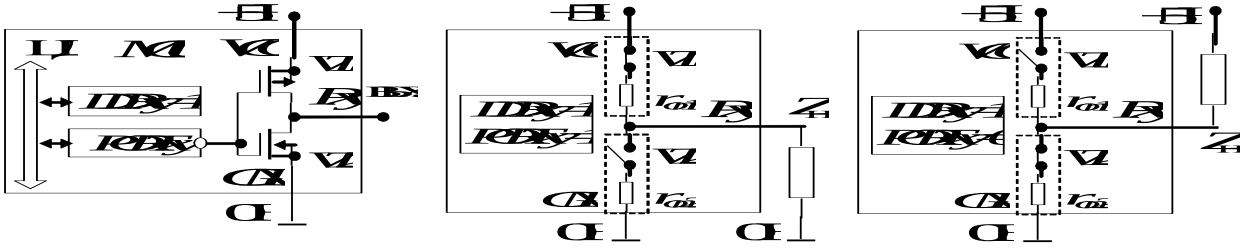
Когда регистр *PORTxy* = 1 (рис. 1.6, б), верхний транзистор *VT1* замкнут, а нижний *VT2* – разомкнут. Если при этом подключена нагрузка $Z_н$ между выходом *Pxy* и нулем питания, то через нее течет ток равный $I_{OH} = VCC / (Z_н + r_{on1})$, здесь r_{on1} – сопротивление транзистора *VT1* в открытом состоянии. Например, по паспортным данным при токе выхода 20 мА VCC = 5 В напряжение на выходе V_{OH} = 4.2 В, то есть сопротивление r_{on1} = (5 – 4.2) / 0.02 = 40 Ом.

Когда регистр *PORTxy* = 0 (рис. 1.6, в), верхний транзистор *VT1* разомкнут, а нижний *VT2* – замкнут. Если при этом подключена нагрузка $Z_н$ между выходом *Pxy* и плюсом питания контроллера, то через нее течет ток равный $I_{OH} = VCC / (Z_н + r_{on2})$, здесь r_{on2} – сопротивление транзистора *VT2* в открытом состоянии. Здесь сопротивление r_{on2} = 0.6 / 0.02 = 30 Ом.

Напоминаем, что внешняя нагрузка выхода (то есть ее сопротивление в любой момент времени) должна ограничивать ток (предельное значение в любом направлении I_{max} = 40 мА), иначе возможен выход данного вывода (ножки) из строя. Значение сопротивления нагрузки должно быть заведомо больше $R_{н\_мин}$ = VCC / I_{max} – r_{on_min} = 5 / 0.04 – 30 = 95 Ом. По этой же причине не рекомендуется подключать непосредственно к выходу конденсатор или другую нагрузку, имеющую значительную емкость. Лучше их подключать через резистор.

Чтение регистра *PINxy* в режиме вывода при сравнительно высокоомной нагрузке будет давать значения, совпадающие со значениями, записанными в регистр *PORTxy*. При подключении низкоомной нагрузки ВЫСОКИЙ уровень становится ниже *Vcc*, а НИЗКИЙ

- выше GND из-за сопротивления транзисторов выходного каскада.



а б в

Рис. 1.6. Порт в режиме вывода

Расчет сопротивления в цепи подключения светодиода.

Вариант с подключением светодиода *VD* и токоограничивающего резистора *R* между выходом порта и *GND* (см. схему Примера 1-1).

$VCC = I_{OH} (R + r_{on1}) + U_{VD}$, здесь U_{VD} – падение напряжения на светодиоде. Для расчета сопротивления верхнего транзистора используем паспортные данные МК $r_{on1} = (VCC - V_{OH}) / I_{OH}$ = (5 – 4.2) / 3 мА = 267 Ом. Полагая ток светодиода 10 мА, падение напряжения (по паспортным данным конкретного светодиода 1.5 В) можем определить сопротивление резистора $R = (VCC - U_{VD}) / I_{OH} - r_{on1}$ = (5 – 1.5) / 0.01 – 267 = 350 – 267 = 83 Ом.

Алгоритм программной реализации управления

Программный опрос. Типовые программные звенья.

Структура программного обеспечения при чисто программной реализации задач управления показана на рис. 1.6. Она состоит из *инициализации* – группы действий по настройке устройств ввода/вывода, переменных и пр., выполняемых однократно после рестарта МК и *основного цикла* – бесконечно повторяемой группы действий. Нередко действия в основном цикле называют поллингом.

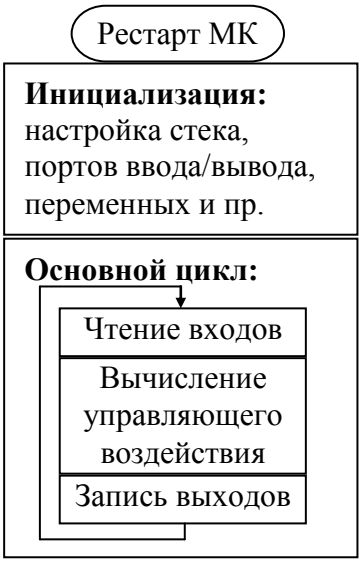


Рис. .1.7

Основной цикл состоит чтения состояния входов (и внутренних переменных), вычисления управляющего воздействия и его вывода на выходы. Это основной способ программной реализации задач управления. $Y = f(X)$, где $X = \{x_0, x_1 \dots x_n\}$ – одно- или многоразрядные входы, $Y = \{y_0, y_1 \dots y_m\}$ – одно- или многоразрядные выходы, в общем случае $n \neq m$.

Периодичность основного цикла $T_П$ определяет минимальное время реакции (программируемого) устройства, складывается из времени выполнения команд, составляющих тело основного цикла: $T_П = \sum_{i=1}^{K} Ti$, где Ti – время выполнения *i*-ой машинной команды, К – число команд основного цикла.

Количество машинных команд зависит от многих факторов, объективные:

1) количество обслуживаемых входов и выходов;
2) сложность алгоритма программной обработки;
3) система команд и разрядность МК.

К субъективным факторам можно отнести «уровень» и стиль программиста, выбор средств разработки программ и пр.

Использование команд условных ветвлений приводит к варьированию времени основного цикла. Период основного цикла может быть постоянным только для очень простых программ, имеющих минимум ветвлений. Чем сложнее логика программы, тем более варьируется длительность основного цикла. В большинстве случаев имеют в виду некое среднее или максимальное значение, полученное по результатам тестирования программы в различных условиях.

Нередко периферийные устройства имеют бит-флаг готовности в регистре состояния, например, флаг завершения записи в *EEPROM* или флаг завершения преобразования АЦП. В этом случае очередную операцию ввода или вывода не следует начинать до установления флага. Здесь возможно два варианта действий.

Один вариант – организовать цикл ожидания готовности. Это может привести к росту времени выполнения основного цикла, а в некоторых случаях – к зависанию. Для борьбы с возможными зависаниями используют сторожевой таймер.

Второй вариант – выполнять проверку однократно, если флаг не установлен – пропустить выполнение на этом цикле. В этом случае минимизируются потери времени на задержки ожидания, но задержка выполнения действий по флагу может достигать длительности основного цикла.

Нередко в структуре основного цикла возникают подциклы с существенно разной, как правило, кратной между собой периодичностью исполнения. Это характерно, прежде всего при программной генерации периодических сигналов с варьируемыми временными параметрами – периодом, задержкой, фазой (будут рассмотрены ниже).

Не редко требования к времени реакции различных задач основного цикла сильно различается. Например, обновление изображения индикаторов должно выполняться не чаще 10 раз в секунду. Измерение медленно меняющихся параметров, например, температуры массивных объектов целесообразно выполнять с шагом по времени от долей секунды до десятков секунд. Задачи управления приводом перемещения зависят от требуемой динамики и степени детализации этапов перемещения. Требуемое время реакции меняется в очень широких пределах – от десятков миллисекунд до десятков микросекунд.

В этих случаях целесообразно группировать задачи по требуемому времени реакции и учитывать их суммарное время выполнения. Задачи с самым малым временем реакции выполняются с макимально возможной частотой, периодически их выполнение приостанавливается для выполнения группы более медленных задач, а еще реже – для еще более медленных… Удобно, когда время выполнения каждой задачи в группе много меньше периода ее выполнения. Иначе приходится продумывать механизм разбиения каждой задачи на кванты времени…

Фактически, распределение времени процессора между задачами называется диспетчеризацией. При значительном количестве и переменном составе задач лучше всего использовать для этого операционную систему, но в нашем курсе мы этот подход рассматривать не будем.

Типовые программные звенья:

1) *логические*: связывают состояния отдельных (одноразрядных) входов, одноразрядных переменных (флагов) или их сочетаний с требуемым состоянием выходов, используют команды поразрядной логической обработки, команды работы с битами, условные ветвления, табличные вычисления;

2) *арифметические*: связывают численные значения многоразрядных входов с численными значениями многразрядных выходов; используют арифметические команды, табличные вычисления;

3) их комбинации, например, компарирование – сравнение двух чисел и изменение состояния бита поо результату сравнения.

Логические подразделяются:

1) комбинационная логика – состояние выходов определяется только состоянием входов на текущем (i-м) временном шаге $Y_i = f(X_i)$;

2) последовательностная логика – состояние выходов определяется не только состоянием входов на текущем (i-ом) временном шаге, но и предыдущими (i-1, i-2, i-3…) состояниями $Y_i = f(X_i, Y_{i-1}, X_{i-1}, \ldots)$, что требует битовой памяти; например, выявление фронта или среза сигнала на цифровом входе, подавление механического дребезга контакта и т.д.

Арифметические подразделяются:

1) Функциональное преобразование $Y_i = f(X_i)$, например, сигналов с датчиков; для линейного преобразование обычно используются операции сложения/вычитания и умножения/деления в целочисленном формате, реже – в формате с плавающей запятой; для нелинейного преобразования используется кусочно-линейная аппроксимация и табличное преобразование.

2) Вычисления с памятью $Y_i = f(X_i, Y_{i-1}, X_{i-1}, \ldots)$, что требует создания и обслуживания различных буферов разрядностью от одного до четырех байт, например, программная фильтрация, программный регулятор и т.д.

Важно подчеркнуть, что производительность арифметических вычислений в первую очередь зависит от разрядности процессора, а проиводительность логических вычислений больше зависит от состава логических команд и способов адресации. В задачах управления преобладают логические задачи, что объясняет широкое распространение 8-разрядных МК в «эпоху» дешевых 32-разрядных МК.

Часть 2. Процесс проектирования устройств на МК

2.1. Этапы процесса проектирования устройств на МК

Процесс проектирования микропроцессорного устройства, точнее, устройства с применением встроенного МК или МП во многом похож на процесс проектирования любого другого электронного устройства. Отличия заключаются в том, что кроме разработки схемы электрической и конструкции – печатной платы, корпуса и пр., необходимо параллельно разрабатывать управляющую программу.

1. Разработка технического задания (ТЗ) выполняется разработчиком по требованиям пользователя и согласовывается с последним. Описывается назначение устройства, перечень функций, основные технические характеристики, связи и взаимодействие устройства с другими элементами системы управления или автоматизируемого комплекса, при необходимости – взаимодействие с оператором (человеко-машинный интерфейс), требования к конструкции.

От качества выполнения ТЗ во многом зависит результат. Степень детализации ТЗ зависит как от сложности проекта, так и от уровня взаимопонимания заказчика и исполнителя. Разработка новых для заказчика и исполнителя проектов требует для успешной реализации по меньшей мере двух-трех итераций процесса разработки, включая изготовление и испытания.

2. Разработка алгоритма работы устройства – слабо формализуемый этап. Состоит в одновременном выборе способа решения (организация циклической последовательности действий) и средств решения (выбор МК и других основных электронных компонент).

Как и в любом другом процессе проектирования, важно соблюдать баланс между принципами проектирования «сверху – вниз» и «снизу – верх». Первый требует понимания принципов системного подхода, а второй – знания конкретных приемов реализации простейших алгоритмов (последнее и есть основная задача этого курса).

Как правило, заканчивается разработкой структурных схем аппаратной и программной части.

Степень детализации в этих схемах зависит как от выбора средств проектирования, так и от уровня подготовленности разработчиков. Например, использование в качестве языка программирования Ассемблера требует использования языка блок-схем с проработкой алгоритма программы почти до уровня машинных команд. При использовании языка высокого уровня (типа C/C++) структурно-модульные средства последнего в сочетании с «хорошим стилем программирования» снижают требования к уровню детализации алгоритма на языке блок-схем. Однако совместное использование программных и аппаратных решений требуют поиска новых удобных и компактных средств представления алгоритма.

Одним из важнейших критериев в выборе формы представления алгоритма мне представляется возможность и удобство проверки логической непротиворечивости алгоритма.

Далее процесс проектирования формально может идти параллельно – разработка программной и аппаратной частей, до начала их совместной отладки.

3.1. Разработка программной части.

3.1.1. Написание текста программы на выбранном языке (ассемблер/C/C++) –

необходим текстовый редактор. Здесь весьма важен, так называемый, «хороший стиль программирования», базирующися на хорошем знании языковых средств, архитектуры МК и достаточном опыте программиста.

3.1.2. Трансляция исходного текста и сборка загрузочного модуля. При работе с проектом из нескольких файлов исходного текста и использовании библиотек готовых подпрограмм требуется объединение объектных модулей в один загрузочный. Необходимы программы транслятора (компилятора) и редактора связей, обычно работающие в связке под управлением программной оболочки среды программирования.

В англоязычных меню обычно весь процесс трансляции называется Build (построение или сборка), для указания выполнения только компиляции отдельного текстового файла используется термин Compile (компиляция). Нет специальной команды вызова редактора связей, его работа инициируется по команде Build (или ее разновидностям) при наличии полного набора объектных модулей.

На этом этапе выявляются синтаксические ошибки *[Error]* и предупреждения *[Warning]*. Последние не препятствуют созданию загрузочного модуля, но желательно добиваться их отсутствия – это способствует правильному стилю программирования и уменьшает вероятность логических ошибок.

Загрузочный модуль может иметь несколько форматов. Для загрузки в память программ целевого МК требуется только двоичный код команд и данных (констант). Для загрузки в моделирующую программу (симулятор) в дополнение к машинному коду требуется связь с исходным текстом для поддержки процесса отладки в символьном виде.

3.1.3. Отладка программы *[Debug]* заключается в выявлении вычислительных, логических ила алгоритмических ошибок в программе: корректной работы вычислительных функций, правильного использования адресов, последовательности действий, времени выполнения критических частей программы и пр.

До появления прототипа или макета целевой платы отладка может выполняться в программной модели МК – симуляторе. Простейшие симуляторы моделируют работу ядра – процессора, памяти программ и данных, более сложные моделируют работу периферийных устройств ввода/вывода с учетом масштаба времени, задаваемого значением тактовой частоты процессора.

Сравнительно недавно появился новый класс симуляторов, включающих кроме логической модели МК схемотехнические модели всех узлов ввода/вывода и наборы моделей типовых электронных и др. компонент. Как правило, такие пакеты строятся на базе широко распространенного математического пакета моделирования цифровых и аналоговых устройств *PSPICE* (например, *Proteus VSM*). Такие симуляторы позволяют проверить работоспособность не только программной, но и аппаратной частей.

Рассмотрим перечисленные выше этапы подробнее, чтобы обсудить выбор готовых аппаратных и программных средств для их исполнения.

Для наглядности рассмотрим и практическую реализацию этих вопросов на очень простом «сквозном» примере «Пульта управления». Он будет включать ТЗ, разработку алгоритма, выбор элеметов схемы, тексты программ на ассемблере и разновидностях языка С, рассмотрение процедур трансляции и отладки. Все готовые элементы примера находятся в папке Primer \ p1-1 OldLab \ Lab1-2012, разбитой на подкаталоги, их назначение рассмотрено ниже.

2.2.Техническое задание и разработка алгоритма (блок-схемы)

Техническое задание (ТЗ) представляет собой набор требований к разрабатываемому устройству или программе. Как правило, ТЗ разрабатывает исполнитель работы и согласовывает его с заказчиком. Исходным материалом для составления ТЗ выступают потребности заказчика и опыт исполнителя.

Потребности заказчика выражаются чаще в виде перечня выполненяемых функций и их кратких количественных характеристиках, нередко количественные характеристики подменяются качественными (сделайте мне «хорошо»).

Исполнитель заинтересован в более детальном описании разрабатываемого/ поставляемого устройства. Отталкиваясь от своих знаний и опыта, набора готовых решений собственной разработки или имеющихся в даннм секторе рынка, он должен определить объем своих работ, сроки и стоимость. При необходимости разработки он должен в этот момент уже представлять себе алогритм работы, чтобы можно было оценить его предельные характеристики.

Сотав технических характеристик устройств на МК имеет много общего с другими техническими, в особенности с электронными устройствами:

- назначение изделия и выполняемые им функции,
- связи с другими техническими устройствами (аппаратный и аппаратно-программный интерфейс),
- человеко-машинный интерфейс,
- электропитание,
- условия эксплуатации,
- степень защиты,
- массо-габаритные характеристики и т. д.

В процессе обучения отсутствие опыта не позволяет студенту самому разрабатывать ТЗ, поэтому очень важно, чтобы в процессе выполнения учебных заданий обучаемый понял взаимосвязь параметров задания и предлагаемого способа его решения, то есть алгоритма.

Алгоритмизация – процесс разбиения задачи на элементарные операции или звенья. Сам процесс плохо формализуем, конечный результат во многом зависит от выбора средств и опыта разработчика…

Наиболее универсальным средством выражения мысли человека является текст на родном языке. Поэтому общие идеи любого алгоритма выражаются текстом и дополняются расчетами, таблицами, диаграммами… Полное описание алгоритма в таком виде страдает излишним объемом при недостаточной формализованности. Сложно проверить его логику, временные характеристики.

Попытки решить эту проблему выражаются в появлении все новых средств представления алгоритма. Раньше это были в основном текстовые языки, теперь преобладает визуальное представление в виде схем, лестничных диаграмм или направленных графов…

Применительно к устройствам на базе встраиваемых МК требуется, по крайней мере. разработать печатную плату и программу для прошивки памяти МК.

Конечным видом проектной документации для производства печатной платы

являются сборочный чертеж платы с компонентами (микросхемы, резисторы, конденсаторы, разъемы и пр.), схема электрическая принципиальная, чертежи слоев печатной платы с переходными отверстиями. Предшествующим документом обычно является структурная электрическая схема. Она в упрощенном виде показывает внутренние и внешние связи элементов разрабатываемого устройства.

Конечным видом проектной документации по программе для прошивки МК являются тексты программ на выбранном языке программирования, предшествующим – обычно блок-схема, хотя программы на языках высокого уровня считаются «самодокументируемыми» (при условии выполнения правил хорошего стиля и достаточном объеме комментариев).

Спецификой применения МК является широкое использование его встроенных периферийных узлов: таймеров/счетчиков, АЦП, контроллеров последовательного интерфейса и пр. Практически любая функция, выполняемая устройством на базе МК, задействует аппаратные узлы вне МК, периферийные узлы МК и программу. Перечисленные выше документы отображают лишь отдельные фрагменты этой мозаики.

Функциональная схема показывает взаимосвязь всех этих элементов при выполнении заданных функций. Она может быть документом, предшествующим разделению процесса проектирования на аппаратную и программную части, а также существенно помогает в процессах настройки, наладки и ремонта готового устройства на базе МК. Широкое применение функциональных схем сдерживается отсутсвием строгого описания и готовых программ-редакторов.

При разработке программ широкое распространение получило представления алгоритма в виде *блок-схем* (ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем). Основные элементы «языка» блок-схем представлены на рис. 2.1:

– блоки начала (входа) и конца (выхода) программы или подпрограммы;
– блок элементарных действий (которому соответствует одна или несколько машинных команд либо операторов ЯВУ);
– блок сложных действий или подпрограмма (подразумевает «расшифровку» в виде отдельной блок-схемы или пояснительного текста);
– условный блок с одним входом и минимум двумя выходами, возле которых написаны альтернативные условия;
– комментарий – для более подробного описания блока или группы блоков;
– линии передачи управления между блоками, стрелки на которых можно опускать при движении вниз и вправо. Для блоков, расположенных последовательно сверху вниз прорисовка вертикальных связывающих линий не обязательна.

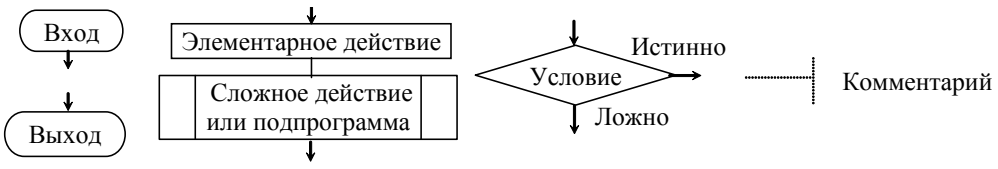


Рис. 2.1

Пример 1. Пульт управления

Задание: Пульт управления должен состоять из двух кнопок и одного индикатора, нажатие кнопки «Вкл» должно подавать питание на «нагрузку» и индикатор, нажатие

кнопки «Выкл» – снимать напряжние с «нагрузки» и индикатора.

Разработка алгоритма

Для подключения кнопок нам понадобятся два дискретных входа МК, для управления «нагрузкой» и индикатором – один дискретный выход, итого – три вывода порта. С учетом выводов питания (VCC, GND) и входа сброса (RESET), выбора встроенного тактирования (пусть будет 1 МГц) нам было бы достаточно МК с 6 ножками, но в семействе AVR минимальное число ног 8. Для наших целей подходит даже самый простой МК из подсемейства Classic AT90S2323.

Для столь простой задачи можно сразу привести схему электрическую (рис. 2.2).

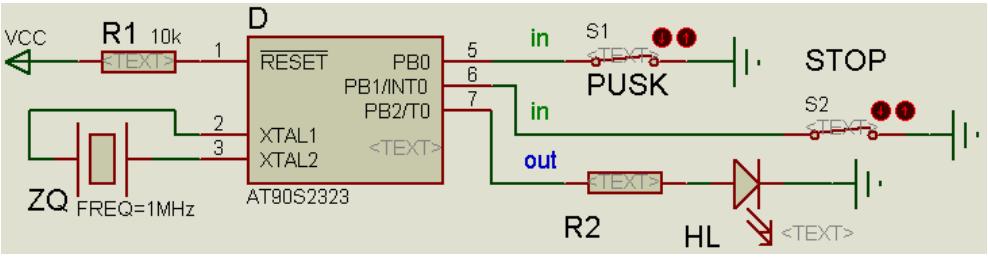


Рис. 2.2

Здесь мы видим МК D (AT90S2323), к выводам порта подключены кнопки S1, S2 (входы PB0, PB1), нагрузка R2 и индикатор – светодиод HL (к выходу PB2). Ко входу RESET подключен подтягивающий резистор R1, к выводам XTAL1, 2 – кварцевый резонатор ZQ для генерации тактовой частоты 1 МГц. Выводы питания МК (VCC, GND), как и сам источник питания не показаны, подразумевается, что «треугольник» VCC подключен к +5 В, а ноль – к значку заземления.

Выбор типа светодиода и значения сопротивления резистора R2 мы рассмотрим позже (п.2.6). Сейчас нам важно понять, что питание нагрузки и индикатора выполняется от выхода PB3 в ВЫСОКОМ состоянии.

Кнопки S1, S2 подключены между выходами и нулем питания, в замкнутом состоянии это обеспечивает НИЗКИЙ уровень на соотвествующем входе. В разомкнутом состоянии кнопки не задают уровень, поэтому необходимо использовать встроенный в каждый порт резистор, подтягивающий потенциал входа к ВЫСОКОМУ уровню. Реальные кнопки по конструкции могут быть нормально разомкнутыми, при нажатии таких кнопок они замыкают контакты. В нашей схеме предполагается использование номально замкнутых кнопок, поэтому в исходном состоянии на обоих входах НИЗКИЙ уровень, штатно нажатие должно приводить к размыканию кнопки и появлению ВЫСОКОГО уровня. Одновременное нажатие кнопок считается не штатной ситуацией.

На рис. 2.3 представлен алгоритм программы управления. Все действия сводятся к операциям с содержимым регистров порта, точнее с битами этих регистров, что характерно для простых программ управления. Здесь исползуется синтаксис структур из языка С, то есть PINB.0 – это нулевой бит 8-разрядного регистра PINB.

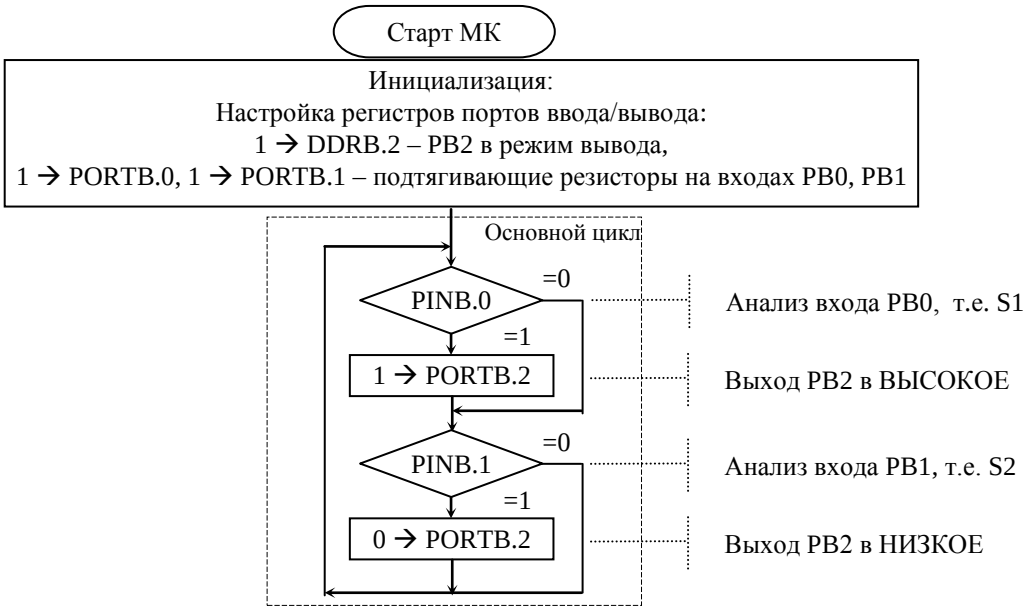


Рис. 2.3

Следует отметить, что это не единственный вариант алгоритма даже для столь простой задачи. Недостатком алгоритма является, то, что при одновременном нажатии кнопок S1 и S2 (то есть при PINB.0 == 0 и PINB.1 == 0) выход PB2 будет переключаться с периодом, равным периоду основного цикла. Этого можно избежать, если проверять нажатие двух кнопок одновременно, но это уже несколько другой алгоритм.

2.3. Языки программирования и синтаксическая проверка проекта

2.3.1 Язык программирования Ассемблер

Ассемблером называется машинно-зависимый язык, каждой мнемонической команде которого соответствует одна единственная двоичная команда. Для трансляции исходного текста в машинный (или объектный) код используется программа, называемая ассемблером.

Текст этого раздела ориентирован на Ассемблер-1 для МК семейства *AVR* в среде *AVR_Studio*, нг общие принципы едины для всех разновидностей ассемблеров любых процессоров.

Запись программы выполняется построчно. Строка может содержать до 120 символов (буквы, цифры, знаки, пробелы). Строчные и прописные символы не различаются.

Строка может быть пустой. В строке выделяются четыре поля, разделенные пробелами или табуляторами:

[Метка:]	Команда/.Директива	[Операнд/ы]	[;Комментарий]
[Label:]	*Instruction/.Directive*	*[Operands]*	*[;Comment]*

Основным элементом строки является мнемокод команды или директивы ассемблера. Одна команда порождает одну машинную инструкцию (код). Директивы указывают транслятору, как вести процесс трансляции и не порождают машинных кодов.

Следом за командой или директивой могут следовать операнды – данные или адрес. Команды/директивы определяют число операндов, для команд – от 0 до 2, для директив число варьируется более широко. Операнды разделяются запятыми, допустимо добавлять пробелы или табуляторы. Численные значения операндов для удобства восприятия часто

заменяют символическими именами, замена объявляется соответствующей директивой. В операндах команды первый является приемником результата, второй – источником.

Далее находится поле для записи комментария – свободного пояснения на любом языке. Запись комментария начинается со знака «;». Поле комментария может быть пустым.

Слева находится поле для записи метки. Оно может быть пустым, используется для адресации команды или ячейки памяти. Метка представляет собой последовательность из букв, цифр и некоторых знаков и начинается с буквы. Запись метки заканчивается знаком «:». Если текст метки достаточно объемный, адресуемую команду или директиву можно перенести на следующую строку.

В процессе ассемблирования мнемокоды команд преобразуются в машинные коды команд, директивы ассемблера и метки используются при формировании программы в машинных кодах. Комментарии игнорируются. Они используются программистами в качестве пояснений к программе.

В результате работы программы-транслятора (ассемблера) создается объектный или загрузочный файл – образ выполняемой программы в машинных кодах, а также файл листинга, содержащий кроме исходного текста колонки адресов и машинных кодов в шестнадцатеричном формате.

Директивы ассемблера выполняют следующие функции (подробно в Приложении):

- определяют тип сегмента памяти: директивы *.CSEG* (команды), *.ESEG (EEPROM)*, *.DSEG* (данные);
- распределяют память микроконтроллера: директивы *.ORG* (адрес в текущем сегменте), *.DB* (константа- байт), *.DW* (константа- слово), *.BYTE* (переменная в ОЗУ или EEPROM);
- присваивают имена и значения символическим именам: директивы *.DEF* (РОН), *.EQU* (константа), *.SET* (константа изменяемая в процессе трансляции);
- управляют процессом ассемблирования (то есть сборки): директивы *.DEVICE* (тип МК), *.INCLUDE* (подключение текстового файла), *.EXIT* (конец трансляции), *.MACRO* (начало макрокоманды), *.ENDMACRO* (конец макрокоманды);
- управляют формированием листинга (директивы *.NOLIST, .LIST, .LISTMAC*).

Краткое рассмотрение команд и директив находится в Приложениях 3 и 4, подробное можно найти в *Help*`е *AVR Studio*.

Назначение и основные компоненты среды *AVR Studio*

IDE AVR_Studio – интегрированная среда разработки *[IDE – Integrated Development Environment]* приложений для микроконтроллеров семейства *AVR* (*AT90S, ATmega, ATtiny*), свободно распространяемый продукт фирмы *Atmel [Atmel.com]*. Здесь и ниже используется версия 4.18 (2009 г.), следующие версии поддерживают семейства 32-разрядных МК и требуют значительно больших ресурсов компьютера при установке.

IDE AVR Studio содержит:

- текстовый редактор исходных текстов программ;
- транслятор языка ассемблера *(Atmel AVR macroassembler*);
- отладчик (*Debugger*);
- симулятор (*AVR Simulator*);
- загрузчик машинного кода в память МК с поддержкой внутрисхемного

программирования *(In-System Programming, ISP*);

- встроенную справочную систему (Help), содержащую описание языка ассемблер и системы команд МК серии AVR и др. компоненты.

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

Отладочная среда поддерживает выполнение программ, как в виде ассемблерного текста, так и в виде исходного текста языка *C/C++*. В последнем случае загружаемый модуль программы должен содержать не только машинный код, но и исходный текст программы, привязанный к машинному коду.

С отладчиком мы лучше познакомимся ниже, здесь нас интересует разработка программы на ассемблере.

При разработке и отладке программ удобно использовать встроенные средства поддержания проекта. Понятие «проект» в интегрированных средах разработки включает: перечень файлов, содержащих исходные модули, пути размещения этих и создаваемых в процессе трансляции и «линкования» файлов, параметры настройки процессов создания и отладки программы и др. Удобство заключается в том, что вы однократно осуществляете настройку проекта, сохраняете в поименованном файле <*name.aps*>, а потом используете эти настройки, «открывая» файл проекта.

Пример 1 (продолжение).

Разработка программы Примера 1 на языке ассемблера

Ниже представлен текст на ассемблере, реализующий алгоритм по рис. 2.3.

```
.include <2323def.inc>  ;Подключение файла описания символьных имен МК
.cseg                   ;сегмент памяти программ, не обязательная строка
.org 0                  ;Нулевой адрес текущего сегмента, начало программы
      rjmp START        ;Переход на метку START для обхода векторов преывания
      reti              ;External Interrupt 0
      reti              ;Timer/Counter0 Overflow
.org INT_VECTORS_SIZE   ;Резервирование места под область векторов прерывания
START:                  ;
      ldi   r16, 0b0100 ;Загрузить число 0b0100 в РОН r16
      out   DDRB, r16   ;Вывод содержимого РОН r16 в РСФ (PBB) DDRB
      ldi   r16, 0b0011 ;Загрузить число 0b0011 в РОН r16
      out   PORTB, r16  ;Вывод содержимого РОН r16 в РСВ (PBB) PORTB
LOOP:                   ;Начало основного цикла
      sbic  PINB, 0     ;Если бит 0 регистра PINB равен нулю, пропустить
                        ; следующую команду
      sbi   PORTB, 2    ;Установить (в 1) бит 2 регистра PORTB
      sbic  PINB, 1     ;Если бит 1 регистра PINB равен нулю, пропустить
                        ; следующую команду
      cbi   PORTB, 2    ;Обнулить бит 2 регистра PORTB
      rjmp  LOOP        ;Переход к метке LOOP, т.е. к началу цикла
```

Текст в комментариях нацелен на понимание команд и директив. Для человека с хорошим знанием ассемблера и архитектуры данного МК важнее было бы комментировать выполнение алгоритма.

Для записи численных констант в командах ldi использован двоичный формат. Тождественным было бы использование и других форматов:

0b0011 = 0x03 = 3 = (1<<1) | (1<<0) = (1<<1) + (1<<0).

Все операнды в командах и директивах являются числами, их символьные значения закодированы либо в файле описания конкретного МК (первая строка текста, файл из пакета AVR-Studio, путь .. \ Atmel \ AVR_Tools \ AvrAssembler \ Appnotes), либо непосредственно в тексте программы (метки START и LOOP). По умолчанию текст программы «привязан» к сегменту памяти программ (.cseg) начиная с ячейки 0 (.org 0).

Первой исполняемой командой является jmp START, она всегда размещается в нулевой ячейке ПП.

Две следующие строки с командами reti (возврат из прерывания) являются «заглушками» не используемых в нашем алгоритме прерываний. Смысл их станет поятнее после рассмотрения системы преываний (п. 3). Формально первые три команды можно было бы не использовать, но лучше привыкать к надежной структуре программы сразу.

До метки LOOP идет этап инициализации, далее – основной цикл.

Для записи «единиц» в заданные биты регистров порта использованы команды, оперирующие с байтами – загрузка константы в РОН и копирование содержимого в РСФ. Четыре команды требуют 4 ячейки ПП и выполняются за 4 машинных цикла. Альтернативой было бы использование команд изменения одного бита РСФ:

```
sbi DDRB, 2  ; Запись 1 в бит 2 РСФ DDRB
sbi PORTB, 0 ; Запись 1 в бит 0 РСФ PORTB
sbi PORTB, 1 ; Запись 1 в бит 1 РСФ PORTB
```

Этот вариант требует 3 ячейки ПП и выполняется за 6 машинных циклов. Изменение состояния двух и более бит оптимальнее делать байтовыми командами. Также следует учитывать ограничения в адресации РСФ – только 32 из 64 основных доступны в битовых командах.

Для выполнения простейшего действия по условию состояния одного бита РСФ удобно использовать команду sbic/s. Данная команда образует ветвление для выполнения одной команды без формирования адреса (метки). Если бы требовались разные или сложные действия, пришлось бы формировать отдельные ветви программы с использованием команд типа jmp.

Действия в среде AVR Studio по созданию загрузочного кода

На рис. 2.4 показан вид экрана среды AVR-Studio после компиляции проекта Lab1.aps на ассемблере.

Вверху находятся привычные строки Меню и пиктограмм среды.

Центральное окно занято редактором текста, в данном случае это файл на языке ассемблера lab1.asm. настройки цветового выделения: синий – мнемоники команд, зеленый – комментарии. Внизу окна видны закладки текущего и ране открытых в окне редактора файлов.

Окно слева (Project) содержит дерево проекта:

в папке Soutce Files – текстовый файл с исходным текстом программы Lab1.asm,

в папке Included Files – текстовый файл символьных имен выбранного МК 2323def.inc,

в папке Output – двоичный файл кода для загрузки в ПП Lab1.hex и служебный текстовый файл карты памяти Lab1.map,

в папке Object Files – двочный объектный файл Lab1.obj (результат работы компилятора, используется компоновщиком при создании выходных файлов).

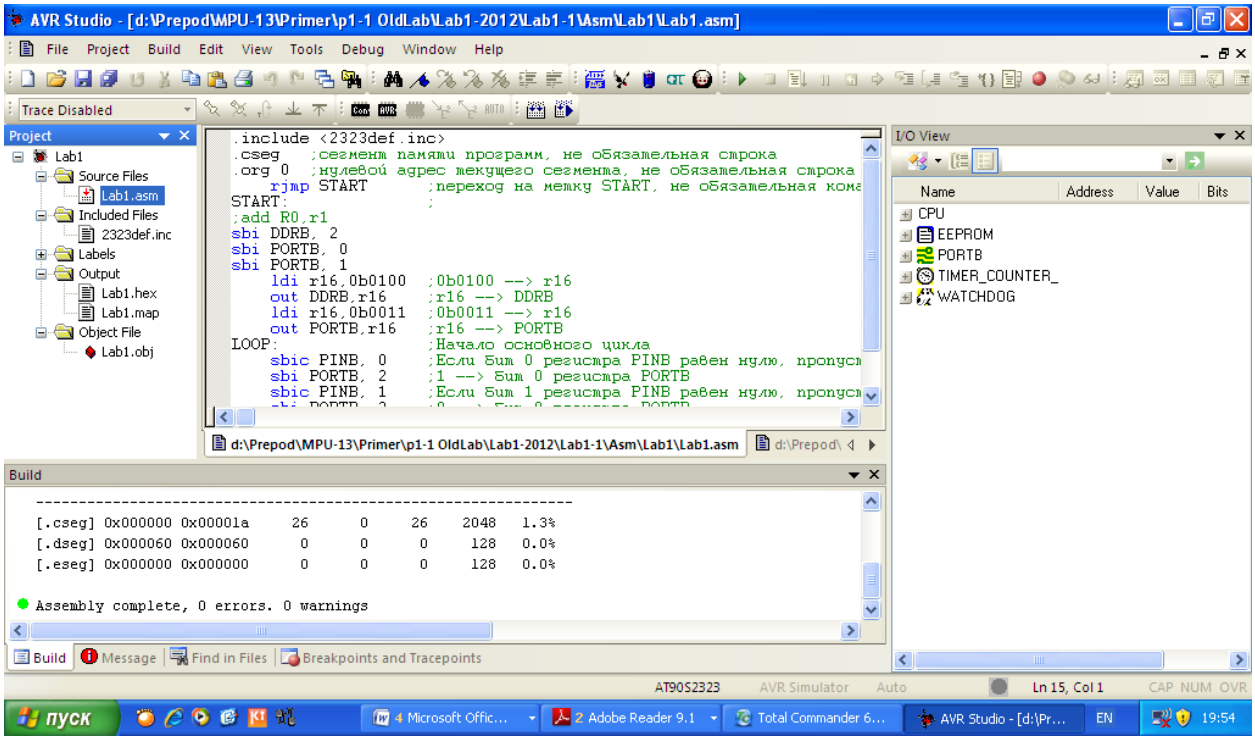


Рис. 2.4

Внизу находится окно вывода сообщений, в данном случае активна закладка Build (работа компоновщика), наболее важное последнее сообщение «Assembly complete, 0 errors. 0 warnings» говорит об успешном завершении сборки (трансляции) без ошибок и предупреждений. Выше видна информиция по использованию ресурсов МК – числе ячеек памяти программ (Code) и памяти данных (Data) в абсолютных и относительных значениях (Use %, по отношению к имеющемуся в данном МК количеству ячеек Size). Эта информация позволяет количественно оценить относительную сложность и качество реализации алгоритма, а также оценить запасы по ресурсам МК.

Окно справа (I/O View) индицирует наличие и состояние регистров периферийных устройств (РСФ или РВВ), о нем подробнее будет рассказано в разделе «Отладка».

Для создания нового проекта используют меню Project → New Project или вызов мастера проектов Project Wizard (обычно появляется автоматически при запуске среды). После выбора пути и имени проекта, выбирается модель МК и средства отладки (по умолчанию – встроенный симулятор), подробнее смотрите в Приложении 2 к [МУ ЛР по МПУ]. После выполнения команды меню Save Project все параметры проекта сохраняются в файле *.aps (* - имя проекта), для повторного открытия достаточно «запустить» этот файл. Замечание: следует избегать использования кириллицы или других не латанских шрифтов в элементах пути и имени проекта.

Настройки транслятора с языка Ассемблер находится в меню Project → Assembler Options, по умолчанию выбрана версия AVR Assembler: Version 2, для знакомства с так называемым *листингом* включена опция Create List File.

Запуск процесса трансляции выполняется командой меню Build → Build или соотвествующей пиктограммой. При успешном завершении трансляции создается загрузочный модуль – двоичный файл Lab1.hex. содержащий машинный код

разработанной программы. При обнаружении ошибок [Errors] загрузочный модуль не создается, требуется исправление ошибок, ссылки на ошибки в окне Build выделены красными точками, клик по ним перемещает курсор в окне редактора на строку, вызвавшую ошибку. При обнаружении предупреждений [Warnings] загрузочный модуль создается, но лучше добиться отсутствия предупреждений; их спсок выделен желтыми точками, клик по ним также отражается в окне редактора.

Ниже приведена выдержка из текстового файла Lab1.lst (появляется в папке проекта Output после повторной трансляции). Здесь в первой колонке видим адрес в текущей области памяти (здесь Code, то есть ПП), во второй – машинный код (команды), все в шестнадцатирричном коде, далее – текст на ассемблере (комментарии убраны мной). Такой вид полезен при изучении системы команд МК и понимания расхода ячеек памяти.

```
                .cseg   ;сегмент памяти программ, не обязательная строка
                .org 0 ;нулевой адрес текущего сегмента,ельная строка
000000 c002      rjmp START        ;переход на метку START, первая…
000001 9518      reti              ;External Interrupt 0
000002 9518      reti              ;Timer/Counter0 Overflow
                .org INT_VECTORS_SIZE
                START:                  ;
000003 e004      ldi   r16,0b0100  ;0b0100 --> r16
000004 bb07      out   DDRB,r16    ;r16 --> DDRB
000005 e003      ldi   r16,0b0011  ;0b0011 --> r16
000006 bb08      out   PORTB,r16   ;r16 --> PORTB
                LOOP:                   ;Начало основного цикла
000007 99b0          sbic PINB, 0  ;Если бит 0 регистра PINB равен нулю, …
000008 9ac2          sbi PORTB, 2  ;1 --> бит 0 регистра PORTB
000009 99b1          sbic PINB, 1  ;Если бит 1 регистра PINB равен нулю, …
00000a 98c2          cbi PORTB, 2  ;0 --> бит 0 регистра PORTB
00000b cffb      rjmp  LOOP  ;Переход к метке LOOP, т.е. к началу цикла
```

Результаты компиляции – 12 строк программы требуют 24 байта ПП, ПД не используется.

2.3.2 Язык программирования C/C++

Семейство языков программирования C/C++ относится к текстовым языкам высокого уровня и широко применяется для программирования МК/МП для встроенных приложений. Эти языки относятся к языкам высокого уровня (ЯВУ), то есть формально один и тот же язык может использоваться для программирования МК/МП различных семейств. Для программирования конкретного семейства необходимо иметь соответствующий транслятор и знать особенности архитектуры семейства – адресацию конкретных областей памяти и регистров. Преимущества ЯВУ перед машинно-зависимым языком (ассемблером) – более высокая производительность труда программиста и гибкость в переносе алгоритмов с одного МК/МП на другой.

В реальности стандарты языков C/C++ не были ориентированы на особенности архитектуры МК/МП для встраиваемых применений, такие как разделение памяти на память программ и память данных, выделение регистров ввода/вывода, энергонезависимой памяти данных, наличие механизма прерываний и пр. Поэтому разработчики трансляторов и систем программирования вынуждены были добавлять свои решения для поддержки этих механизмов. Эти решения не тождественны по синтаксису у

различных разработчиков, что усложняет перенос алгоритмов с одного семейства МК/МП на другое.

Следует иметь в виду, что код, порождаемый транслятором ЯВУ всегда несколько проигрывает перед программой на ассемблере по времени выполнения и расходу памяти программ (при одинаково хорошем уровне программистов). Современные трансляторы ЯВУ имеют гибкую настройку оптимизации кода по времени выполнения или по объему кода, но использовать такую оптимизацию следует осторожно (возможна потеря некоторых действий, «лишних» с точки зрения транслятора). Эффективность трансляторов не является серьезным сдерживающим фактором, так как и объем памяти и производительность легко выбираются по параметрам конкретных МК/МП. Исключение составляет подгруппа цифровых сигнальных процессоров, для которых эффективность кода по быстродействию является одним из важнейших критериев, что заставляет использовать в ответственных частях алгоритмов вставки на ассемблере.

Ряд устаревших моделей МК AVR at90xxxx с малым числом выводов (от 6 до 8) не имеют ОЗУ (только РОН), что делает невозможным использование языка *C*.

При разработке архитектуры МК семейства *AVR*-8 учитывались особенности реализации трансляторов языка *C/C++*, что обеспечивает высокую эффективность кода. К таким трансляторам относятся *CodeVisionAVR, IAR, AVR_GCC* и др. Ниже мы кратко рассмотрим трансляторы, имеющие свободно распространяемые версии.

CodeVisionAVR

Среда *CodeVisionAVR* (*http://www/hpinfotech.com*) предназначена для разработки программ для МК серии *AVR* на языке *C* и ориентирована прежде всего на начинающих программистов. Основные элементы среды:

- текстовый редактор, дружественный к синтаксису языка С (цветовое выделение, выделение и свертка выражений в скобках и пр.),

- транслятор с языка *C* в машинный код *(*форматы *hex* и др.) и в код, предназначенный для символьной отладки *(cof)* в среде *AVR_Studio* либо в среде моделирования *Proteus VSM,*

- генератор исходного текста *CodeWizardAVR* обеспечивает удобство по инициализации регистров ввода/вывода, реализации функций прерывания и функций программной реализации некоторых последовательных интерфейсов и пр.,

- внутрисистемный программатор (ISP или JTAG), поддерживающий процессы загрузки/выгрузки кодов программы и данных в кристалл МК с использованием различных аппаратных загрузчиков (параллельный через LPT, последовательные через COM, USB и пр.),

- встроенный Терминал служит для аппаратной отладки устройств, поддерживающих канал последовательной связи *RS*-232 (*COM-port*),

- компактная, но достаточно полная система помощи, включающая описания особенностей реализации языка *C*, библиотек стандартных и интерфейсных функций и пр.

Перечень расширений языка *C* в среде *CodeVisionAVR* (аналогично и в трансляторах других фирм):

- описатели ***sfrb/sfrw*** обеспечивают доступ к регистрам ввода/вывода (РСФ), (см.

файлы описания <*xxxx.h*>):

```
sfrb PINA=0x19;  /* 8 bit access to the SFR */
sfrw TCNT1=0x2c; /* 16 bit access to the SFR */
```

- выражение, состоящее из имени PBB, точки и номера бита обеспечивает доступ к прямо адресуемым битам PBB по адресам *0…1Fh* для ***sfrb*** и *0…1Eh* для ***sfrw*** (бит – элемент структуры типа байт или слово), например:

- выражение *PIND.0* эквивалентно *PIND & (1<<0)*, то есть возвращает состояние указанного бита («0» или «1»);

- выражение *PORTD.0 = 1* эквивалентно выражению *PORTD |= (1<<0)* – установка указанного бита в «1»;

- описатель типа переменной ***flash*** размещает указанную константу или массив констант в энергонезависимой памяти программ *FlashROM*:

- описатель типа переменной ***eeprom*** размещает указанную константу или массив констант в энергонезависимой памяти данных *EEPROM*;

- описатель типа переменной ***bit*** резервирует один бит в регистрах общего назначения *R2…R14* – для глобальных и *R15* – для локальных переменных;

- описатель функции ***interrupt [2]*** размещает указанную функцию по адресу вектора 2 и завершает ее тело командой выхода из подпрограммы прерываний, пример:

```
interrupt [2] void external_int0(void) {/* Place your code here */}
```

- директивы ***#asm*** и ***#endasm*** позволяют вставлять как отдельные команды, так и цепочки команд ассемблера в текст программы на языке *C*, например :

```
#asm("sei") /* enable interrupts */
```

Готовый проект Lab1.prj с текстом программы Lab1.c, загрузочными модулями Exe \ Lab1.hex, реализующей алгоритм с рис. 3.2 находится в папке C-CodeVision.

GNU-AVR (Win AVR)

WinAVR (http://winavr.sourceforge.net/[WinAVR Home Page]) – программный пакет для операционных систем семейства Windows, включающий в себя кросс-компилятор и инструменты разработки для МК серий AVR и AVR32 фирмы Atmel.

WinAVR и все входящие в него программы являются открытым программным обеспечением, выпущенным под лицензией GNU, но распространяются в скомпилированном виде.

Хотя WinAVR является полноценной средой разработки программ на языке C/C++, удобно его использовать в объединении со средой AVR-Studio, используя мощные средства отладки последней.

Основные компоненты пакета, используемые при работе в среде AVR-Studio:

- AVR GCC – оптимизирующий компилятор языков C/C++ для AVR (после установки WinAVR интегрируется в AVR-Studio);

- AVR-LibC – стандартная C-библиотека AVR для использования с GCC.

Если установка WinAVR выполнена на компьютере с установленной средой AVR-Studio, то компилятор и библиотека WinAVR автоматически «подключаются» и доступны из среды AVR-Studio. Создается впечатление работы без выхода из AVR-Studio, не требуется особых навыков для освоения меню и команл WinAVR, требуется только некоторое знакомство с самим компилятором AVR GCC и библиотекой AVR-LibC.

В Help среды AVR-Studio также интегрируются краткиая помощь (AVR GCC Plag-in Help) и ссылки на полное описание в Интернете (avr-libc Reference Manual).

В качестве русскоязычного описания можно рекомендовать серию статей опубликованых в журнале «Радиолюбитель» (http://www.simple-devices.ru/articles/7-soft/12-winavr-avr-studio). Там же можно найти краткое описание основных элементов языка С – полезно тем, кто «забыл» пройденное…

Поэтому здесь ограничимся краткими замечаниями по ходу рассмотрения примеров.

Пример 1 (продолжение).

Текст программы Примера 1 на языке С:

```
#include <avr/io.h>     //Файл описания символьных имен МК

int main()  {           //Начало основной программы
  PORTB = (1<<1)|(1<<0); //Подключение притягивающих резисторов к PB0, PB1
  DDRB = (1<<2);        //PB2 – выход

  while (1)  {          //Начало основного цикла
    if(PINB & (1<<0)) { //Если PB0 == 1,
      PORTB |= (1<<2);  // то установить PB2
    }
    if(PINB & (1<<1)) { //Если PB1 == 1,
      PORTB &= ~(1<<2); // то обнулить PB2
    }
  }   //Конец основного цикла
}     //Конец основной программы
```

Напомним некотрые «азы».

Знак равенства «=» служит для присовоения левой части выражения содержимого правой части. Знак двойной стрелки влево «<<» выполняет побитовый сдвиг левой части выражения влево на число разрядов, указанных в правой части. Знаки «|»,«&» выполняют побитовые логические операции «ИЛИ», «И» соответственно, знаки «|=»,«&= ~» присваивают результат описанных операций левому операнду, знак тильда в последней группе выполняет побитовую инверсию, в итоге получаем обнуление указанного бита.

Бесконечное выполнение основного цикла образовано оператором цикла while с постоянным «истинным» условием 1.

Фигурные скобки после операторов if можно было не прменять, так как выполняемые действия представлены простыми выражениями, но транслятор разберется и так, а стиль лучше соблюдать…

Работа с проектом и трансляция в AVR-Studio

Файл проекта для среды AVR-Studio по расширению (*.aps) совпадает с вариантом для ассемблер, все файлы проекта Примера 1 находятся в папке ..GCC AVR. Можно воспользоваться готовым проектом, для его открытия достаточно в любом браузере кликнуть файл Lab1.aps.

Можно создать проект самостоятельно с помощью менеджера Project Wizard автоматически вызываемого при запуске среды AVR-Studio. Он в диалоговой форме предложит выбрать тип проекта (GCC AVR), путь и имя проекта, выбрать готовый файл с текстом программы Lab1.c или создать новый (и скопировать в него приведенный выше текст), выбрать платформу для отладчика (AVR Simulator) и тип МК (AT90S2323).

Вид экрана среды AVR-Studio при работе с проектом на языке C/C++ (с AVR GCC) отличается от вида на рис. 2.4 лишь в деталях.

Меню Project содержит пункт Configuration Options, при его выборе появляется всплывающее окно с несколькими закладками, рассмотрим лишь General (то есть общие, рис. 2.5).

В строке Output File Name имя файла образуется из имени проекта (Lab1) и расширения elf. Последнее обозначает тип файла, содержащего не только машинный код программы и данных, но и информацию для отладчика, обеспечивающую связь с текстом программы на зыке C. В следующей строке укаан подкаталог, в котором сохраняется как файл с расширением elf, так и другие служебные и выходные файлы проекта, в том числе файл Lab1.hex – только загрузочный машинный код программы.

В строке Device виден текущий выбор МК. Именно эта строка в объединении с первой строкой текста программы (`#include <avr/io.h>`) обеспечивают настройку на систему команд и объявление символьных имен регистров и битов выбранного МК.

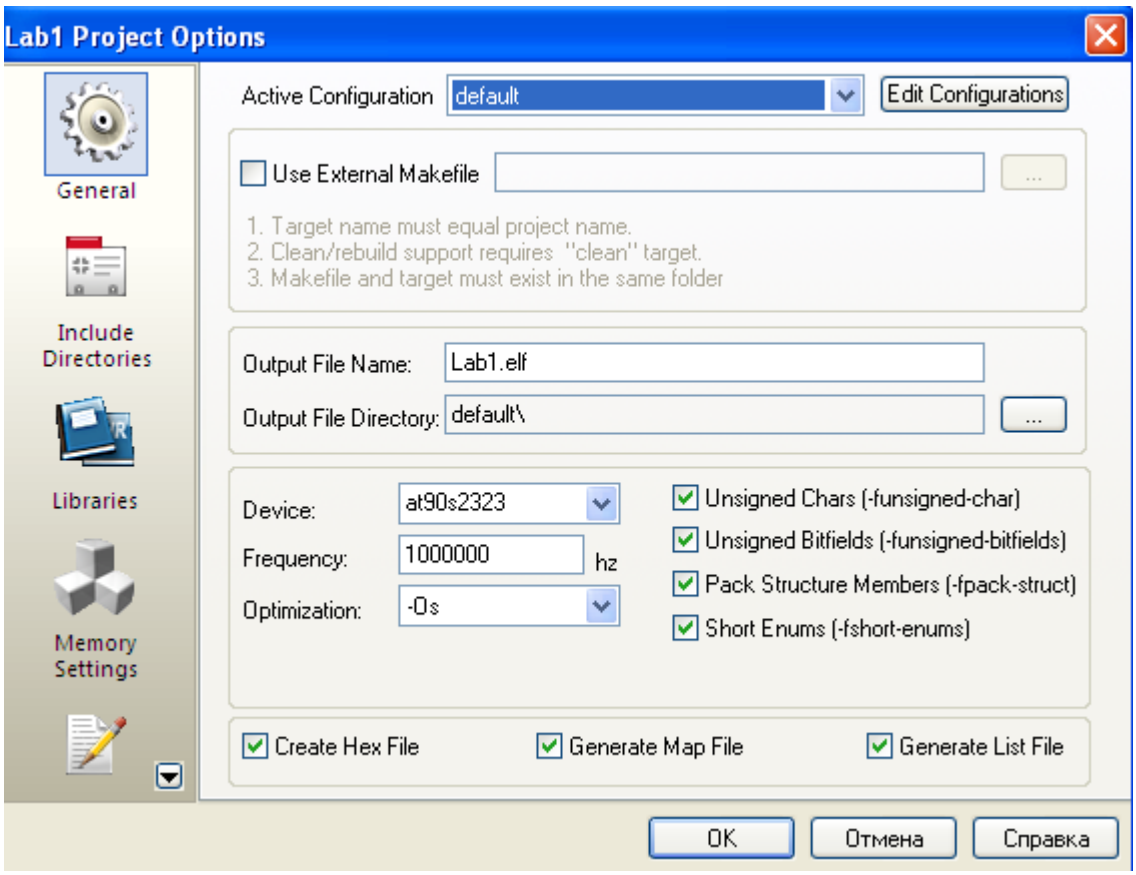


Рис. 2.5

В строке Frequency: 1000000 hz задана частота работы МК fclk = 1 МГц, по умолчанию число не задано. Его важно задавать только при использовании определенных библиотечных модулей, например, при использовании функции задержки delay_us() или delay_ms() (будут рассмотрены позднее).

В строке Optimization выбирается тип и уровень оптимизации, доступные опции:

-O0 – нет оптимизации – основной выбор для тестирования (!);

-O1 – оптимизация по скорости и размеру кода слабая;

-O2 – оптимизация по скорости и размеру кода средняя;

-O3 – оптимизация по скорости и размеру кода сильная;
-Os – оптимизация по размеру кода (O2+ по размеру).

Три «галки» в нижней строке разрешают создание файлов:

- «Hex» - загрузочный машинный код программы,

- «Map» - карта распределения памяти,

- «List» - листинг программы, то есть текстовый файл с исходным текстом и зпмещающими его строками машинного кода с мнемониками ассемблерных команд.

После внесения необходимых изменений в настройки проекта следует их сохранить через команды меню Project → Save Project.

Процесс трансляции запускается командой меню Build → Build или соотвтествующей пиктограммой. При успешной трансляции в нижнем окне Build появляется сообщение «Build succeeded with 0 Warnings...», выше расположена статистика использования ПП и ПД МК.

Сообщение «Build failed with N errors ...» говорит о наличии ошибок трансляции, это не позволяет создать загрузочный код. В списке выше этого сообщения обязательно есть строки сообщений о расположении и типе ошибки, помеченные красной точкой. Следует найти самое первое из таких сообщений и кликнуть его, в результате курсор и синий маркер поместятся на строку, вызвавшую ошибку. Для языка C/C++ важно исправлять в первую очередь именно первую ошибку, так как следующие «ошибки» могут быть вызваны не правильным пониманием транслятора границ выражения, в котором встретилась первая ошибка.

Сообщение «Build succeeded with N Warnings...» говорит об успешном завершении трансляции (загрузочный файл создан), но есть предупреждения о возможных ошибках. В списке выше есть строки, помеченные желтой точкой, указывающие на расположение и тип предупреждения. Желательно добиваться отсутствия предупреждений или, хотя бы, хорошо понимать их смысл.

При выборе опции -O0 (без оптимизации) размер программы Примера 1 составляет 114 байт ПП, при выборе -Os (средняя оптимизация по размеру) – 44 байта ПП, ПД в обоих вариантах не используется, так как нет переменных. Для сравнения напомним, размер программы на ассемблере – 24 байта.

Для любознательных приводятся варианты листинга для двух вариантов оптимизации.

```
Disassembly of section .text: (Optimization: -Os)

00000000 <__vectors>:
   0:   02 c0          rjmp    .+4             ; 0x6 <__ctors_end>
   2:   07 c0          rjmp    .+14            ; 0x12 <__bad_interrupt>
   4:   06 c0          rjmp    .+12            ; 0x12 <__bad_interrupt>

00000006 <__ctors_end>:
   6:   11 24          eor     r1, r1
   8:   1f be          out     0x3f, r1        ; 63
   a:   cf ed          ldi     r28, 0xDF       ; 223
   c:   cd bf          out     0x3d, r28       ; 61
   e:   02 d0          rcall   .+4             ; 0x14 <main>
  10:   0b c0          rjmp    .+22            ; 0x28 <_exit>

00000012 <__bad_interrupt>:
  12:   f6 cf          rjmp    .-20            ; 0x0 <__vectors>

00000014 <main>:
#include <avr/io.h>
```

```
int main()  {
  PORTB = (1<<1)|(1<<0);
  14:   83 e0          ldi     r24, 0x03      ; 3
  16:   88 bb          out     0x18, r24      ; 24
  DDRB = (1<<2);
  18:   84 e0          ldi     r24, 0x04      ; 4
  1a:   87 bb          out     0x17, r24      ; 23

  while (1)  {
    if(PINB & (1<<0)) {
  1c:   b0 99          sbic    0x16, 0 ; 22
      PORTB |= (1<<2);
  1e:   c2 9a          sbi     0x18, 2 ; 24
    }
    if(PINB & (1<<1)) {
  20:   b1 9b          sbis    0x16, 1 ; 22
  22:   fc cf          rjmp    .-8            ; 0x1c <main+0x8>
      PORTB &= ~(1<<2);
  24:   c2 98          cbi     0x18, 2 ; 24
  26:   fa cf          rjmp    .-12           ; 0x1c <main+0x8>

00000028 <_exit>:
  28:   f8 94          cli

0000002a <__stop_program>:
  2a:   ff cf          rjmp    .-2            ; 0x2a <__stop_program>

Disassembly of section .text: (Optimization: -O0)

00000000 <__vectors>:
   0:   02 c0          rjmp    .+4            ; 0x6 <__ctors_end>
   2:   07 c0          rjmp    .+14           ; 0x12 <__bad_interrupt>
   4:   06 c0          rjmp    .+12           ; 0x12 <__bad_interrupt>

00000006 <__ctors_end>:
   6:   11 24          eor     r1, r1
   8:   1f be          out     0x3f, r1       ; 63
   a:   cf ed          ldi     r28, 0xDF      ; 223
   c:   cd bf          out     0x3d, r28      ; 61
   e:   02 d0          rcall   .+4            ; 0x14 <main>
  10:   2e c0          rjmp    .+92           ; 0x6e <_exit>

00000012 <__bad_interrupt>:
  12:   f6 cf          rjmp    .-20           ; 0x0 <__vectors>

00000014 <main>:
#include <avr/io.h>

int main()  {
  14:   df 93          push    r29
  16:   cf 93          push    r28
  18:   cd b7          in      r28, 0x3d      ; 61
  1a:   de b7          in      r29, 0x3e      ; 62
  PORTB = (1<<1)|(1<<0);
  1c:   e8 e3          ldi     r30, 0x38      ; 56
  1e:   f0 e0          ldi     r31, 0x00      ; 0
  20:   83 e0          ldi     r24, 0x03      ; 3
  22:   80 83          st      Z, r24
  DDRB = (1<<2);
  24:   e7 e3          ldi     r30, 0x37      ; 55
  26:   f0 e0          ldi     r31, 0x00      ; 0
  28:   84 e0          ldi     r24, 0x04      ; 4
  2a:   80 83          st      Z, r24

  while (1)  {
    if(PINB & (1<<0)) {
```

```
  2c:   e6 e3           ldi     r30, 0x36       ; 54
  2e:   f0 e0           ldi     r31, 0x00       ; 0
  30:   80 81           ld      r24, Z
  32:   88 2f           mov     r24, r24
  34:   90 e0           ldi     r25, 0x00       ; 0
  36:   81 70           andi    r24, 0x01       ; 1
  38:   90 70           andi    r25, 0x00       ; 0
  3a:   88 23           and     r24, r24
  3c:   39 f0           breq    .+14            ; 0x4c <__SREG__+0xd>
      PORTB |= (1<<2);
  3e:   a8 e3           ldi     r26, 0x38       ; 56
  40:   b0 e0           ldi     r27, 0x00       ; 0
  42:   e8 e3           ldi     r30, 0x38       ; 56
  44:   f0 e0           ldi     r31, 0x00       ; 0
  46:   80 81           ld      r24, Z
  48:   84 60           ori     r24, 0x04       ; 4
  4a:   8c 93           st      X, r24
    }
    if(PINB & (1<<1)) {
  4c:   e6 e3           ldi     r30, 0x36       ; 54
  4e:   f0 e0           ldi     r31, 0x00       ; 0
  50:   80 81           ld      r24, Z
  52:   88 2f           mov     r24, r24
  54:   90 e0           ldi     r25, 0x00       ; 0
  56:   82 70           andi    r24, 0x02       ; 2
  58:   90 70           andi    r25, 0x00       ; 0
  5a:   00 97           sbiw    r24, 0x00       ; 0
  5c:   39 f3           breq    .-50            ; 0x2c <main+0x18>
      PORTB &= ~(1<<2);
  5e:   a8 e3           ldi     r26, 0x38       ; 56
  60:   b0 e0           ldi     r27, 0x00       ; 0
  62:   e8 e3           ldi     r30, 0x38       ; 56
  64:   f0 e0           ldi     r31, 0x00       ; 0
  66:   80 81           ld      r24, Z
  68:   8b 7f           andi    r24, 0xFB       ; 251
  6a:   8c 93           st      X, r24
  6c:   df cf           rjmp    .-66            ; 0x2c <main+0x18>

0000006e <_exit>:
  6e:   f8 94           cli

00000070 <__stop_program>:
  70:   ff cf           rjmp    .-2             ; 0x70 <__stop_program>
```

2.4. Средства отладки для выявления логических и схемотехнических ошибок

Отсутствие ошибок на этапе трансляции прораммы гарантирует отсутствие синтаксических ошибок, но не отсутствие логических, вычислительных или алгоритмических ошибок.

К таким ошибкам относятся неправильное использование регистров специальных функций и их битов, ошибки адресации к переменным или массивам, неверно рассчитанные коэффициенты в арифметических выражениях, не соблюдение требований к времени выполнения и многие другие.

Ошибки на уровне схемы так же весьма разнообразны: неправильно выбранные номиналы резисторов, конденсаторов, типов диодов, транзисторов, неверное понимание логики работы микросхем, функций и характеристик отдельных выводов микросхем и пр.

Рассмсотрим способы и средства отладки в порядке от простого к сложному.

1. Отладка с использованием программной модели МК (симулятор – *Debugger*) – самый простой и дешевый способ, его полнота ограничена возможностями программной

модели. Почти во всех «средах» программирования имеется подобный отладчик. В базовом варианте программная модель выполняет функции ядра МК – выполнение машинных команд процессором и взаимодействие с основными областями памяти. Функции ввода/вывода реализуются далеко не всеми моделями, в лучшем случае выполняются чисто цифровые функции, включая ввод/вывод через порты.

Процесс отладки состоит в пошаговом выполнении программы и просмотре результатов выполнения по содержимому регистров и ячеек памяти, редактирование содержимого регистров позволяет имитировать внешние воздействия на МК. Размер шага в программе может выбираться от одной машинной команды до подпрограмм или функций на языке C/C++, также существует мощный механизм задания точек останова (breakpoints) от самого простого – по заданному адресу, до сложного – по заданному состоянию выбранных ячеек памяти или регистров.

Более подробно мы познакомимся с симулятором среды AVR-Studio, который наиболее полно моделирует все разнообразие выпускаемых моделей МК семейства AVR.

2. Отладка с использованием схемно-программной модели МК в среде моделирования электронных устройств *Proteus VSM*. Наиболее полный способ тестирования целевой схемы или платы до ее реализации. Здесь доступны:

а) пошаговые методы отладки программы,

б) схемотехнические приемы: наблюдение формы сигналов напряжения, тока на выводах компонентов, имитация тестовых воздействий на входы богатым набором источников сигналов, наконец,

в) псевдовизуальная имитация человеко-машинного интерфейса в виде интерактивных моделей кнопок, тумблеров, клавиатур, потенциометров, светодиодов, светодиодных индикаторов, жидкокристаллических индикаторов и пр.

Трудоемкость этого способа связана как с высоким уровнем требуемых знаний и навыков моделирования, так и с затратами времени на поиск, предварительное тестирование моделей компонентов и микросхем, не редко отсутствующую модель приходится делать самому. Неизбежные упрощения при моделировании не гарантируют работоспособность реального устройства при работоспособности модели, но помогают проверить очень многие элементы схемотехники и программы до затратного периода производства печатной платы. Ниже на примерах рассмотрим некоторые способы тестирования в среде моделирования *Proteus VSM*.

3. Отладка на плате-прототипе. Обычно для этого используют сравнительно недорогие готовые платы с целевым МК и некоторым набором периферийных устройств. Обычно в комплект поставки включается минимально необходимый набор средств программирования и примеры готовых прикладных программ. Полезно при «знакомстве» с МК вообще или с новыми для пользователя моделями МК, облегчая начальный цикл освоения.

4. Отладка в целевой плате – обязательный завершающий этап разработки, наличие на целевой плате элементов человеко-машинного интерфейса (от светодиодов и кнопок до символьных или графических дисплеев) значительно упрощает процесс отладки.

Внутрисхемный эмулятор (*In Circuit Emulator - ICE*) является самым эффективным инструментом отладки на целевой плате, сочетая удобства *Debugger'a* с работой в реальном времени. Современные технологии используют размещенные в кристалле МК

специализированные последовательные интерфейсы, поддерживающих основные команды отладки средствами внутри кристалла.

В частности, подсемейства *ATmega* и *ATxmega* оснащены встроенным четырехпроводным интерфейсом *JTAG*, подсемейства *ATtiny* – однопроводным интерфейсом *debugWIRE*. Набор команд и методика отладки в среде AVR-Studio не отличаются от отладки в симуляторе.

Эти интерфейсы кроме загрузки/выгрузки кодов программ/данных выполняют функции внутрисхемного отладчика, то есть выполнение программы, останов программы в заданном месте, просмотра (и редактирование) содержимого областей памяти и регистров.

Симулятор в среде AVR-Studio

Процесс отладки в симуляторе предполагает выполнение разработанной программы в пошаговом или более быстром темпе, наблюдение изменения состояния регистров ввода/вывода, ячеек памяти (переменных), воздействие на модель, посредством изменения состояния регистров ввода/вывода или ячеек памяти (переменных). Наблюдение за изменением числа машинных циклов позволяет оценить расходы времени на выполнение заданных функций.

Наиболее функционально полным для МК семейства *AVR* следует признать симулятор в среде программирования и отладки *AVR_Studio*. Он моделирует работу как устройств ядра (процессор и все виды памяти), так и ряда встроенных периферийных устройств, прежде всего портов ввода/вывода, входов прерывания, таймеров/счетчиков, некоторых последовательных интерфейсов, цифровых узлов аналогового компаратора и аналого-цифрового преобразователя.

Все команды отладчика находятся в меню Debug, до запуска сеанса отладки большинство команды не активны («серые»). Активны только группа команд работы с точками останова (… Breakpoint, о них ниже), команда выбора симулятора и конкретной модели МК (Select Platform and Device) и команда запуска сеанса (Start Debugging).

Запуск сеанса отладки возможен также через команду Build and Run меню Build, в этом случае сначала выполняется компиляция и сборка загрузочного модуля, затем, при успешном завершении – запуск сеанса отладки. Ясно, что для успешного запуска необходим доступ проекта к загрузочному модулю и файлу (или группе файлов) с исходным текстом программы (на языке ассемблера или C/C++). При запуске сеанса отладки на другом компьютере нередко внутренние связи проекта не совпадают, это легко исправляется предшествующей сборкой командой Build and Run.

После запуска сеанса отладки:

- в окне текстового редактора появляется желтая стрелка, указывающая на первую выполнемую строку программы (рис. 2.6, а);
- в левом окне добавляется закладка Processor, содержащая значения частоты тактирования модели (Frequency), времени после сброса в машинных циклах (Cycle Counter) и в секундах (Stop Watch: us – мкс, ms - мс), ряда регистров процессора (в том числе указатель команды Program Counter) и РОН (Registers), значения РОН важны при отладке программы на ассемблере, при необходимости их можно редактировать;
- абсолютное время после сброса t определяется номером машинного цикла n и частотой

тактирования fclk: t = n / fclk; значения времени можно обнулять посредством контекстного всплывающего меню, значение частоты можно менять в настройке модели (описано ниже), изменение частоты приводит к сбросу;

- в окне I/O View в графе Value появляются численные значения регистров PBB (в большинстве случаев по умолчанию нулевые), в графе Bit эти числа разбиты на разряды в виде квадратиков-битов: белых при нулевых значениях, черных при единице; эти значения можно при необходимости менять, используя мышь и клавиатуру; изменяя значения битов регистров PINx человек имитирует воздействие внешней схемы на входы МК, подобное «воздействовие» на выходы допустимо, но невозможно в реальном МК;
- в меню Debug (рис. 2.6, б) и View (рис. 2.6, в) становится активна большая часть команд.

Рассмотрим кратко меню Debug.

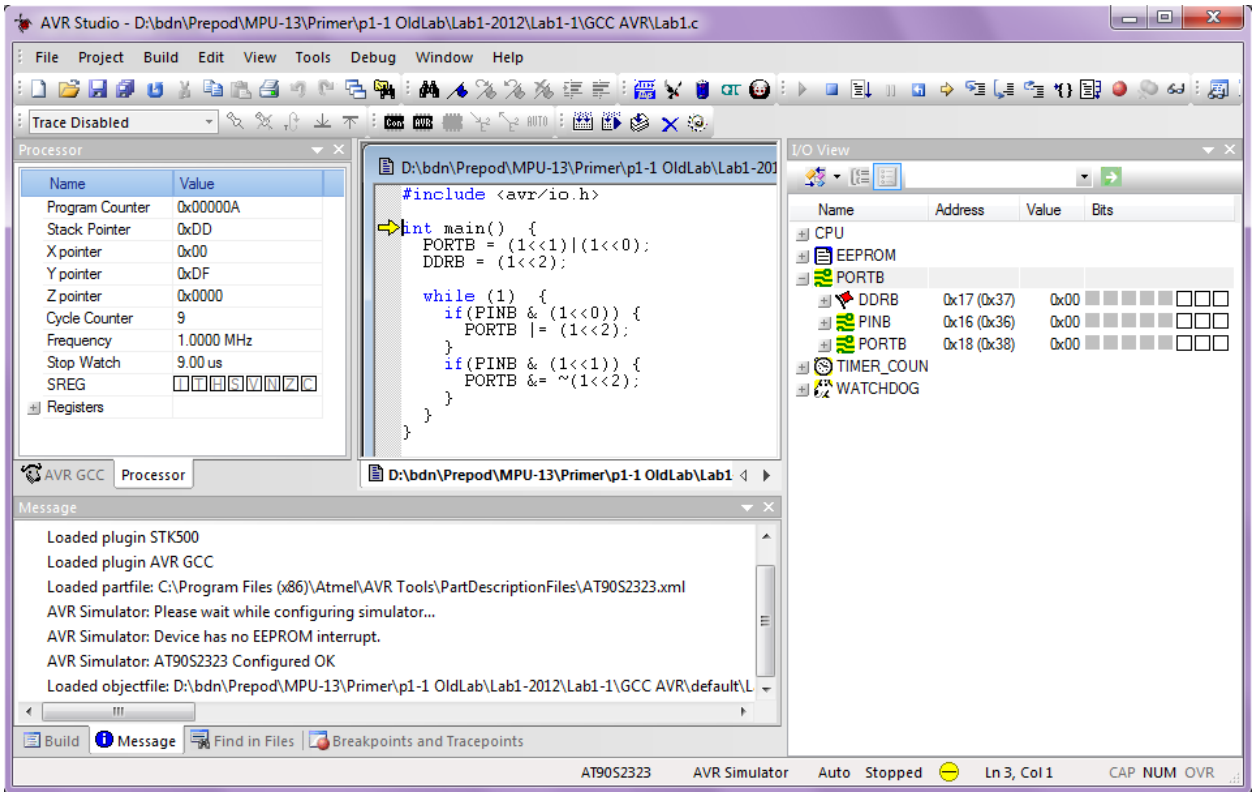
Stop Debugging – завершение сеанса отладки.

Run [F5] – запуск непрерывного выполнения программы; после этого индикация в окнах наблюдения не обновляется до останова по Break или Breakpoint.

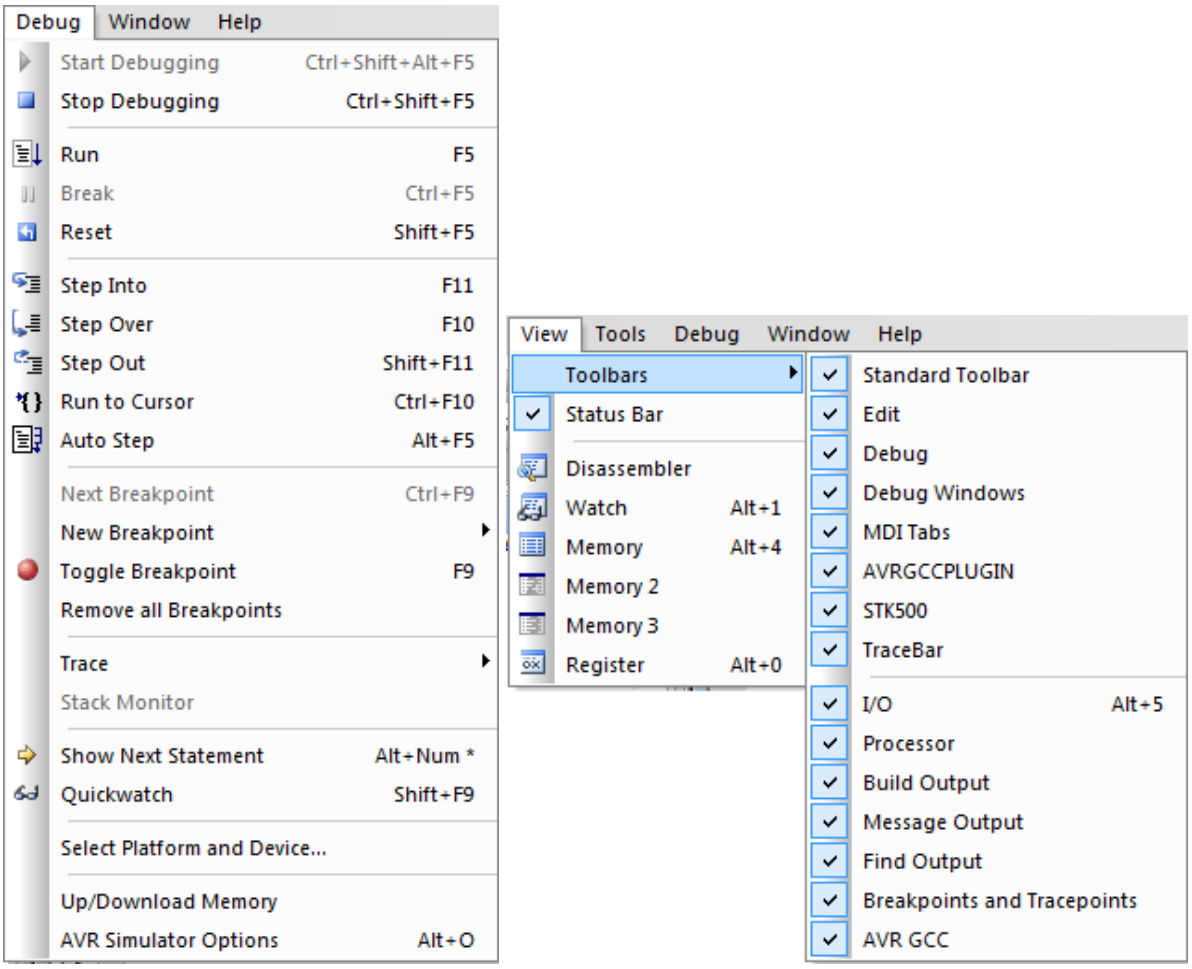
Break [Ctrl+F5] – останов непрерывного выполнения программы.

Reset [Shift+F5] – сброс, то есть останов выполнения программы и переход к началу программы.

Step Into [F11] – выполнение программы на одну строку исходного текста, «желтая стрелочка» перемещается на следующую строку программы, обновляется индикация значений в окнах регистров, памяти и пр.; при использовании оптимизации AVR GCC «желтая стрелочка» может то пропадать, то зависать на одно-два нажатия из-за некоторого несоответствия строк исходного текста машинным командам.



а



б в

Рис. 2.6

Step Over [F10] – выполнение программы на одну строку, но при встрече подпрограммы или функции, они выполняются сразу.

Step Out [Shift+F11] – выполнение подпрограммы или функции до конца и выход на строку, следующую за вызовом подпрограммы или функции.

Run ro Cursor [Ctrl+F10] – выполнение программы до курсора, курсор выделен мигающим «столбиком», перемещается мышью или клавишами навигации (стрелки и пр.).

Auto Step [Alt+F5] – запуск автоматического выполнения программы «шагами» до останова, индикация обновляется после очередного шага, длительность шага может подстраиваться.

Next Breakpoint – перейти к следующей точке останова (переход курсора, а не желтого указателя выполнения программы).

New Breakpoint – вызов подменю создания новой точки останова:

Program Breakpoint – выбором строки текста программы;

Data Breakpoint – по изменению данных (переменой);

Program Tracepoint – точка трассировки программы.

Toggle Breakpoint [F9] – установка/удаление точки останова в строке программы, на которую указывает курсор.

Remove all Breakpoints – удаление всех точек останова.

Trace – подменю управления трассировкой.

Stack Monitor – команда зарезервирована.

Show Next Statement – показать следующую выполняемую строку; удобно при отладке многофайловых проектов для возврата к «желтой стрелке».

Qwickwatch – добавление переменной, на которую указывает курсор в окно быстрого просмотра.

Select Platform and Device – вызов подменю выбора симулятора и конкретной модели МК (как и при создании проекта).

Up/Download Memory – вызов подменю выгрузки содержимого указанного блока памяти в файл или загрузки содержимого файла в указанный блок памяти.

AVR Simulator Options – вызов подменю настройки параметров модели МК:

Device Selection – выбор модели МК (Device) и частоты работы модели в МГц (Frequency, MHz); при выборе МК, не соответствущего выбору при сборке проекта отвественность за работоспособность несет человек; выбор частоты тактирования влияет только на временные характеристики алгоритма, симулятор ведет отсчет времени в машинных циклах.

Stimuli and Logging – стимуляция и логгирование портов; стимуляция – формирование воздействия на входы портов (биты входов PINx) из специально созданных файлов, логгирование – запись изменния выходов портов (PORTx) в файлы.

Механизм стимуляции и логгирования удобно использовать при высокоинтенсивном вводе или выводе через параллельные порты, когда отладка через окно I/O View становится слишком трудоемкой. Формат файлов стимуляции и логгирования одинаков и весьма прост:

```
0000000000:00
0000000039:01
0000000040:00
9999999999:FF
```

Здесь значение, указанное до разделителя, — десятичный номер машинного цикла с момента сброса микроконтроллера, после разделителя «:» — шестнадцатеричное значение восьмибитного сигнала. Одинаковый формат позволяет формировать файл тестового воздействия с помощью логгирования во вспомогательной программе.

Последняя строка файла стимуляции не должна быть использована в сеансе отладки, поэтому в приведенном примере указано максимально возможное число циклов.

Для подключения файла стимуляции в окне Stimuli and Logging в рамке Port выбирается требуемый порт PORTx, в рамке Function – Stimuli, в рамке Input → File следует ввести полное имя файла, воспользовавшись кнопкой вызова браузера. После этого кнопкой Add Entry добавить данный выбор в список Action List.

Аналгично, для подключения файла логгирования в окне Stimuli and Logging в рамке Port выбирается требуемый порт PORTx, в рамке Function – Logging, в рамке Input → File следует ввести имя файла, кнопка To Screen позволяет дублировать вывод лога на экран (в окно Message). После этого кнопкой Add Entry добавить данный выбор в список Action List.

В этот список можно включить несколько файлов стимуляции или логгирования, для удаления файла служит кнопка Delete Entry.

В начале лог файла всегда находится строка 0000000:00, это соответствет настройкам порта по умолчанию. Если изменений на выходах данного порта не было,

новых строк не будет. Если лог файл не стирать, то он пишется в продолжение (со строкой 0000000:00). Так же и после рестарта.

Проект «помнит» настройки логов и стимулов.

Рассмотрим кратко меню View.

Toolbars – подменю видимости набора окон среды AVR-Studio.

Status Bar – выключение строки состояния внизу экрана: модель МК, тип и состояние отладчика, координаты курсора.

Disassembler – переход к отладке в машинном коде: в окне редактора текст в виде листинга, «желтая стрелка» перемещается по строкам ассемблера; этот режим удобен при знакомстве с системой команд, для понимания логики компилятора при выборе уровня оптимизации.

Watch – открыть окно просмотра и редактирования именованных переменных и массивов в виде таблицы с колонками Name – имя, Value – значение в десятичном или шестнадцатиричном (Hex) формате, Type – тип, Location – адрес, массивы отображаются в «свернутом виде», при нажатии на «крестик» возле их имени они разворачиваются.

Memory, Memory 1, Memory 2 – открыть окна просмотра и редактирования содержимого различных областей памяти: Data – энергозависимая ПД, EEPROM – энергонезависимая ПД, Program – ПП, I/O – РВВ (РСФ), Register – РОН, адрес и данные отображаются в шестнадцатиричном формате, для данных можно выбирать одно- и двух байтный формат, а также формат ASCII (символьный).

Register – открыть окно просмотра и редактирования значений РОН в шестнадцатиричном (Hex), десятичном (Dec), символьном (Ascii) или двоичном (Binary) формате.

Для сохранения настроек отладчика (положения окон и их состава, точек прерывания, подключения файлов логгирования и стимуляции и пр.) используйте команду сохранения проекта Project → Save Project.

При необходимости редактирования текста программы можно остановить сеанс отладки (кнопка Stop Debugging), внести измения в текст, запустить повторную сборку и запуск отладки (Build and Run). Но можно сразу внести изменения в текст, тогда после вызова любой команды «хода» отладчика появится всплывающее меню, где вам предложат выбор: выполнить перекомпиляцию и сборку либо продолжить отладку со старой сборкой. Последний выбор оправдан при трудоемкой отладке больших проектов, когда исправления важно запомнить, но они не мешают дальнейшему ходу отладки.

Пример 1 (продолжение).

Тестирование программы Примера 1 на ассемблере.

Запуск сеанса отладки выполняется командой Start Debug или Build and Run.

Старт программы начинается с нулевой ячейки памяти (Program Counter = 0), счетчик модельного времени также обнулен (Cycle Counter = 0, Stop Watch = 0), «желтая стрелка» в окне редактора указывает на команду jmp START, она еще не выполнена.

При нажатии на клавиатуре функциональной клавиши F11 (Step Into) симулятор выполняет одну команду ассемблера (jmp START) и останавливается, желтая стрелка перемещается на строку с командой, следующей за меткой START. Значения указателя программы меняется в соотвествии с адресом перехода (3), счетчики времени увеличиваются на время выполнения команды (2 машинных циклах и 2 мкс). Кроме

перехода к следующей команде никаких действий эта команда не выполняет, поэтому значения остальной индикации не меняется.

Следующее нажатие <F11> приводит к выполнению команды <ldi r16, 0b0100>, в результате константа помещается в РОН r16 (= 0x04), значение указателя программы и счетчиков времени увеличились на 1.

После третьего нажатия <F11> константа 0x04 попадает в регистр направления порта B DDRB, это отображается численно и в виде битов в окне I / O View.

Таким образом можно проследить пошаговое выполнение программы в отладчике.

При первом выполнении команды, следующей за меткой LOOP, значение времени показывает программную задежку инициализации.

При каждом следующем прохлже этой команды выполняется основной цикл, для данного примера это требует 6 машинных циклов.

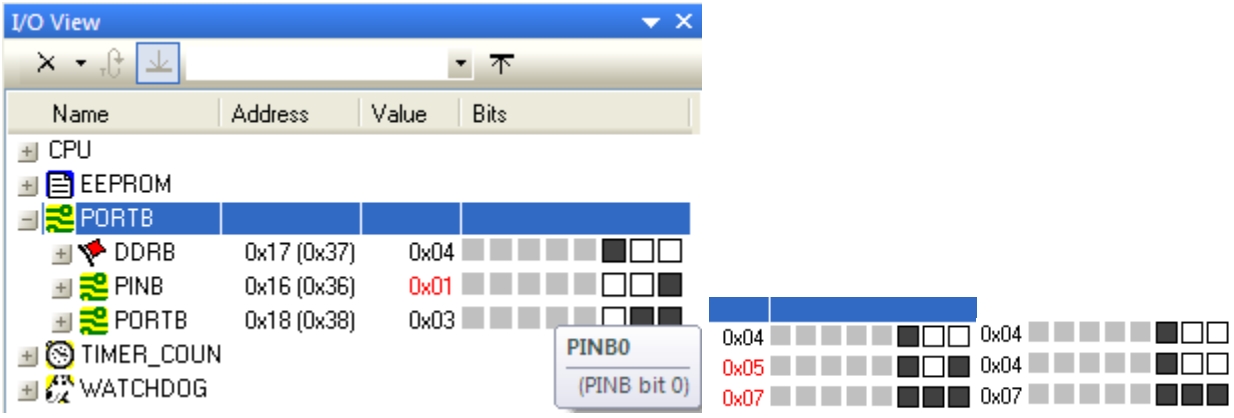
Имитация воздействия кнопок.

Клик мышкой над битом 0 регистра PINB меняет его значение с 0 на 1 (квадратик темнеет, рис. 2.7, а), так имитируется нажатие кнопки «Вкл». При выполнении команд цикла сначала программа установит бит 2 регистра PORTB, на следущем машинном такте драйвер порта установит в 1 бит 2 регистра PINB (рис. 2.7, б). Это означает, что выход PB2 перешел в ВЫСОКОЕ состояние.

Следущий клик над PINB.0 возвращает вход PB0 в 0 (квадратик белеет, рис. 2.7, в). На это изменение входа наша программа не реагирует.

Аналогичные действия над PINB.1 имитируют нажатие и отжатие кнопки «Выкл», что приводит к переводу выхода PB2 в НИЗКОЕ состояние (сначала обнуляется PORTB.2, затем PINB.2).

Наша программа может заметить эти манипуляции только если между нажатиями и отжатиями будет выполнен хотя бы один проход основного цикла.



а б в

Рис. 2.7

Проверим поведение МК в «запрещенном» состоянии, для этого одновременно установим в 1 биты 0 и 1 регистра PINB. Теперь при выполнении цикла программы выход PB2 последовательно сначала устанавливается в ВЫСОКОЕ состояние, затем в НИЗКОЕ. Фактически мы получили генерацию импульсов с периодом 6 мкс. Перевод любого из входов в 0 прекращает генерацию.

Теперь о манипулциях с остальными битами регистров порта. Оператор может кликом мышки инвертировать состояние любого бита любого регистра в окне I/O View.

Но воздействию внешней схемы на входы соответствует только инверсия битов регистра PINx соответствующие биты регистра DDRx которого находятся в 1, то есть, настроены как входы.

Период основного цикла удлиняется при «нажатых» кнопках (PINB.0 = 1 или PINB.1 = 1) до 7 мкс.

Рассмотрим способ измерения периода эффективный при значительном числе команд основного цикла. Установите точку прерывания (Togle Breakpoint <F9>) на любой строке основного цикла. Выполните требуемые действия над входами. Обнулите время (всплывающее меню в окне Processor, Reset Stopwatch), запустите непрерывное выполнение программы (Run <F5>). Если в основном цикле нет других точек прерывания, программа выполнит полный цикл и остановится, результат измерения периода находится в окне Processor, Stop Watch. Если же в основном цикле несколько точек прерывания, программа остановится на ближайшей, несколько шагов по <F5> доведут вас до цели.

Тестирование программы Примера 1 на языке С.

После запуска сеанса отладки «желтая стрелка» в окне редактора указывает на первую строку функции main(), но этому предшествовало выполнение части скрытого машинного кода. Об этом нам сообщают не нулевые значения указателя программы (Program Counter = 0x0A = 9) и счетчиков модельного времени (Cycle Counter = 9, Stop Watch = 9 us).

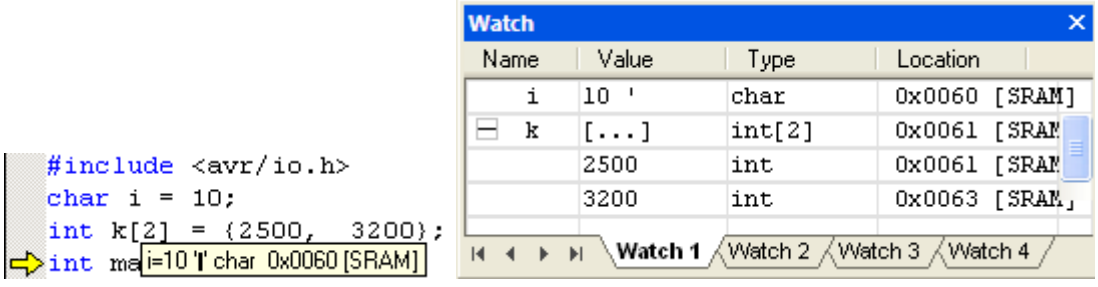
Объем скрытого кода и задержка выполнения первой строки программы растут при появлении и росте объема переменных и библиотечных функций. Это надо учитывать при расчете времени этапа инициализации МК.

Методика использования шагов отладчика не отличается от описанного в примере на ассемблере, кроме того, что минимальный шаг определяется строкой исходного текста. Для любителей писать «плотный» текст (по несколько выражений в строке) это затрудняет просмотр промежуточных результатов.

Период основного цикла в данной реализации от 5 до 8 мкс, 9 мкс – в «запрещенном» состоянии при генерации импульсов с периодом равным периоду основного цикла.

Для понимания реализации выражений языка С/С++ машинными командами можно перейти в режим дизассемблера, тогда минимальный шаг выполнения составить одну команду.

При отладке программы на языке С/С++ важно следить за состоянием именованных переменных. Для этого удобно использовать несколько механизмов наблюдения.



а б

Рис. 2.8

Наш исходный пример не требовал использования переменных, поэтому в

закомментированной части добавлены одна переменная и один массив. Уберите комментарии, выполните сборку и вход в отладку. Теперь расходы ПП выросли до 200 байт и появились расходы ПД (SRAM) – 6 байт.

При наведении курсора на имя переменной (если она определена на данном этапе) появляется всплывающее окошко (рис. 2.8, а): имя переменной, значение десятичное и ‘ASCII’, тип и адрес размещения, так же отображаются и функции. При наведении на переменную из массива видны только размерность и адрес.

Для наблюдения значений переменных в динамике удобнее использовать окно Watch (рис. 2.8, б), оно имеет до 4 страниц-закладок, в плавающем состоянии (выбор Floating во всплывающем меню) положение окна и все его визуальные границы настриваются мышью, можно закрепить положение в одном из фиксированных окон (выбор Docking). Занести переменную или массив в окно Watch можно двумя способами. Традиционный заключается во вводе имени в графу Name с использованием всплывающего меню. Другой способ заключается в выделении имени переменной в тксте программы, вызове вплывающего контекстного меню и выборе Add Watch: “имя переменной”.

В данном примере использованы только глобальные переменные, они, как известно, видны везде. Значения локальных переменных с ограниченной областью видимости в зоне их неопределенности индицируются как Not in scope.

Отладка в системе моделирования Proteus VSM

Среда сквозного проектирования электронных устройств *Labcenter Electronic* (*Proteus v.7.1 SP2, http://www.labcenter.co.uk*) состоит из пакета проектирования электрических принципиальных схем *ISIS (Intelligent Schematic Input System)* и пакета проектирования (разводки) многослойных печатных плат *ARES* (*Advanced Routing end Editing Software*).

Пакет *ISIS* содержит графический редактор схем, библиотеки компонентов (схемное изображение, модель, изображение для печатной платы). Как и многие современные системы схемотехнического проектирования *ISIS* содержит средства аналого-цифрового моделирования: пакет *Proteus VSM*: состоит из математического ядра *PSPICE (Berkeley SPICE3F5)*, средств визуализации сигналов и состояний модели. Особенностью пакета *Proteus VSM* является библиотека цифровых моделей микроконтроллеров (прежде всего 8-разрядных, включая *AVR*), позволяющая моделировать их поведение под управлением программы (встроенные ассемблеры или загрузка исполняемого кода в *hex, cof* и других форматах).

Сеанс моделирования можно проводить в двух режимах.

В интерактивном режиме (***interactive simulation***) расчет ведется в «реальном» времени с возможностью запуска, останова и паузы. Во время паузы доступны обычные функции отладки. При загрузке файла программы в формате *cof (CodeVision*) или *elf* (*AVR_GCC*), поддерживается символьная отладка на уровне исходного текста на языке *C*/*C*++. Средства визуализации (цифровые индикаторы напряжения, тока, осциллографы и пр.), различные индикаторы (лампы, светодиоды, ЖКИ), модели компонентов с анимацией (двигатели, вентиляторы) и пр. позволяют следить за состоянием и поведением схемы. Интерактивные элементы (кнопки, переключатели, потенциометры и др.) позволяют воздействовать на состояние и поведение схемы.

В режиме временной диаграммы (***graph based simulation***) производится расчет заданного интервала времени и визуализация временных диаграмм выбранных сигналов (маркеры сигналов: логических, напряжения, тока и пр.) в специальном окне (аналоговое, цифровое, смешанное и пр.).

На рис. 2.9, а показаны основные элементы окна *ISIS*:

1 - окно графического редактора схем (*Editing Window*),

2 - окно выбора объектов (*Object Selector*),

3 - окно обзора (*Overview Window*),

вверху - строки меню и пиктограмм,

столбец слева - пиктограммы выбора режима и доступа к библиотекам элементов (рис. 2.9, б),

внизу:

слева - кнопки запуска/останова интерактивного режима (*Play, Step, Pause, Stop*),

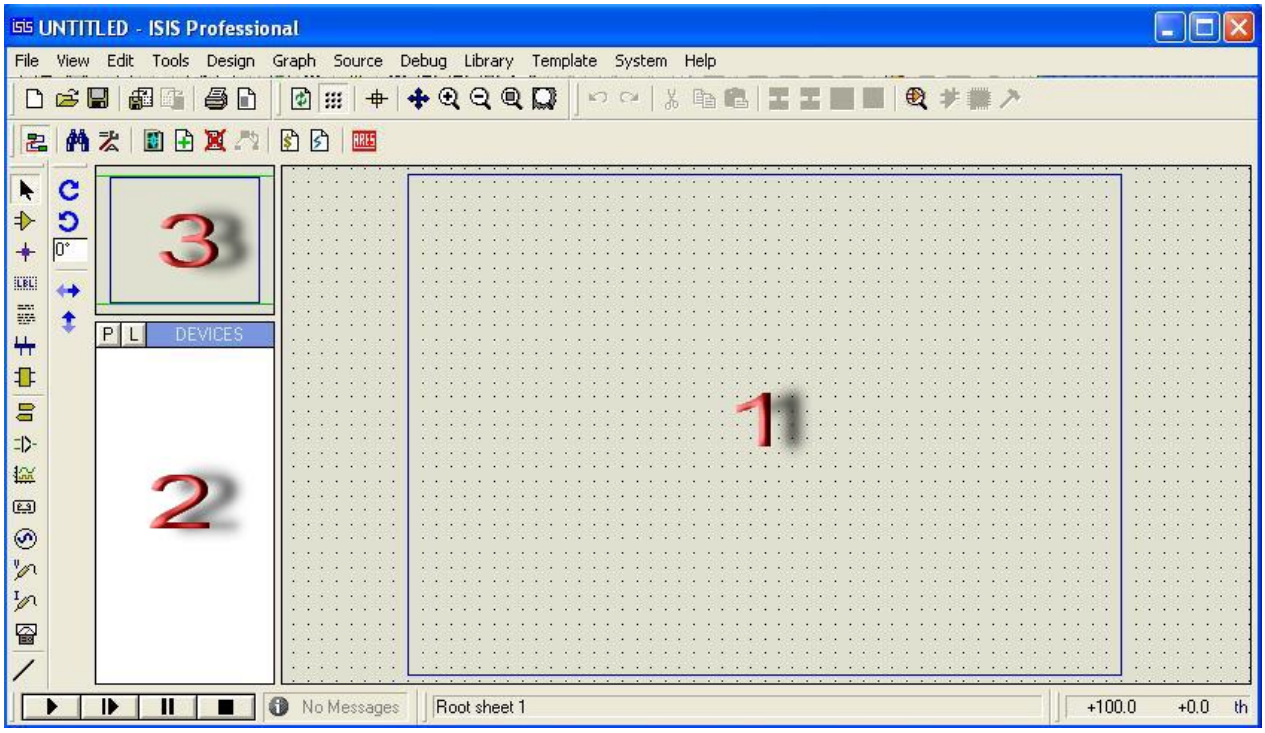
правее - наличие и вызов окна сообщений (*Messages*),

правее - строка состояния и подсказок,

крайнее правое - координата курсора в окне редактора или окне панорамы.

В меню *Help* кроме описания пакета в целом (*Proteus VSM Help*) и редактора в частности (*ISIS Help*) расположен богатый набор готовых примеров (*Sample Designs),* структурированных как по виду деятельности, так и по семействам микроконтроллеров.

Изменение масштаба наиболее просто выполняется "колесом" мыши от текущей позиции курсора. Изменение положения окна редактора по отношению к листу схемы удобно менять в окне обзора.



а

1. Selection Mode – режим выбора
2. Component Mode – просмотр компонентов проекта в окне выбора
3. Junction Dot Mode – режим прорисовки узла соединений (точки)
4. Wire Label Mode – режим метки провода (соединения)
5. Text Script Mode
6. Buses Mode – режим рисования шин
7. Subscript Mode
8. Terminals Mode – Библиотека «клемм»: 0, питания, связей цепей и шин
9. Device Pin Mode – режим рисования выводов (при создании/ред. компонента)
10. Graph Mode – Библиотека графиков (аналоговых, дискретных и пр.)
11. Tape Recorder Mode – режим установки устройства записи сигналов
12. Generator Mode – Библиотека источников тестовых сигналов
13. Voltage Probe Mode – Библиотека флагов измерения напряжения/лог.сигн.
14. Current Probe Mode – Библиотека флагов измерения токов
15. Virtual Instrument Mode – Библиотека интерактивных инструментов (осцилл…

б

Рис. 2.9

Создание и редактирование схемы

На начальном этапе освоения удобнее пользоваться готовыми примерами. Они хранятся в файлах с расширением <*.dsn>, двойной клик по имени запускает работу среды ISIS с данным проектом. Одновременно могут быть запущены несколько проектов, каждый работает независимо в своем окне. Готовые примеры удобно использовать для создания своих проектов, используя команду File → Save Desine As….

Создание схемы «с нуля» начинается с запуска среды ISIS (из меню Пуск: Proteus 7 Professional → ISIS 7 Professional), далее команда меню File → New Design… Сразу после этого можно дать имя новому проекту (File → Save Desine As…), все сохраняется в одном файле с раширением <*.dsn>.

Далее надо заполнить окно выбора моделями компонентов из библиотеки.

Вызов окна библиотеки выполняется командой меню Library → Pick Device / Symbol…, появляется окно *Pick Devices* (рис. 2.10).

Слева расположены окощки поиска и выбора. Сверху находится окошко Keywords для автопоиска модели по первым введенным символам, чуть ниже – опиции настроек автопоиска (Match Whole Words? – по целому слову, Show Only Parts witf Models? – показывать только компоненты с моделями). Ниже представлены окошки выбора групп и подгрупп компонентов (*Category*, *Sub-category*, *Manufacturer* – производитель).

Центральное окно Results показывает список предварительного выбора моделей, здесь колонки Device – имя модели, Library – имя библиотеки, Description – кратное описание свойств. Любой компонент в библиотеке имеет схемное изображение, большая часть имеет модель (для Proteus VSM) и изображение корпуса на печатной плате.

Справа вверху находится окошко <Device> Preview со схемным изображением компонента и текстовым описанием типа модели (некторые компоненты не имеют моделей), ниже – окошко PCB Preview с изображением корпуса для размещения на печатной плате, внизу – выпадающий список доступных *наименований* корпусов.

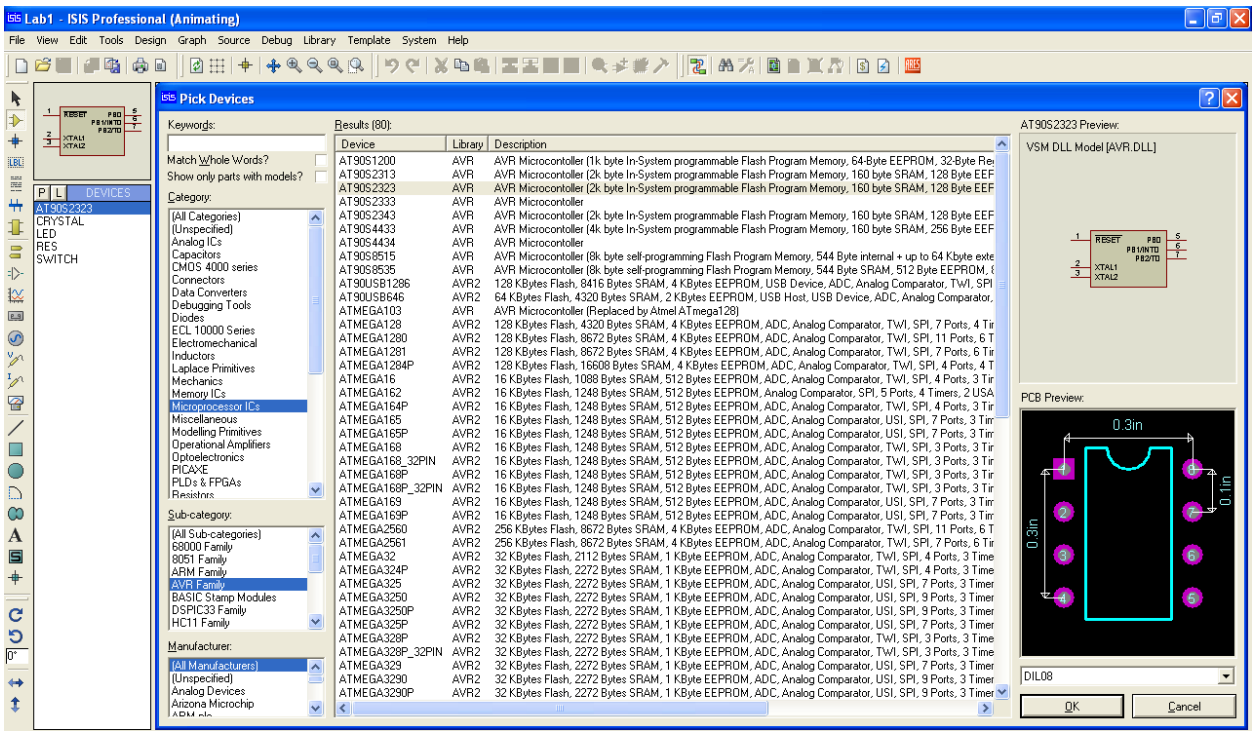


Рис. 2.10

Из списка доступных моделей двойным кликом выбираем требуемую, ее имя копируется в окно выбора среды *ISIS*. Закрываем окно *Pick Devices* и кликом в окне редактора схемы размещаем схемное изображение выбранного МК.

После заполнения окна выбора моделей не забудьте сохранить проект.

Далее в окне графического редактора схем размещаются изображения компонентов. Для этого кликните требуемый компонент в окне выбора, затем двойным кликом разместите его в окне редактора. Всплывающее меню позволяет изменить ориентацию компонента.

Далее прорисовываются электрические связи. Клик по выводу компонента, далее клик по другому выводу – связь готова. В ряде случаев удобнее делать промежуточные клики для уточнения пути линии связи. Иногда удобно создать узел вне вывода компонента, для этого используется пиктограмма Junction Dot Mode (рис. 2.9, б, п.3).

При размещении возле графического изображения компонента появляется текстовое имя и схемное обозначение (Component Reference). Последнее состоит из одного или двух латинских символов и порядкового номера, например, резисторы именуются R1, R2,…, конденсаторы C1, C2. Текстовое имя в пассивных компонентах заменяется численным значением номинала, например, 10k возле резистора означает сопротивление 10 кОм. Весь набор свойств компонента можно посмотреть командой всплывающего меню Edit Properties, состав свойств зависит от сложности модели. Для сложных моделей имеется встроенная подсказка (Help).

Модель МК серии AVR

Программно управляемые модели МК семейства AVR находятся в библиотеке компонентов в разделе *Category*: *Microprocessors ICs* → *Sub-category*: *AVR Family* или *Manufacturer*: *Atmel*. Все модели МК имеют тип DLL Model. Библиотека моделей МК AVR состоит из двух частей – старая AVR содержит в основном уже устаревшие модели

подсемейства Classic, AVR2 – современные подсемейства ATtiny, ATmega.

После размещения выбранного МК в окне редактора можно познакомиться с ним поближе. После двойного щелчка мыши на схемном изображении МК появляется окно параметров Edit Component (рис. 2.11).

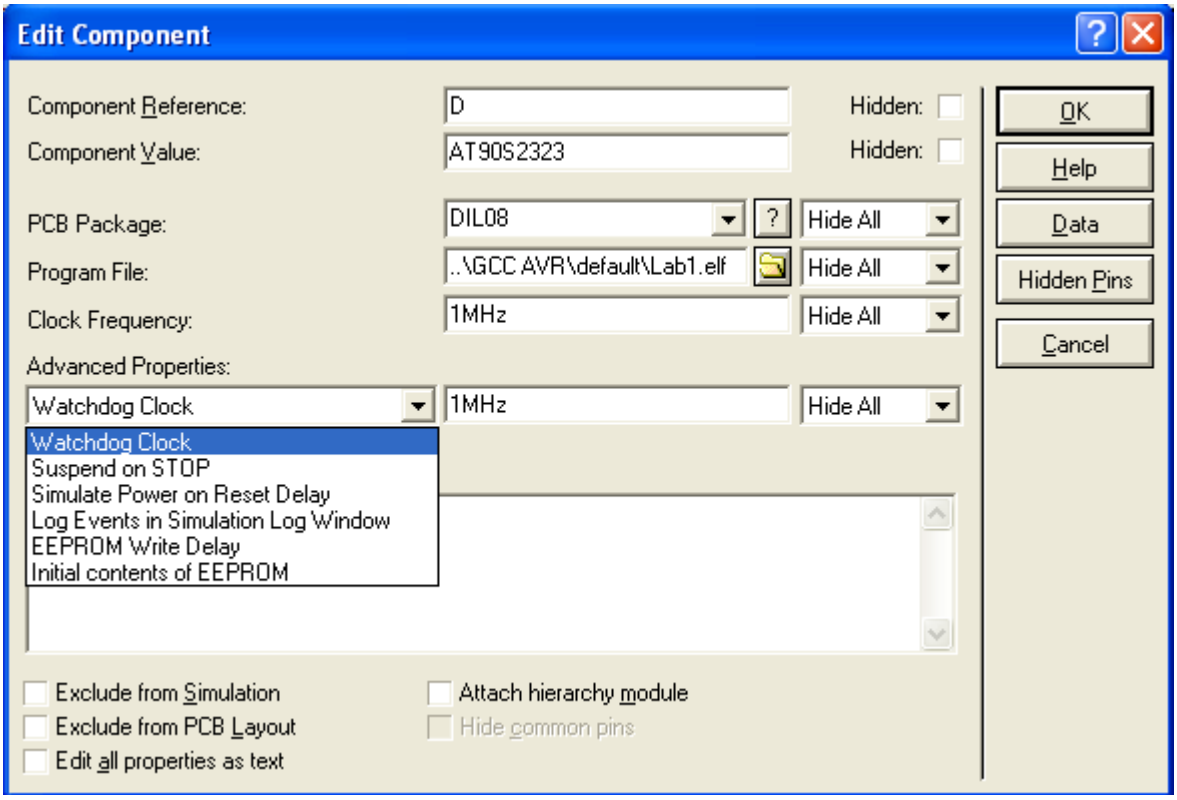


Рис. 2.11

Самой важной для нас является строка «*Program File*», в ней указывается файл с кодом исполняемой программы в формате **.hex* (машинный код), либо **.cof* или **.elf* (машинный код с привязкой к исходному тексту программы). Для выбора файла используется кнопка вызова браузера  . Выпадающий список Hide All (Скрыть все) управляет визуализацией элементов данной строки, остальные позиции выбора: Show All – Показать все, Hide Name – Скрыть имя, Hide Value – Скрыть значение.

В строке «Conponent Reference» указывается схемное обозначение компонента, по умолчанию обозначение микросхемы состоит из символа U и порядкового номера от 1, при единственном компоненте данного типа номер может отсутствовать, по ГОСТ Р микросхемы обозначают символом D. «Галка» в окошке «Hidden» позволяет скрыть изображение обозначения на схеме.

В строке «Component Value» находится имя модели.

В строке «PCB Package» находится список выбора корпуса микросхемы.

В строке «Clock Frequency» задается частота тактирования модели. Обращаем внимание, что кварцевый резонатор ZQ, подключенный к выводам XTAL1, XTAL2 не влияет на тактирование, он только изображает компонент схемы, что важно при разработке печатной платы.

В строке «Advanced Properties» находится выпадающий перечень параметров, состав которых зависит от модели МК:

- «Watchdog Clock» – частота сторожевого таймера,
- «Simulate Power on Reset Delay» – моделировать задержку (рестарта) по включению питания (16.5 мс для данной модели),
- «EEPROM Write Delay» – моделировать задержку записи в EEPROM,
- «Initial contents of EEPROM» – подключение файла загрузки в EEPROM.

Выводы напряжения питания МК скрыты (*hidden*) и подключены к клеммам GND = 0V и VCC = +5V.

Частота тактирования задается в свойствах (*Clock Frequency*), подача сигнала тактирования на входы подключения кварцевого резонатора (*XTAL1, XTAL2*) не влияет на частоту работы МК.

Кнопка «Ok» сохраняет изменение свойств модели.

Кнопка «Help» вызывает окно описания моделей AVR МК.

Кнопка «Data» служит для поиска файла технического описания (Datasheets).

Кнопка «Hidden Pins» вызывает окно настройки подключения скрытых выводов питания VCC и GND, по умолчанию они подключены к системным клеммам VCC = 5 В и GND = 0 В. Для настройки системных клемм используйте команду меню Design → Configure Power Rails.

Кнопка «Cancel» закрывает окно свойств модели Edit Component без сохранения изменений.

Пример 1. Описание схемы

Схема кнопочного пульта (рис. 2.2) состоит из МК D, резисторов R1, R2, кварцевого резонатора ZQ, светодиода HL, кнопок S1, S2, системных клемм питания VCC = 5 В и 0 В (значок заземления).

Вход RESET МК подключен через резистор R1 к VCC, что обеспечивает высокий уровень для запуска работы программы. Напомним, что низкий уровень на входе переводит МК в состояние сброса и разрешает выполнение вспомогательных функций загрузки/выгрузки. В модели работает только сброс. На следующих схемах с МК вход RESET оставлен свободным, внутренняя притяжка к высокому уровню гарантирует отсутствие сброса.

Кварцевый резонатор ZQ в реальной схеме может использоваться как один из источников тактирования. В модели выводы XTAL1, XTAL2 не используются, подключение ZQ имеет смысл только для проектирования печатной платы, поэтому на следующих схемах с МК отсутствует.

Кнопки S1, S2 являются интерактивными элементами, их можно замыкать (кликом на кружке со стрелкой вниз или на самом контакте) или размыкать (аналогично), как в режиме редактирования схемы (обычное состояние), так и в процессе моделирования.

Модель резистора R2 настроена на цифровой режим (Model Type: Digital в окне свойств Edit Component), это упрощает структуру математической модели схемы и снижает расходы времени на моделирование. Такой прием удобен и полезен, когда нас интересуют только типовые состояния цифровых сигналов: высокий, низкий и пр., а численные значения напряжений и токов не имеют значения.

Пошаговая отладка в Proteus VSM.

Запускается из меню командой Debug → Start/Restart Debugging, останваливается либо этой же командой, либо Debug → Stop Animation.

Вид окна схемного редактора в режиме пошаговой отладки приведен на рис. 2.12. Возле каждого вывода компонента появляется цветной квадратик, красный – высокий уровень, синий – низкий, серый – уровень не определен (на данном рисунке нет).

Состав окон наблюдения ресурсов МК выбирается пользователем в нижней части меню Debug. Размеры и положение окон также задаются пользователем обычным способом.

Меню Debug содержит стандартный набор команд выполнения программы Step…, Execute…, Animate. Последняя похожа на команду AutoStep среды AVR-Studio.

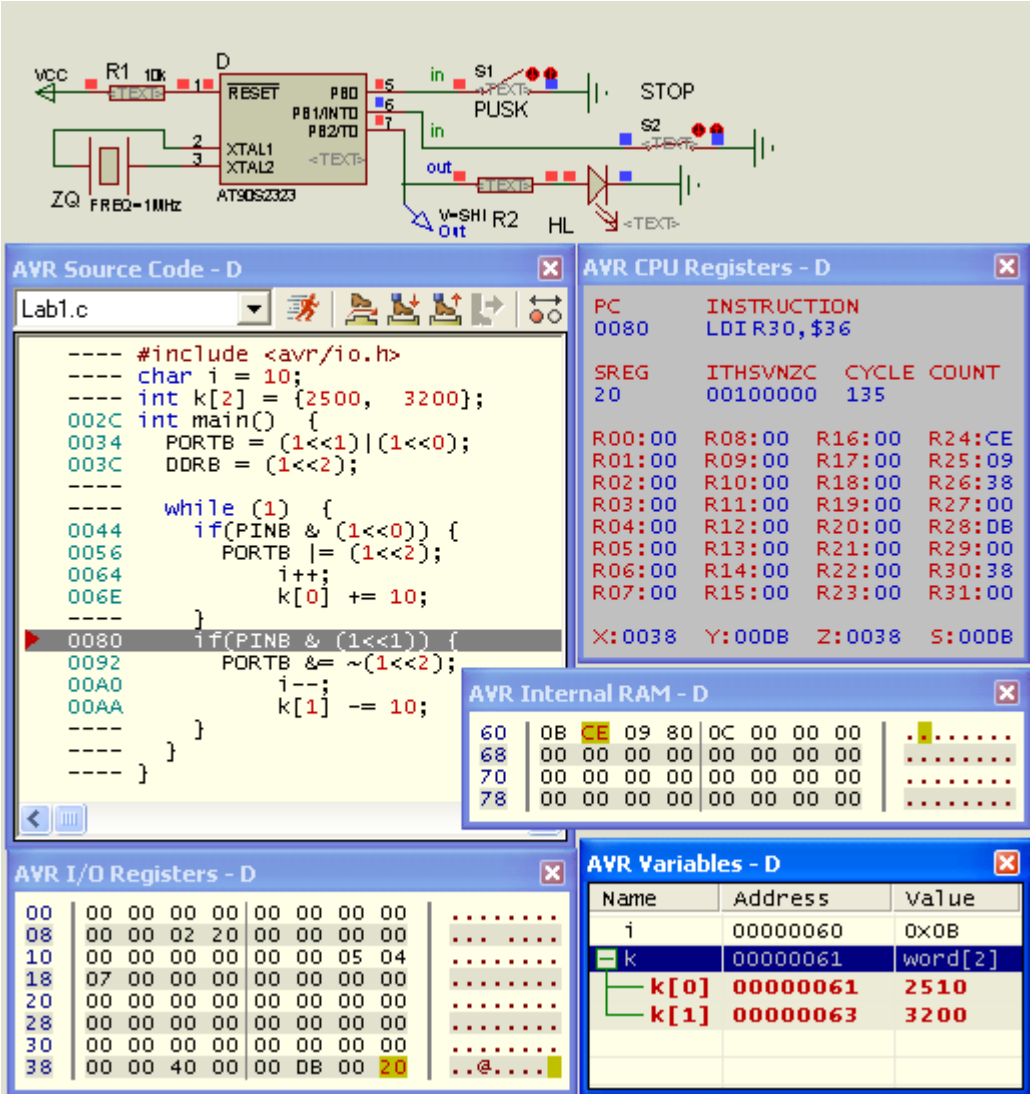


Рис. 2.12

В окне «AVR Source Code» можно наблюдать исходный текст программы на языке C/C++, в колонке слева – адрес машинной команды (Hex), следующая выполняемая строка выделена серым цветом с красной стрелкой. Для наблюдения текста программы файлы проекта Proteus VSM <*.dsn> и файл исходного текста должны находиться в одном каталоге.

Всплывающее меню окна позволяет менять состав и формат отображения программы, а также управлять точками прерывания (Breakpoints).

Имя D в окнах отладчика – позиционное обозначение МК.

В окне «AVR CPU Registers»: PC – указатель команды, INSTRUCTION – мнемоника следующей команды, SREG и ITHSVNZC – регистр состояния и его биты, CYCLE COUNT – счетчик машинных циклов, R00…R31 – РОН, X, Y, Z – регистры- указатели, S – указатель стека.

В окне «AVR Variables» собраны все переменные программы.

В окне «AVR I/O Registers» представлены РВВ (РСФ) без символьного представления имен, только адреса (Hex).

Всплывающее контекстное меню позволяет менять формат представления данных, копировать данные, но не позволяет изменять данные.

При перестроении загрузочного модуля (*.elf) в среде AVR-Studio он автоматически подключится для отладки только после рестарта процесса отладки (Start / Restart Debugging). Это справедливо и для остальных способов отладки.

Отладка в режиме анимации

Похожа на пошаговую отладку с командой Animate, но без просмотра ресурсов МК. Запускается с пульта (рис. 2. 13) левой кнопкой (стрелка вправо), после этого кнопка зеленеет, в строке состояния появляется надпись Animating, рядом счетчик модельного времени (на рис. 2.13 1 с 9704 мкс). Останавливается по правой кнопке (черный квадрат), две другие кнопки пульта простанавливают и продолжают анимацию. Они активны и в пошаговом режиме.

Главное «удовольствие» режима анимации – использование элементов с интерактивными и анимационными возможностями и наблюдение поведения схемы в режиме «реального времени». В качестве органов управления могут использоваться разнообразные кнопки, переключатели, потенциометры и др. В качестве «показывающих» устройств могут выступать как разнообразные элементы индикации – лампочки, светодиоды, многсегментные светодиодные индикаторы, жидкокристаллические индикаторы, так и спциальные анимационные модели управляемых устройств – электродвигатели, вентиляторы, для профессионалов есть даже ряд виртуальных измерительных инструментов – осциллограф, вольтметр, амперметр и др.

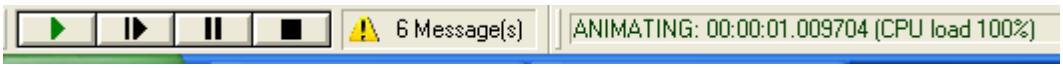


Рис. 2.13

Для отладки нашей схемы (верхняя часть рис. 2.12) достаточно двух интерактивных кнопок (S1, S2) и цветовых индикаторов состояния выводов. Предполагается, что кнопки S1, S2 подпружинены в замкнутое состояние – «нормально замкнутые». Если выполнить запуск программы в таком положении, то на входах PB0, PB1 синие квадратики (низкий уровень), выход PB2 – в низком состоянии.

При размыкании кнопки S1 (PUSK) уровень на входе PB0 становится высоким, следом программа устанавливает высокий уровень на выходе PB2. Замыкание кнопки S1 возвращает низкий уровень на вход PB0, но выход PB2 остается в высоком состоянии.

Аналогично, при размыкании кнопки S2 (STOP) уровень на входе PB1 становится высоким, следом программа устанавливает низкий уровень на выходе PB2. Замыкание кнопки S2 возвращает низкий уровень на вход PB1, но выход PB2 остается в низком состоянии.

Если же разомкнуть обе кнопки сразу получим «запрещенное» состояние для данной программы, выход PB2 начнет переключаться. В режиме анимации эти переключения заметны, но достоверно оценить их частоту нвозможно. Удобнее это сделать с использованием окна просмотра временной диаграммы.

Отладка с окнами временных диаграмм.

Сначала надо выбрать точки наблюдения на схеме и разместить в них Voltage Probe (датчик напряжения) или Current Probe (датчик тока) (рис. 2.9, б). На рис. 2.12 к выводу PB2 подключен датчик напряжения Out (синяя стрелка).

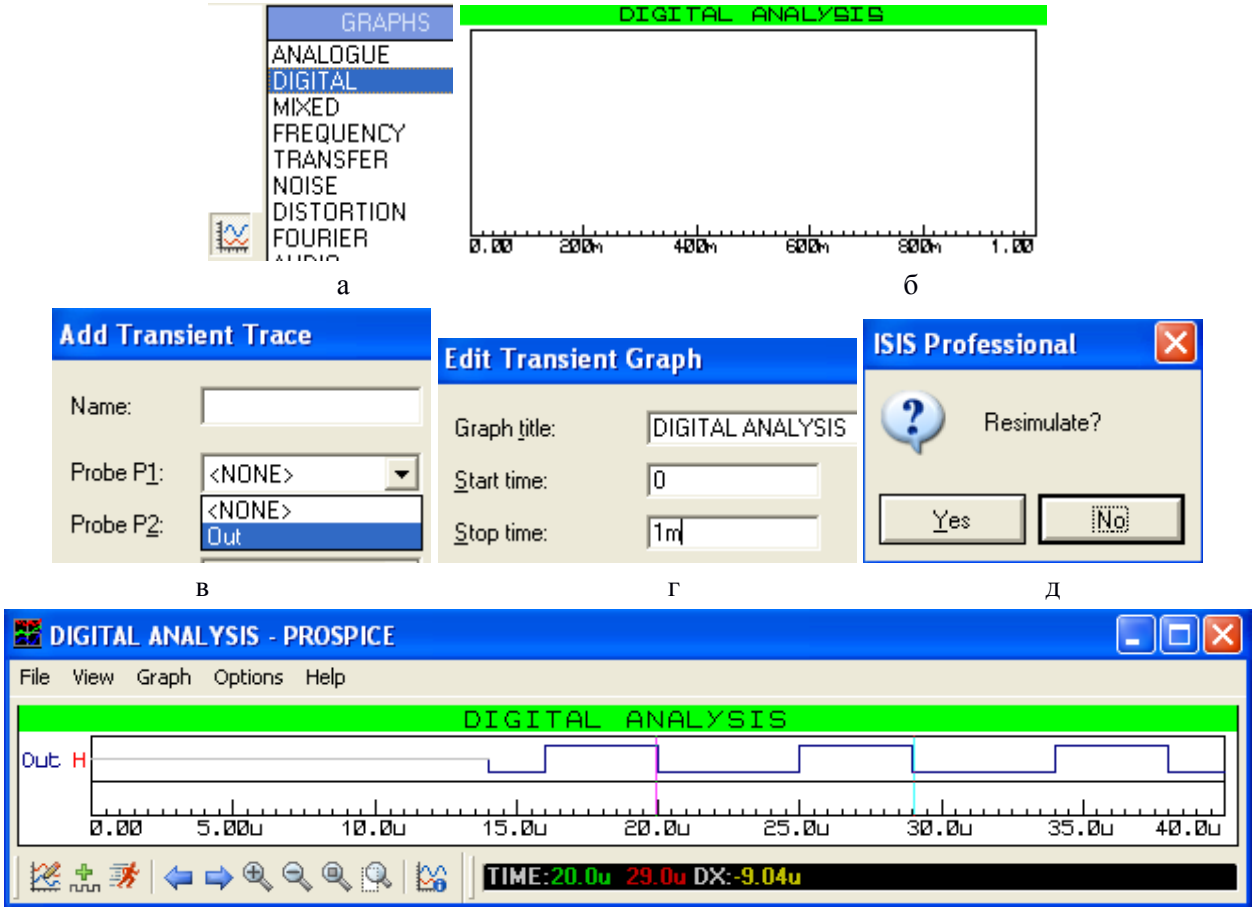
Следующим шагом надо нажать пиктограмму Gtaph Mode (рис. 2.13, а), в окне

выбора выбрать требуемый тип временной диаграммы (ANALOGUE – аналоговый, DIGITAL – цифровой, MIXED – смешанный и пр.), затем «нарисовать» мышью контур окна диаграммы (рис. 2.13, б). При выборе DGITAL различают только состояния High – высокий, Low – низкий и Float – плавающий, неопределенный; каждая переменная занимает свою строку. При выборе ANALOGUE графики отображают численные значения напряжения, тока или результат их постобработки, например, умножение двух графиков; в этом окне по умолчанию все графики имеют общую шкалу ординат.

Для создания перечня сигналов выполните команду меню Graph → Add Trace, в открывшемся окне Add Transient Trace откройте выпадающий список Probe P1 и выберите требуемое имя сигнала (на рис. 2.13, в список содержит только Out), подтвердите выбор кнпокой Ok. Для добавления следующего сигнала повторите действия по выбору.

Настройка параметров графиков и самого процесса расчета выполняется в окне Edit Transient Graph (рис. 2.13, г), вызываемом командой меню Graph → Edit Graph. Обычно требуется изменить длительность расчета (Stop time), в нашем случае – 1 мс.

После подтверждения выбора кнопкой Ok, в ответ появиться окошко запроса Resimulate? (то есть запрос на перестройку графика, рис. 2.13, д), следует согласиться. Для повторного построения графика используйте команду Simulate Graph из контекстного всплывающего меню данного окна графиков.



е

Рис. 2.13

Для подробного наблюдения деталей графика удобно перейти в режим Maximize (Show Window), доступен во всплывающем контектсном меню окна Digital Analysis. На рис. 2.13, е приведен вид окна с расширенными возможностями: меню, пиктограммы, два

курсора и численные параметры курсоров. Для изменения настроек цветов окна использована команда меню Options → Set Graph Colors… Первый курсор перемещается мышкой или клавишами стрелок (по точкам изменения состояния цифрового сигнала). Второй курсор перемещается мышкой при нажатой клавише <Ctrl>. В строке состояния после Time: отображаются численные значения временного положения курсоров, после DX: их разность, справа отображаются уровни, соответствующие курсорам (для аналогового сигнала еще и разность уровней).

Рассмотрим внимательно временную диаграмму для программы без переменных (рис. 2.13, е). Первые 14 мкс выполняется программная инициализация, выход PB2 находится состоянии воскоомного входа Flt. Затем, после команды перевода вывода в состояние выхода устанавливается низкий уровень Low. Через 2 мкс выполняется первая команды установки в цикле (уровень High), через 4 мкс – обнуление в цикле, через 5 мкс – следующая установка (во втором цикле). Дальше все циклически повторяется: период 9 мкс, длительность высокого состояния – 4 мкс.

Для варианты программы с переменными (i, k[2]) инициализация (состояние FLT) выросла до 101 мкс, через 19 мкс переключение первого цикла, период цикла – 71 мкс, длительность высокого состояния – 36 мкс.

При отладке одной схемы можно создать множество окон графиков с разным типом, составом переменных и временными интервалами. При запуске «расчета» каждого окна в реальности моделируется работа всей схемы, но отображается только выбранный состав переменных.

2.5. Средства загрузки кодов программ и данных (программаторы)

Для размещения кодов программы и данных (констант) во встроенной энергонезависимой памяти МК/МП используются разные технологии и устройства.

Самый дешевый способ основан на применении масочно программируемого ПЗУ, код «прошивается» на этапе производства, для изменения программы требуется перепроектирование кристалла, что ограничивает этот способ только устройствами с большим тиражом. Следующими по стоимости идут микросхемы с однократно программируемым ПЗУ.

Самыми гибкими являются микросхемы с многократно стираемым и программируемым ПЗУ. Число гарантированных циклов записи/стирания от 10000 и выше, это позволяет вести процесс разработки и поддержки программы без замены кристалла. Сегодня многократно программируемые ПЗУ базируются на технологиях электрического стирания и записи (программирования) памяти, такие узлы именуются *EEPROM* и *FlashROM*. Разница между двумя последними сводится к наличию индвидуального у каждой ячейки электрода стирания (*EEPROM*) или общего на блок ячеек размером от 256 и более байт (*FlashROM*). Эта экономия позволяет значительно повысить плотность размещения, следовательно, и количество ячеек на кристалле.

Еще одним широко распространенным атрибутом стало наличие в кристалле узла загрузки/стирания/выгрузки, управляемого извне через один или несколько последовательных интерфейсов. Это позволяет выполнять процедуры редактирования программы/данных непосредственно на целевой плате, что значительно упрощает

разработку и отладку программного обеспечения, поддержку развития программы на этапе эксплуатации платы.

Этот же механизм поддерживает редактирование небольшой области энергонезависимой памяти, называемой *Fuse* и *Lock* битами. *Fuse* биты (или биты конфигурации) служат для настройки свойств МК, таких как выбор источника тактирования, длительность процесса рестарта, запрет работы сторожевого таймера и пр. *Lock* биты (или биты защиты) управляют доступом к ранее загруженным кодам программы и данных – защита от чтения и от стирания.

Для проверки связи и автоматической настройки каждая модель МК оснащена уникальным номером (signature bytes).

МК серии *AVR* уже на этапе разработки первого семейства (Classic) были оснащены механизмом программирования «в системе» (*In System Programming – ISP*), обеспечивающим чтение и запись областей FlashROM, EEPROM, Fuse, Lock, чтение Signature.

Технология *ISP* использует один вывод кристалла для перевода в режим программирования (вход *RESET* в низком состоянии) и три вывода собственно для подачи специальных команд, загрузки данных и выгрузки данных. Эти три вывода работают по алгоритму последовательного синхронного интерфейса *SPI* в режиме ведомого. Один вывод служит для подачи к МК тактов (вход *SCK*), второй для подачи команд/данных (вход *MOSI / PDI / DI)*, третий – для чтения данных (выход *MISO / PDO / DO)* – см. рис. 1.3, подробное описание логики SPI интерфейса в разделе 7.3. Как правило, все эти функции выводов альтернативные, то есть активны только в режиме сброса-программирования, в режиме выполнения программы они могут использоваться как выводы портов и пр.

В качестве устройства, управляющего процессом загрузки/выгрузки, используется персональный компьютер. Для этого используется специальный кабель-загрузчик (*download cable*), один конец которого подключается к одному из интерфейсных разъемов компьютера (LPT, COM, USB), другой – к разъему загрузки/выгрузки на плате с целевым МК. Чаще всего загрузочный кабель оснащен встроенным преобразователем интерфейса.

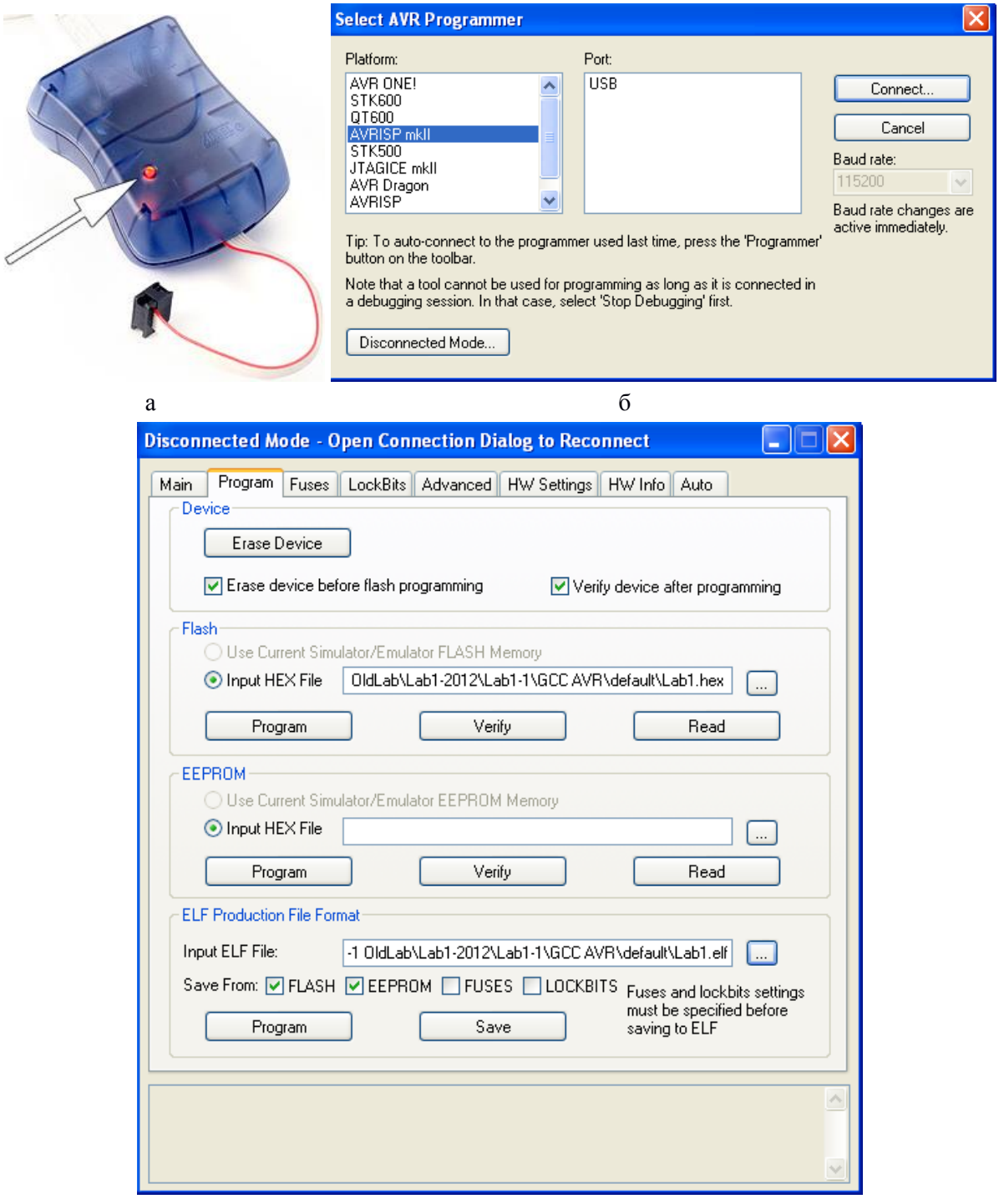
Программное управление процессом загрузки/выгрузки выполняет программа-загрузчик, работающая автономно или в составе пакета (среды) проектирования программ.

Для загрузки/выгрузки МК серии AVR используются среда *AVRStudio (Tools → Program_AVR), CodeVisionAVR (Tools → Chip_Programmer)* и пр., либо автономные программы – *Kanda_System, PonyProg, Avreal* и пр. Они поддерживают работу через различные кабели загрузки.

На рис. 2.14, а прведена фотография программатора AVRISP mkII. Это печатная плата с МК в полупрозрачном пластиковом корпусе с тремя дагностическими светодиодами. На фото виден плоский кабель с 6-контактным разъемом (розетка IDC-6) для подключения к вилке на целевой плате. С другой стороны корпуса находится розетка для подключения USB-кабеля, он используется для интерфейса и питания устройства.

Вызов программатора в среде AVR-Studio выполняется командой меню Tools → Programm AVR → Connect. В появившемся окне Select AVR Programmer (рис. 2.14, б) в окошке Platform представлен набор поддерживаемых устройств, в окошке Port уточняется

интерфейс с компьютером, кнопка Connect (или Disconnect Mode) устанавливает связь с выбранным устройством (или выбирает режим работы без программатора для знакомства).



а б

в

Рис. 2.14

Многостраничное окно (рис. 2.14, в) представлено в виде закладки Program. Рамка Device содержит кнопку Erase Device (стирание FlashROM), настройки стирания и верификации (проверка результата записи). Рамка Flash (память программ) содержит окно выбора файла прошивки с машинным кодом и кнопки для выполнения операций загрузки Program, верификации и чтения Read (перед чтением всплывает запрос имени файла для выгрузки машинного кода). Рамка EEPROM (энергонезависимая память данных) имеет

аналогичные функции. Рамка ELF Production File Format выполняет загрузку/выгрузку всех доступных ячеек МК (ПП, ЭПД, биты настроек и биты защиты) с использованием файлов в формате ELF. Нижняя рамка используестя для вывода сообщений о процессах загрузки/выгрузки.

Закладка Main позволяет выбрать тип МК, тип загрузчика, прочитать сигнатуру типа МК. Закладка Fuses позволяет детально просмотреть и редактировать биты настроек. Закладка LockBits позволяет детально просмотреть и редактировать биты защиты. Закладка Advanced позволяет подстроить частоту тактирования (если доступно). Закладки HW Settings и HW Info предназначены для настройки и просмотра устройств загрузки (напряжения, тактирование, прошивка). Закладка Auto позволяет настроить и автоматизировать процесс загрузки/выгрузки (выбор состава операций и логгирование).

2.6. Подключение индикаторов и клавиатуры

Индикаторы и органы управления, расположенные на целевой плате или устройстве сМК являются хорошим средством отладки. Индикаторы позволяют отслеживать выполнение определенных функций, состояние переменных. Органы управления, такие как кнопки, переключатели, клавиатура обычно размещаются на пультах управления для штатного включения/выключения, выбора режима и вариации уставок.

В процессе отладки те и другие «временно» можно нагружать дополнительными функциями для тестирования отлаживаемых функций.

Светодиоды

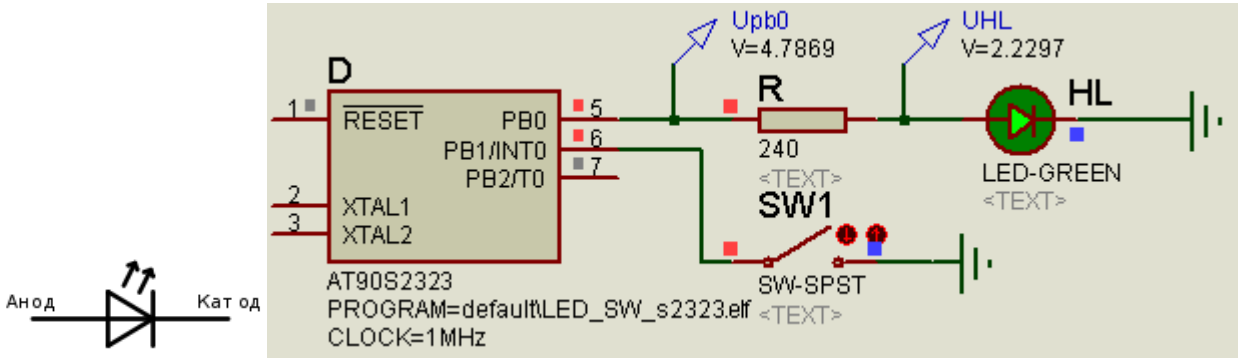
Для начала рассмотрим схемотехнику подключения отдельных светодиодов, схемотехника подключения переключателя (кнопки) была рассмотрена в подразделе «Параллельный 8-разрядный порт» п. 1.6.

На рис. 2.15, а приведено схемное обозначение светоизлучающего диода (СИД), к аноду прикладывается положительный потенциал, к катоду – отрицательный (прямое включение pn-перехода). Вольтамперная характеристика светодиодов в прямом направлении нелинейна. Диод начинает проводить ток, начиная с некоторого порогового напряжения (1.6 … 2.7 В). Ток свечения сильно зависит от конструкции, для малогабаритных приборов он составляет единицы - десятки миллиампер.

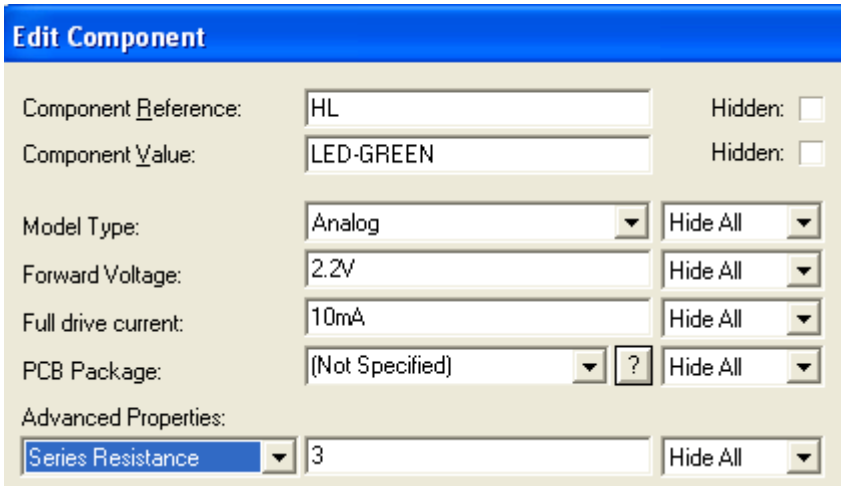
При напряжении питания 5 В номинальная нагрузочная способность одного выхода порта составляет 20 мА, что вполне достаточно для большинства индикаторных СИД. Рекомендуется подключать светодиод через резистор сопротивлением в несколько десятков или сотен Ом, это позволяет задать требуемый ток, то есть яркость свечения.

На рис. 2.15, б приведена схема подключения светодиода между выходом (анод) и нулем питания (GND) (катод), в этом случае высокий уровень на выходе «зажигает» светодиод, низкий – «гасит». Так же можно подключить светодиод между выходом (катод) и плюсом питания VCC (анод), в этом случае «зажигает» низкий уровень. В обоих случаях последовательно со светодиодом должен быть токоограничивающий резистор.

Для расчета сопротивления резистора желательно знать требуемый ток светодиода Iсид и примерное значение падения напряжения Uсид при этом токе. Тогда, с учетом падения напряжения на выходе порта МК сопротивление резистора R = (VCC – Uсид) / Iсид – r_{on1}, где r_{on1} = 40 Ом – внутреннее сопротивление выхода (см. п. 1.6).



а б



в

Рис. 2.15

Здесь (рис. 2.15, б) и далее большинство электрических схем нарисованы в редакторе схем ISIS системы сквозного проектирования Proteus. На рис. 2.15, в прведен вид окна свойств зеленого светодиода LED-GREEN из системы моделирования Proteus VSM. Аналоговая модель светодиода (выбор Analog в строке Model Type) имеет нелинейную вольтамперную характеристику. Рабочий участок прямой ветви ВАХ характеризуется параметрами: порогового напряжения Forvard Voltage U_0 = 2.2 В и дифференциального сопротивления Series Resistance $r_д$ = 3 Ом, с некоторой погрешностью зависимость тока от напряжений описывается уравнением $U_{HL} = U_0 + I * r_д$. Параметр Full drive current: 10 мА характеризует анимационные свойства модели: при токе от нуля до указанного значения яркость «свечения» растет линейно, далее – не изменяется.

Для выбора резистора следует решить систему уравнений. Первое из предыдущего абзаца. Второе описывает цепь, состоящую из блока питания (напряжение VCC), выхода МК в высоком состоянии (падение напряжения dU_{OH}), резистора и светодиода (падение напряжения U_{HL}) $VCC = U_{HL} + dU_{OH} + I*R$

При наличии системы моделирования можно обойтись без расчетов – наблюдая напряжения в схеме (синие стрелки Upb0, U_{HL} и значения напряжения V=… рядом с ними на рис. 2.15, б), по ним вычислить ток, либо непосредственно измерить ток подобным пробником. Варьируя сопротивление резистора подобрать требуемый ток.

По данным на рис. 2.15, б рассчитаем ток I = (4.79 – 2.23) / 240 = 2.56 / 240 = 0.0107A = 10.7 мА – это значение почти совпалает с оптимальным для данной модели. Можно уточнить сопротивление верхнего транзистора на выходе порта МК при данном токе r_{on1} =

(5 – 4.79) / 0.0107 = 19.6 Ом.

По поводу моделей светодиода и МК следует сделать важное замечание. При выборе в свойствах модели светодиода Model Type: Digital (цифровая модель) мы теряем возможность регулирования яркости, но приобретаем два преимущества. Во-первых, не требуется токоограничивающий резистор, либо в свойствах резистора также можно выбрать цифровую модель. Во-вторых, отказ от аналоговых моделей резко сокращает объем вычислений при моделировании и повышает так называемую сходимость шагов расчета. В простых примерах с малым временем расчета это не существенно, но при разрастании схемы и длительном сеансе моделирования это весьма заметно.

В любой системе моделирования сложность модели должна соотвествовать задачам, стоящим при исследовании объекта. В применении к моделированию электрических схем всегда следует отдавать предпочтение цифровой модели перед аналоговой, при условии, что задачи анализа не требуют различать уровни или форму аналоговых сигналов.

Ниже приведен текст простейшей программы для МК AT90S2323, практического смыслы в ней нет (переключатель сам может коммутировать цепь питания светодиода), но здесь показаны требуемые настройки и иллюстрируется принцип программного управления.

```
// Управление светодиодом между выходом и нулем по переключателю
#include <avr/io.h>
int main()  {
  DDRB = (1<<PB0);//Выход для управления светодиодом
  PORTB = (1<<PB1);//Включить притяжку к VCC входа кнопки
  while (1)  {
    if(PINB & (1<<PB1)) //Если вход PB1 = 1 (перключатель разомкнут),
   PORTB |= (1<<PB0);// то зажечь светодиод,
    else PORTB &= ~(1<<PB0); // иначе погасить
  }
}
```

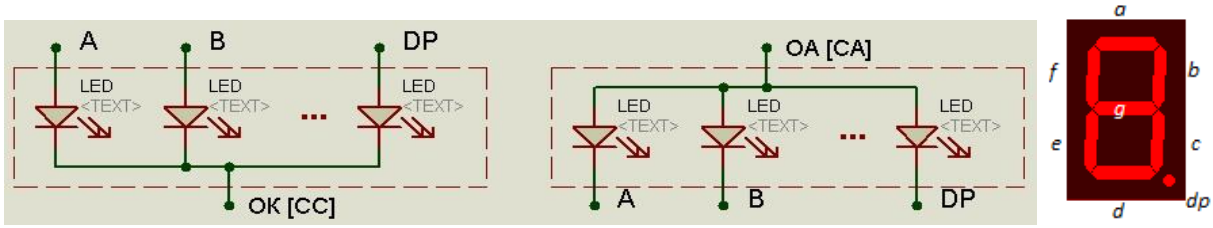
Весь проект находится в папке Primer \ p1-2 Human Mashine Interface \ LED SW, он состоит из файлов LED_SW_s2323.c – исходный текст программы на языке C/C++ (AVR_GCC), LED_SW_s2323.aps – файл проекта программы и сеанса отладки в среде AVR_Studio, LED_SW_s2323.dsn – электрическая схема и модель, соданная в среде Proteus и использующая загрузочный модуль программы, созданный транслятором AVR_GCC.Более сложный пример программного обслуживания – в лабораторной работе № 1, а также в папке Primer \ p1-1 OldLab.

Далее в этом разделе мы познакомимся с устройством, схемами подключения и алгоритмами управления сегментных светодиодных индикаторов и клавиатур. Рассмотренные ниже примеры находятся в папке Primer \ p1-2 Human Machine Interface.

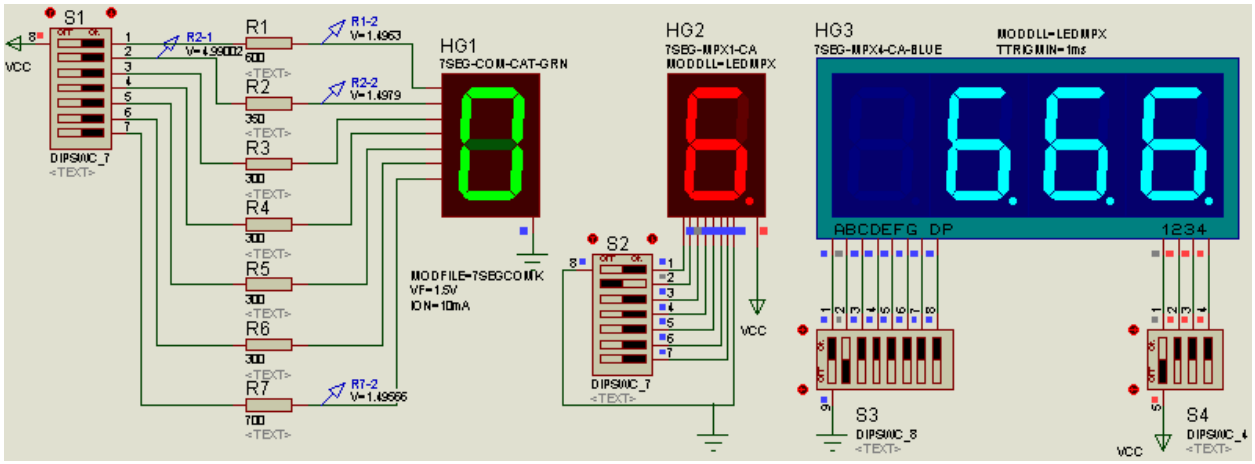
Сегментные индикаторы

Наиболее распространены 7-сегментные индикаторы (рис. 2.16). Они состоят из семи сегментов *a*…*h*, предназначеных для индикации цифр, ряда символов латинского / кириллического алфавита и сегмента десятичной точки (*dp*). Каждый сегмент представляет собой светодиод с номинальным током свечения 10…20 мА, что позволяет присоединять их к выводам МК напрямую (не забудем токоограничивающие резисторы).

Для экономии выводов микросхемы аноды или катоды делают общим для всех 8 светодиодов, что отражается в названии микросхемы: ОА *[CA – Common Anode]* или ОК *[CC – Common Cathode]*. На рис. 2.16, а приведены схемные изображения, на рис. 2.16, б – схема именования сегментов 7-сегментного индикатора.



а б



в

Рис. 2.16

Система моделирования *Poteus_VSM* имеет в библиотеке *Optoelectronics* широкий набор различных, в том числе, 7-сегментных индикаторов (раздел *7-Segment Displays*), которые в интерактивном режиме формируют программно управляемое изображение. В файле 7segLEDmodel_sw.dsn (рис. 2.16, в) представлены схемы включения моделей трех типов:

- модель индикатора на одну цифру по схеме ОК (HG1 7SEGM-COM-CAT-GRN) является аналоговой, ее параметры (Forward Voltage: 1.5 V, Segment On Current: 10 mA) определяют ВАХ светодиодов; ток сегмента задается выбором резистора, сегмент светится при токе не менее 5 мА, при росте тока интенсивность цвета не меняется;

- модель индикатора на одну цифру по схеме ОА (HG2 7SEG-MPX1-CA) является цифровой, ее параметр (Minimum Trigger Time: 1 ms) важен только при использовании алгоритма динамической индикации, о котором речь пойдет ниже.

- модель индикатора на четыре цифры по схеме ОА (HG3 7SEG-MPX4-CA-BLUE) цифровая и идентична предыдущей по свойствам; на примере этой модели видно, что невозможно в статике сформировать разное изображение для цифр.

Интерактивные переключатели S1…S4 позволяют подавать напряжение и ток на сегменты индикаторов. Меняя положение переключателей можно зажигать отдельные сегменты. Далее роль переключателей будут выполнять выходы МК.

На рис. 2.17 показано подключение двух 7-сегментных индикаторов с ОА, *HG1* подключен к порту *A*, *HG2* – к порту B, используется простая статическая индикация. Для индикации цифры необходимо преобразовать двоичный код числа в код, соответствующий изображению цифры. В МК это удобно делать программно табличным способом, при этом появляется свобода выбора выводов порта для управления отдельными сегментами.

Ниже приведен текст программы циклического вывода двузначных десятичных чисел от 0 до 99, другие числа в этом примере индицируются как «9.9.». При делении числа на 10 остаток от деления (операция языка *C «%»)* представляет собой младшую цифру, частное – старшую. Для зажигания сегмента требуется низкий уровень, поэтому перед выводом требуется побитовая инверсия (операция ~). Как видно, алгоритм вывода достаточно прост и требует очень мало процессорного времени (выделение цифр в числе и преобразование в код). Тепм вывода определяется только длительностью программной задержки `_delay_ms(200),` равной 200 мс, при верно указанной частоте тактирования процессора транслятору и симулятору.

Весь пример находится в папке led7-2_m16 и в одноименных файлах протеуса DSN и AVR_Studio aps.

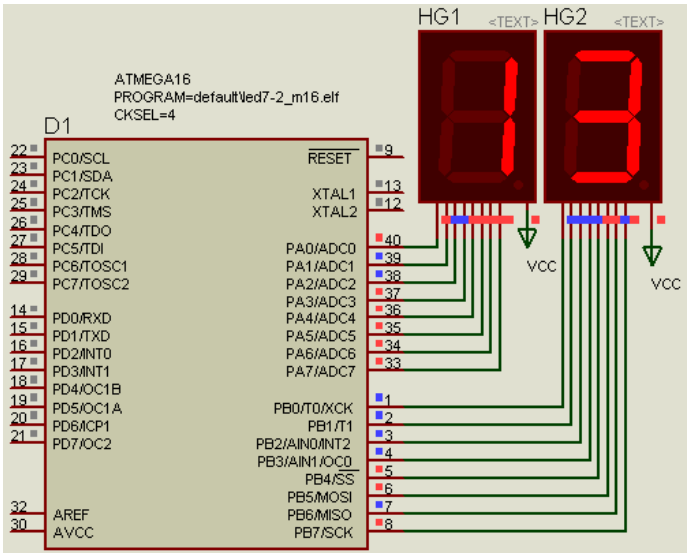


Рис. 2.17

```
#include <avr/io.h>
#include <avr/delay.h>

unsigned char n=0;
#define DP 0x80
//0    1    2    3    4    5    6    7    8    9    A    b    C    d    E    F
char dig7[] = {
0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x27,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71
};

int main()  {
  unsigned char dig0, dig1;
  DDRA = 0xff;
  DDRB = 0xff;
  while (1)  {
    _delay_ms(200);
    if(n<100) n++; else n=0;
    if(n > 99) {
        PORTA = ~(dig7[9] | DP);
        PORTB = ~(dig7[9] | DP);
      } else {
        dig1 = n / 10;
        dig0 = n % 10;
```

```
        PORTA = ~dig7[dig1];
        PORTB = ~dig7[dig0];
    }
  }
}
```

Использование простой статической индикации при значительном количестве цифр (4 и более) наталкивается на две проблемы.

Первая заключается в неэкономном использовании выводов МК: каждая цифра требует отдельного 8-битного порта, то есть для вывода N цифр необходимо N портов МК.

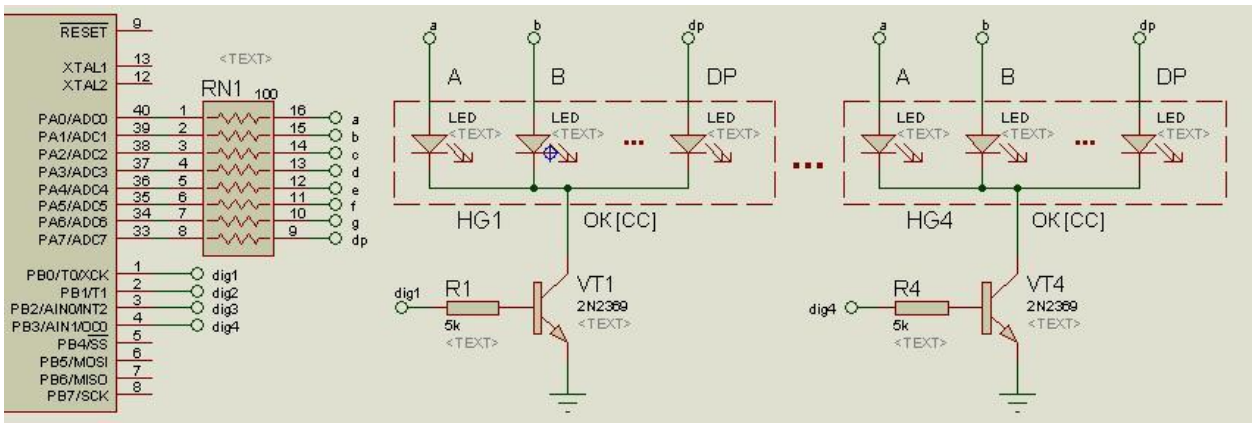
При использовании 7-сегментного индикатора из нескольких цифр (*HG3* на схеме рис. 2.16, в) выводы одинаковых сегментов объединяют в параллель. Это обеспечивает решение первой проблемы, теперь для вывода N цифр необходим один 8-разрядный порт плюс N отдельных выводов МК Это называется матричным подключением. Но теперь в статике невозможно получить изображение разных цифр на соседних знакоместах.

Алгоритм динамической индикации сначала формирует изображение одной цифры, в это время другие погашены. На следующем интервале времени активной становится другая цифра и т.д. При частоте мерцания отдельной цифры выше 50…100 Гц (Ти = 20…10 мс) человеческий взгляд его не замечает. Например, для индикатора из 8 цифр длительность включения одной цифры составит tи = Tи/8 = 1,25…2,5 мс. При этом требуется повышенная яркость сегмента (менее, чем пропорционально), т.е. увеличение значения тока сегмента. Если требуемое значение тока больше, чем допустимое для данного МК, используют ключ на биполярном, реже полевом транзисторе.

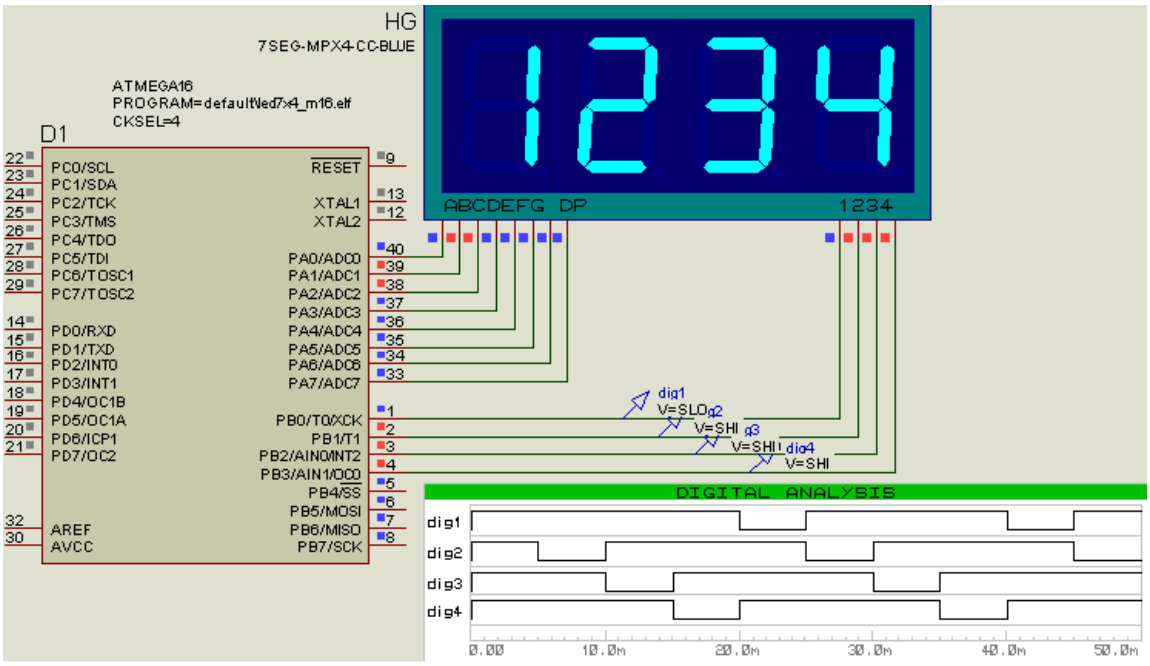
На рис. 2.18, а приведена типовая схема для подключения 7-сегментного индикатора на 4 цифры с общим катодом (папка led7x4_m16). Для коммутации тока 8 включенных светодиодов (при токе одного светодиода 10…20 мА, суммарный ток 80…160 мА) требуются транзисторный ключ (VT1…4, R1…R4), для зажигания светодиода на базу транзистора следует подать положительное напряжение, то есть уровень «1». Такая схема требует использования алгоритма динамической индикации.

Динамическая индикация состоит в периодическом включении (активации) одного из четырех индикаторов, остальные в это время погашены (временная диаграмма на рис. 2.18, б). При частоте динамической индикации не ниже 50 Гц человеческий глаз не замечает мерцания отдельных цифр. Для нашего случая выберем частоту 50 Гц, тогда за период 20 мс один из четырех индикаторов должен светить 20/4 = 5 мс.

На рис. 2.18, б приведен вариант схемы динамической индикации для моделирования в ProteusVSM, в котором используется цифровая модель HG (7SEG-MPX4-CC-BLUE) для индикации. Для коммутации ее катодных выводов 1…4 достаточно обычного выхода МК, поэтому для выбора индикатора требуется уровень «0» (в отличии от схемы рис. 2.18, а), остальные выходы выбора при этом должны быть в «1». Вывод кода символа выполняется в прямом коде. Ниже приведен текст программы для тестирования схемы и алгоритма, на индикатор выводится содержимое массива цифр dig[4] = {1,2,3,4}.



а



б

Рис. 2.18

```
#include <avr/io.h>
#include <avr/delay.h>

unsigned char n=0;
#define DIG1 (1<<0)
#define DIG2 (1<<1)
#define DIG3 (1<<2)
#define DIG4 (1<<3)
#define DP 0x80
unsigned char dig7[] = {
0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x27,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,
0x71 };
unsigned char dig[4] = {1,2,3,4};

void dynamic_indicate(void) {
static unsigned char sel_dig=0;
  if(sel_dig < 4) sel_dig++; else sel_dig = 0;
  switch(sel_dig) {
    case 0: PORTB |= DIG4; PORTB &= ~DIG1;
```

```
      PORTA = dig7[dig[0]];//
      break;
    case 1: PORTB |= DIG1; PORTB &= ~DIG2;
      PORTA = dig7[dig[1]];//
      break;
    case 2: PORTB |= DIG2; PORTB &= ~DIG3;
      PORTA = dig7[dig[2]];//
      break;
    case 3: PORTB |= DIG3; PORTB &= ~DIG4;
      PORTA = dig7[dig[3]];//
  }
}
int main()  {
  DDRA = 0xff;
  DDRB = 0xff; PORTB = 0x0f;
  while (1)  {
    _delay_ms(5);
      dynamic_indicate();
  }
}
```

Матричные светодиодные индикаторы (*Dot Matrix Display* в папке библиотеки Optoelectronics) имеют организацию 5*7 или 8*8 и тоже требуют динамической индикации.

Для подключения к МК одного такого индикатора требуется один порт (7 или 8 разрядов) для формирования изображения «столбца» и еще 5 или 8 разрядов для динамического выбора «столбца».

Для дисплея из 5 индикаторов с организацией 5*7 потребуется 32 выхода МК (7 – для формирования столбца и 25 = 5 * 5 для управления столбцами) и массив размером 25 байт.

При числе столбцов 25 и частоте регенерации 100 Гц частота динамической индикации составит 2500 Гц.

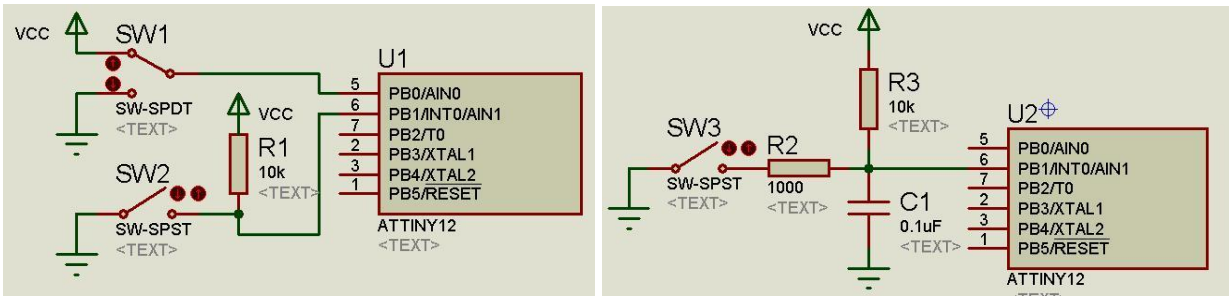
Чаще всего матричные индикаторы используются для построения бегущей строки. В этом случае требуется выделить в ОЗУ МК массив памяти размером *N* байт (*N* – число столбцов изображения) и организовать выборку с циклическим смещением по принципу *FIFO* (первым вошел, первым вышел) с интервалом 0.03…0.1 с.

Переключатели, кнопки, клавиатура

При малом количестве кнопок обычно каждую кнопку подключают к отдельному выводу МК, настроенному как вход (*DDRx.y* = 0). Схемы включения и алгоритм обслуживания аналогичны любому ключевому входу.

Если кнопка или тумблер трехполюсная (*SPDT*), то есть имеет один общий контакт и два противоположных контакта (если один замкнут, то другой разомкнут) целесообразно общий контакт подключить ко входу МК, один подключить к нулю питания (*GND*), другой – плюсу питания (*VCC*). Такая схема подключения (SW1 на рис. 2.5, а) не требует дополнительных элементов, порт МК настраивается в режим высокоомного входа (*DDRx.y* = 0, *PORTx,y* = 0), чтение состоянии через *PINx.y*.

Чаще встречаются переключатели с одной группой контактов (SPST). Обычно в этом случае ключ коммутирует на вход МК уровень «0», в выключенном состоянии уровень «1» обеспечивается резистором R (1…10 кОм) подключенным к питанию (*VCC*) либо подтягивающим резистором в схеме порта МК (*AVR: [Pull-Up]*, выбор *DDRx.y* = 0, *PORTx,y* = 1). В такой схеме (SW2 на рис. 2.19, а) разомкнутому состоянию ключа соответствует «1», замкнутому – «0».



а б

Рис. 2.19

По логике использования различают дискретные входы по уровню и по переходу – фронту или срезу. В первом случае уровень входного сигнала напрямую используется в алгоритме управления. Для выявления фронта или среза сигнала требуется не только текущее значение входа, но и предыдущее, запомненное в выделенной ячейке памяти на предыдущем цикле чтения. Такая процедура может выполняться как программно:

```
static bool pred_pinxy=0;
if((PINx & (1<<y) && !pred_pinxy) //Фронт
  { pred_pinxy=1; /* Действия по фронту */ }
else pred_pinxy=0;
```

так и аппаратно входами прерываний или счетными входами таймеров/счетчиков.

При работе с реальными переключателями и кнопками приходится учитывать так называемый дребезг контактов *[bounce]*. При замыкании и размыкании контактов переходный процесс изменения уровня напряжения сопровождается шумом, при частом чтении которого можно получить хаотичную последовательность «0» и «1». Длительность этого процесса зависит от конструкции переключателя, значения коммутируемого тока и обычно составляет доли – единицы миллисекунд. Подавление дребезга *[debounce]* выполняется схемно или программно.

Схемное решение заключается в размещении на входе МК интегрирующей RC-цепи (рис. 2.19, б), она «заваливает» фронт и срез с постоянной времени, равной произведению значений $\tau = RC = 1000$ Ом * 0.1мкФ = 0.1 мс.

Программное подавление дребезга состоит в периодическом чтении с запоминанием последних 3…5 значений с частотой в 3…5 раз выше частоты выявления фронта/среза. Фронт или срез считается выявленным, если после изменения уровня новое значение устойчиво повторилось на последних 3…5 повторных чтении. Например, при выявлении фронта в буфере на 4 значения самое старое значение должно быть «0», остальные – «1».

```
char debounce_rize(char n_bit) {
  static char b[3]={0,0,0};
  char pin, out;
  pin = PINB & (1<<n_bit);
  if(pin && b[0] && b[1] && !b[2]) out = 1;
```

```
  else out = 0;
  b[2] = b[1];
  b[1] = b[0];
  b[0] = pin;
  return out;
}
```

Для экономии выводов МК при подключении клавиатуры из более 6 кнопок выгодно использовать матричную схему, N + M выводов позволяют подключить клавиатуру из N * M кнопок. Расходы времени процессора при этом не возрастают.

Рассмотрим схему (рис. 2.20) и алгоритм обслуживания клавиатуры. Пусть матрица из 4 * 3 кнопок строчными 4 линиями подключена к выходам МК (PC0…PC3), 3 линий колонок подключены ко входам (PC4…PC6) с подтягивающими резисторами (встроенными в МК или внешними). Если на одном из выходов установить «0», а на остальных «1», то при чтении входов нажатым кнопкам будет соответствовать «0», не нажатым – «1». Таким образом, алгоритм сканирования клавиатуры сводится к установлению одной из линий выходов (PORTC.i) в «0», выдержки порядка 0.5 - 1 мкс, чтении входов (PINC.j) и записи результатов чтения в массив (kb[i]). Цикл подобных действий следует выполнить для всех выходов от 0 до 3, назовем это циклом чтения. При необходимости программного подавления дребезга цикл чтения следует выполнять в 3…5 раз чаще, чем минимальное время выявления новой клавиатурной комбинации. В этом случае понадобится двумерный массив kb[i, j].

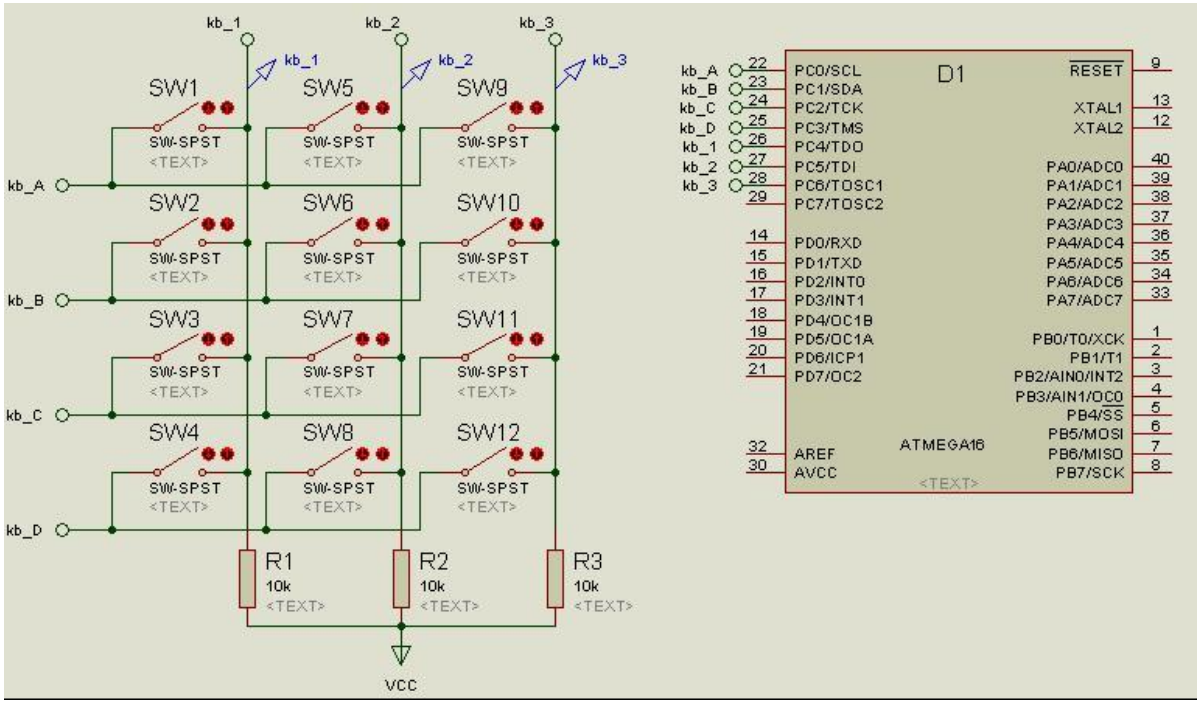


Рис. 2.20

Сложность дальнейшей обработки зависит от того, сколько вариантов нажатия клавиш следует различать. В самом простом случае процедура сканирования выявляет первую из нажатых клавиш и состояние остальных не анализирует. Сложнее различать несколько одновременно нажатых клавиш, а также формировать раздельные действия на нажатие, отпускание, и длительность удержания в нажатом положении.

Также следует различать клавиатуры мгновенного действия и варианты ввода численных или символьных данных. В последнем случае предполагается наличие хотя бы

символьного дисплея для отображения вводимой/редактируемой последовательности цифр или символов, обычно процедура ввода заканчивается нажатием клавиши «Ввод» [“Enter”]. Такие действия обычно поддержаны средствами стандартного ввода/вывода как на уровне библиотек языка программирования, так и средствами операционной системы (при ее использовании).

Пример обслуживание клавиатуры

Цифровая (телефонная) клавиатура (‘0’, ‘1’ … ‘9’, ‘*’, ‘#’) и 7-сегментный индикатор (рис. 2.21). Задача – набор кода: ввод номера начинается со «звездочки», далее ожидается ввод 4 цифр, после нажатия «решетки» последние 4 цифры используются как пароль. Варианты ответов: «*rdY* » - готовность к вводу, «*Good*» - если номер совпал (открывается замок на 10 секунд: «*ti.XX*»), «*bAd* » - если номер не совпал, «*Err* » - ошибка ввода (меньше 4-х цифр).

Процесс ввода кода: если цифра, то сместить строку влево и в младшую позицию поместить новую цифру.

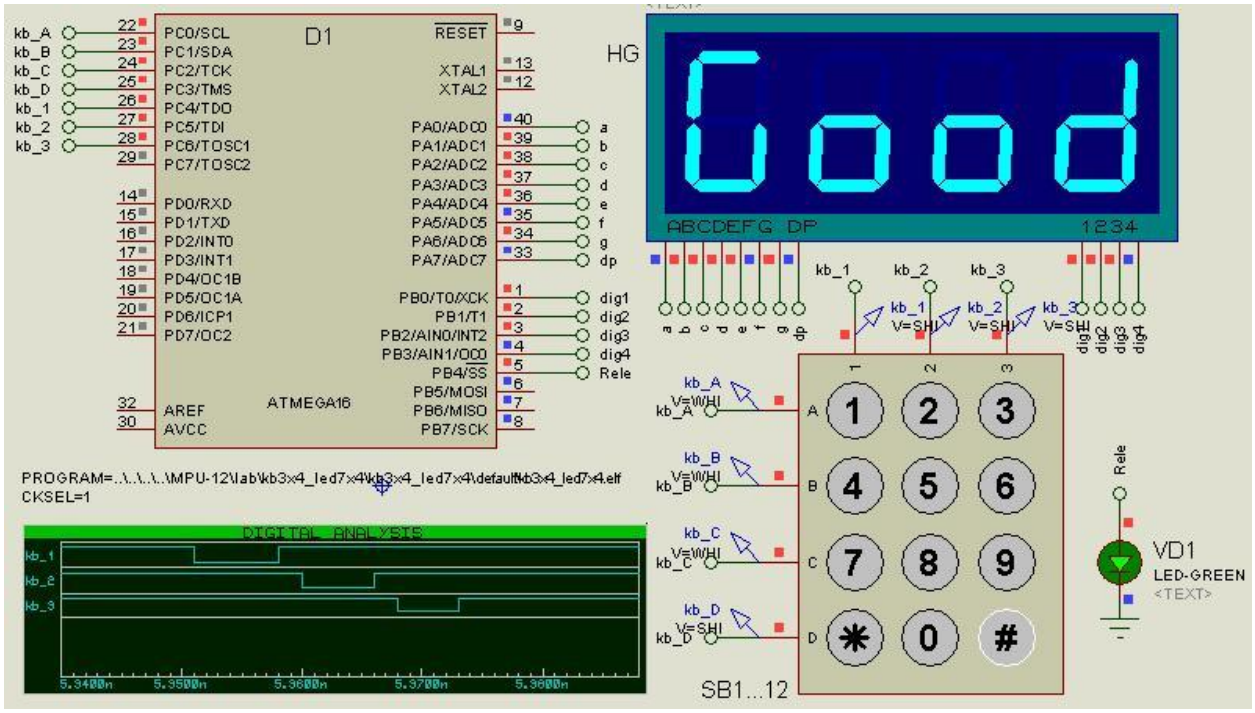


Рис. 2.21

Модель клавиатуры в Proteus’е позволяет различить только одну нажатую клавишу, причем перед нажатием следующей обязательно следует интервал без нажатия (ограничения указателя типа «мышь»). Двухуровненный обработчик клавиатуры: первый считывает состояние столбцов и выявляет код нажатой кнопки (или 0), второй определяет «историю», данном случае – момент нажатия кнопки.

В нашем простом примере действие должно происходить только по фронту, то есть по нажатию клавиши; нет действий по отпусканию и учета длительности нажатия клавиши.

```
#include <avr/io.h>
#include <avr/delay.h>

unsigned char n=0;
#define DIG1 (1<<0)
```

```
#define DIG2 (1<<1)
#define DIG3 (1<<2)
#define DIG4 (1<<3)
#define DP 0x80
#define sim_b 0b01111100
#define sim_A 0b01110111
#define sim_d 0b01011110
#define sim_E 0b01111001
#define sim_r 0b01010000
#define sim_G 0b00111101
#define sim_o 0b01011100
#define sim_Y 0b01101110
char dig7[] = {
0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x27,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
//Кодировка изображений цифр от 0 до 9, от A до F для 7-сегм. индикатора
unsigned char dig[4] = {1,2,3,4};//Массив индикатора

void dynamic_indicate(void) {//Динамическая индикация
static unsigned char sel_dig=0;
  if(sel_dig < 4) sel_dig++; else sel_dig = 0;//Выборка очередного символа
  switch(sel_dig) {
    case 0: PORTB |= DIG4; PORTB &= ~DIG1;
      PORTA = dig[3];//
      break;
    case 1: PORTB |= DIG1; PORTB &= ~DIG2;
      PORTA = dig[2];//
      break;
    case 2: PORTB |= DIG2; PORTB &= ~DIG3;
      PORTA = dig[1];//
      break;
    case 3: PORTB |= DIG3; PORTB &= ~DIG4;
      PORTA = dig[0];
  }
} //-------------------------------------------
char conv_hex_digit2dig7(char in) { return dig7[in]; }
//-------------------------------------------
char kb_read(void) {
  char rc[3]= {0,0,0}, out=0;

  PORTC &= ~(1<<4);      //Выборка и
  _delay_us(1);
  rc[0] = (~PINC) & 0x0f;//чтение первого толбца
  PORTC |= (1<<4);

  PORTC &= ~(1<<5);      //Выборка и
  _delay_us(1);
  rc[1] = (~PINC) & 0x0f; //чтение второго толбца
  PORTC |= (1<<5);

  PORTC &= ~(1<<6);      //Выборка и
  _delay_us(1);
  rc[2] = (~PINC) & 0x0f; //чтение третьего толбца
  PORTC |= (1<<6);
```

```
  switch(rc[0]) { //Выявить единств кнопку в первом столбце
    case 1: out = '1'; break;
    case 2: out = '4'; break;
    case 4: out = '7'; break;
    case 8: out = '*'; break;
  }
  switch(rc[1]) { //Выявить единств кнопку во втором столбце
    case 1: out = '2'; break;
    case 2: out = '5'; break;
    case 4: out = '8'; break;
    case 8: out = '0'; break;
  }
  switch(rc[2]) { //Выявить единств кнопку в третьем столбце
    case 1: out = '3'; break;
    case 2: out = '6'; break;
    case 4: out = '9'; break;
    case 8: out = '#'; break;
  } //Последний столбец имеет высший приоритет
  return out;
} //-------------------------------------------
char kb_front(char in) {//Выявить фронт, т.е. момент нажатия
  static char kb_pred=0;
  char out;
  if(in && !kb_pred) out = in;
  else out = 0;
  kb_pred = in;
  return out;
} //------------------------------
enum type_reg { READY, INPUT, OPEN, CLOSE };//Перечисление состояний двери

int main()  {
  char kb, poz=0;
  static char mode=READY;//0-ready, 1-input,2-open, 3-close
  dig[0] = 0;//conv_hex_digit2dig7(0);
  dig[1] = sim_Y;//conv_hex_digit2dig7(1);
  dig[2] = sim_d;//conv_hex_digit2dig7(2);
  dig[3] = sim_r;//conv_hex_digit2dig7(3);//Стартовая загрузка индикатора
  DDRA = 0xff; //7-0: LED7 o seg a-dp
  DDRB = 0xff; PORTB = 0x1f; //3-0: LED7 o dig1-4, 4:o Rele
  DDRC = 0x70; PORTC = 0x7f; //3-0: ip kb_D-A, 6-4: o kb_3-1
  while (1)  {
    _delay_ms(5);
    dynamic_indicate();//Обслуживание индикатора
    kb = kb_front(kb_read());//Выявить момент нажатия кнопки
    if(kb) { //Есть такой момент!
      if(kb=='*') { //Если нажата кнопка *,
        dig[0]=dig[1]=dig[2]=dig[3]=poz=0;//то очистить буфер приема,
        PORTB &= ~(1<<4); //режим «Ввод пароля».
        mode = INPUT; }
      if(mode==INPUT && kb=='#') {//Если режим «Ввод…», кнопка #,
        if(dig[3]==dig7[3] && dig[2]==dig7[3] && dig[1]==dig7[3] &&
         dig[0]==dig7[3]) { //и в буфере правильный пароль “3333”,
```

```
          mode = OPEN; //то режим «Открыто»
          PORTB |= (1<<4);
          dig[3]=sim_G;dig[2]=sim_o;dig[1]=sim_o; dig[0]=sim_d;//"Good"
         } else { // и индицируется "Good",
          mode = CLOSE; //иначе режим «Закрыто»
          dig[3]=sim_b;dig[2]=sim_A;dig[1]=sim_d; dig[0]=0;// "bAd "
         }  // и индицируется "bAd "
        }
        if(mode==INPUT && (kb >= '0') && (kb <= '9')) {//Если нажата цифра,
          dig[3]=dig[2]; //то продвинуть буфер
          dig[2]=dig[1];
          dig[1]=dig[0];
          dig[0]=dig7[kb-'0']; //и код ввода тоже в буфер
        }
      }
  }
} //===========================================
```

Получилось несколько коряво – сравнение идет не по кодам или номерам клавиш, а по кодам их 7-сегм кодов, но работоспособно. Хороший стиль программирования требует разделить массив дисплея и массив кода замка. Такие приемы облегчают изменение и развитие задания, но выглядят сложнее. Пример находится в папке kb3x4_led7x4 в виде файла протеуса и проекта AVR-Studio.

Контрольные вопросы по разделам 1, 2

Теоретические

1. Алгоритм управления, программная и аппаратная реализация, фактор времени.
2. Системы счисления и форматы данных
3. Микроконтроллеры для встраиваемых применений.
4. Серия (семейство) микроконтроллеров.
5. Архитектура семейства 8-разрядных контроллеров.
6. Структурные схемы семейства 8-разрядных контроллеров.
7. Система команд процессорных устройств.
8. Методы адресации процессорных устройств.
9. Организация памяти и программно доступные регистры 8-разрядных контроллеров.
10. Арифметические команды
11. Логические команды.
12. Команды сдвига
13. Команды передачи данных
14. Команды переходов в программе
15. Команды работы с битами
16. Язык программирования ассемблер.
17. Механизм макросов
18. Механизм подпрограмм
19. Сравните языки программирования МК Ассемблер и C/C++
20. Этапы проектирования МПУ
21. Способы отладки программы
22. Программные и аппаратные средства отладки

23. Загрузка кода программы и данных в МК из персонального компьютера (ПК).
24. Тактирование и рестарт (сброс) в 8-разрядных МК.
25. Питание и энергопотребление 8-разрядных МК.
26. Устройство параллельного порта
27. Ввод через параллельный порт
28. Вывод через параллельный порт
29. Программный опрос и типовые задачи управления.
30. Периферийные устройства 8-разрядных МК.

Практические

1. Назовите режимы адресации и диапазоны адресов операндов в команде *LD R1,X(AVR) / MOV A,@R1(i51)* – на выбор.
2. Какие группы команд используют биты признаков процессора.
3. Какое время проходит с момента установления питания МК до выполнения первой команды и чем оно определяется?
4. Какие программные действия необходимы для настройки механизма вызова подпрограмм?
5. Составьте алгоритм деления на 8 двухбайтного числа R4:R5.
6. Для каких целей может использоваться стек.
7. Приведите пример команды с косвенно-индексной адресацией данных.
8. Приведите пример команды с косвенной адресацией команды (программы).
9. Чем может быть вызвано сообщение об ошибке в строке с командой RJMP LAB1?
10. Дан алгоритм. 1) 0 -> R0. 2) Тело цикла. 3) R0 – 1 -> R0. 4) Если R0 не 0, то идти к п.2), иначе конец. Сколько раз будет выполнено тело цикла в 8-разрядном МК? Команды каких групп необходимы для реализации алгоритма?
11. Какие из подгрупп команд имеют наибольшее время выполнения и чем это объясняется?
12. Назначение директив транслятора. В чем отличие директивы и команды ассемблера?
13. Перечислите основные этапы программирования на языке ассемблера и используемые вами программные средства. В чем заключается процесс отладки в симуляторе?
14. Перечислите названия и назначение основных окон среды программирования в режиме отладки.
15. Перечислите названия и способы использования команд выполнения программы в среде программирования в режиме отладки.
16. Перечислите преимущества и недостатки отладки программ в симуляторе по сравнению с целевым МК
17. Составьте алгоритм вычитания двухбайтных чисел в 8-разрядных МК
18. Составьте алгоритм умножения байта в R7 на константу 260 в 8-разрядных МК
19. Составьте алгоритм сравнения двухбайтного числа с константой 4000 в 8-разрядных МК
20. Составьте алгоритм сложения двухбайтного числа в R2:R3 с константой 100000

---

21. Типовая схема подключения МК с внутренним тактированием в корпусе DIP-8.
22. Типовая схема подключения МК с кварцевым резонатором на 8 МГц в корпусе DIP-20.
23. Схема и алгоритм светофора (3 светодиода на одно направление и 3 на другое).
24. Схема и алгоритм светофора по требованию (кнопка и 3 светодиода).

Часть 3. Ввод/вывод в МПУ

3.1. Понятие и характеристики интерфейса

Интерфейс (И) – организация взаимодействия различных частей управляющей или вычислительной системы (от английского *interface* – сопрягать, согласовывать). Основными элементами интерфейса являются:

- Совокупность правил обмена информации (временные диаграммы и диаграммы состояний сигналов интерфейса).
- Аппаратная реализация (физическая реализация) – приемопередатчики, разъемы, линии связи, контроллеры.
- Программное обеспечение интерфейса – подпрограммы-драйверы.

Интерфейс должен обеспечивать:

- Простое и быстрое соединение данного устройства с любым другим, имеющим такой же интерфейс;
- Совместную работу устройств без ухудшения их технических характеристик;
- Высокую надежность.

Под *стандартным интерфейсом* понимается совокупность аппаратных, программных и конструктивных средств, необходимых для реализации взаимодействия различных функциональных компонентов в системах и направленные на обеспечение информационной, электрической и конструктивной совместимости компонентов.

Основные характеристики интерфейса:

- производительность [бит/с = бод, Байт/с и т.д.],
- максимальная длина линий связи,
- число информационных линий связи – разрядность, различают одноразрядный – последовательный и многоразрядный – параллельный,
- направление передачи:
  - однонаправленный – симплексный,
  - полностью двунаправленный с возможностью одновременного приема и передачи – дуплексный,
  - двунаправленный с разделением времени работы линии на прием и на передачу – полудуплексный,
- по типу объединяемых устройств и распределению ролей между ними:
  - активный - пассивный/е: с постоянным или переменным положением активного,
  - активный - активный/е,

  активным, как правило, выступает устройство, содержащее процессор.
- по назначению:
  - внутрисистемный – как правило, высоко производительный И для связи процессора с блоками памяти и пр., обеспечивает функционирование ядра системы и наиболее требовательных по быстродействию периферийных устройств,

  - внутрикристальный – обеспечивает функционирование системы в пределах одной микросхемы (кристалла),
  - внешний – для связи ядра системы с внешними периферийными устройствами или для связи активных устройств распределенной системы управления.
- число устройств, объединяемых интерфейсом:
  - два устройства – тип «точка-точка» (*Point-To-Point – PTP*),
  - более двух устройств – «многоточечная» (*Multy-Point-Interface – MPI*), то есть локальная сеть; здесь термин «локальная» указывает на ограничение числа устройств и расстояний между ними.

Сети в свою очередь имеют дополнительные характеристики:

- топология, то есть способ объединения устройств в сеть:
  - радиальный или «звезда» – ведущее устройство имеет отдельные связи с каждым ведомым и обеспечивает передачу информации между ними; здесь высоки требования к надежности и производительности ведущего;
  - кольцо – каждое устройство имеет связь на прием от одного устройства и на передачу к другому; целостность системы разрывается при отказе единственного устройства;
  - шина – каждое устройство имеет связь на прием и на передачу со всеми устройствами сети; здесь возникает проблема разделения шины во времени между устройствами, но надежность системы высока, поэтому, этот принцип широко применяется в распределенных системах управления;
  - комбинированные,
- способ адресации устройств в сети и размер адресного пространства.

Параллельный и последовательный интерфейс

Принцип параллельного интерфейса – передача набора из N бит по N информационным линиям за 1 такт. Если разрядность шины R (число линий связи) меньше N, то потребуется N * R тактов передачи.

Принцип последовательного интерфейса – передача набора из N бит по одной информационной линии за N тактов.

Производительность интерфейса оценивается средним числом бит или байт, передаваемых за единицу времени (секунду). При одинаковой частоте тактирования приемника и передатчика последовательный интерфейс принципиально медленнее параллельного.

МК (процессор) работает с данными в *параллельном* виде, то есть с байтами, словами и т.д. Передача данных в этом формате поддержана параллельными связями в виде шин данных, адреса, параллельных портов и т.д. Хотя параллельный интерфейс в принципе самый высокопроизводительный, сейчас он все больше уступает последовательному интерфейсу для связей вне кристалла.

При передаче данных на значительное расстояние экономия числа проводников традиционно определяет преимущества последовательного интерфейса перед параллельным (глобальные связи и сети).

В последнее десятилетие параллельный интерфейс все более заменяется последовательным и во внутрисистемных локальных связях и сетях. Это связано с перекрестными помехами в параллельных шинах. При малых зазорах между проводниками и большой длине проводников растет влияние паразитных емкостей между ними, быстрое изменение уровня напряжения в одном проводнике вызывает появление ложного импульса в соседних проводниках. Рост тактовых частот на шинах сегодня ограничивает применение параллельного интерфейса в основном только внутри кристалла.

Компенсация снижения производительности при переходе от параллельного интерфейса к последовательному идет за счет роста частоты синхронизации.

3.2. Внутрисистемные интерфейсы в МПУ

Объединение модулей микропроцессорного устройства в единую систему производится посредством единой системы сопряжения, называемой внутрисистемным интерфейсом.

МК (процессор) работает с данными "в параллельном" виде, то есть с байтами, словами и т.д. Передача данных в этом формате поддержана параллельными связями в виде шин данных, адреса, параллельных портов и т.д.

Основной механизм внутрисистемного интерфейса – внутренняя системная магистраль или набор системных магистралей. Для работы с периферийными устройствами часть линий системной магистрали выводится на внешние выводы.

В МК серии AVR имеются отдельные магистрали доступа к памяти программ (только встроенная FlashROM) и доступа к памяти данных (встроенные РОН, РВВ, ОЗУ, ЭСППЗУ и внешнее ОЗУ).

Каждая из этих магистралей состоит из *шин адреса, данных и управления*.

Шиной [bus] называется группа проводников, состоящая из информационных линий и хотя бы одного общего проводника, служащая для передачи логически связанной информации.

Шина адреса ША [address bus] представляет собой набор проводников, по которому передается код адреса от активного устройства АУ (процессор) ко всем подключенным пассивным устройствам ПУ1, ПУ2, ... Используется однонаправленная передача, то есть АУ является источником, все ПУ – приемниками. Разрядность определяет размер доступного адресного пространства, обычно кратна 8 битам, хотя может наращиваться и поразрядно.

Шина данных ШД [data bus] представляет собой набор проводников, по которому передаются данные либо от АУ к выбранному (по адресу) ПУ – операция записи, либо в обратном направлении – операция чтения. Шина данных двунаправлена, это значит, что одно и тоже устройство служит как источником, так и приемником сигнала. В интервалах времени, когда роль устройства не определена, оно должно переходить в высокоомное состояние, чтобы не создавать дополнительной нагрузки для источника сигнала – так называемое Z-состояние. Разрядность шины данных (то есть число проводников) обычно совпадает с разрядностью процессора.

Шина управления ШУ [control bus] состоит из проводников для передачи/приема

набора сигналов, синхронизирующих операции обмена данными в магистрали. Состав сигналов шины управления сильно зависит от предусмотренного набора операций ввода/вывода.

1) Наиболее простым видом является *программный ввод/вывод*. Инициировать операцию ввода или вывода может только процессор (активное устройство), для этого достаточно всего двух линий управления – сигнала WR (запись) и RD (чтение).

Процедура записи (вывода) начинается с выставления процессором адреса на ША. Все ПУ дешифрируют адрес, единственное ПУ с помощью селектора адреса опознает свой. Далее АУ выставляет данные на ШД и выдает фронт сигнала записи (*WR*), по которому выбранное ПУ переводит выводы, подключенные к ШД в состояние приема. Затем, по срезу сигнала записи *WR* информация фиксируется в регистре данных ПУ, затем АУ снимает данные и может начать новый цикл.

Процедура чтения (ввода) подобна процедуре записи, но в качестве строба используется сигнал чтения (*RD*), выбранное ПУ выставляет данные по фронту сигнала чтения, и снимает их по срезу, в остальное время выводы данных ПУ находятся в Z-состоянии.

Подробнее фазы операций записи и чтения можно рассмотреть на рис. 2-3, а – только фазы Т2 и Т3.

При обращении к устройствам внутри кристалла МК/МП точно известно максимальное время, необходимое для выполнения всех этапов процедур чтения и записи. При обращении к устройствам вне МК/МП обычно существует механизм подстройки под более медленные устройства.

Программный ввод или вывод выполняется одной машинной командой. Например, в *AVR* МК для работы с РВВ используются команды *IN* или *OUT*, для работы во всем адресном пространстве памяти данных – команды *LDx* или *STx*.

Различают программный ввод/вывод без квитирования и с квитированием. Квитирование необходимо для синхронизации операций ввода или вывода с завершением действий, выполняемых периферийным устройством. Например, для чтения нового значения аналого-цифрового преобразователя после запуска преобразования требуется дождаться завершения преобразования. Синхронизация выполняется чтением в регистре состояния бита (флага) готовности. Если бит установлен, можно выполнять чтение или запись.

В противном случае следует либо организовать частный цикл ожидания готовности, либо вернуться к проверке бита готовности на следующем цикле работы программы. Первый вариант гарантирует минимальную задержку выполнения операции ввода/вывода, при этом существует опасность зависания в цикле ожидания (если бит готовности не устанавливается за ожидаемое время). Во втором варианте нет опасности зацикливания, но задержка в выполнении операции ввода/вывода может достигать длительности двух циклов работы программы.

2) *Ввод/вывод по прерываниям* позволяет прервать последовательное выполнение операторов программы для выполнения малого количества процедур чтения или записи и возвращения к выполнению прерванной программы. Сами процедуры чтения/записи выполняются программным способом и оформляются в виде подпрограммы обслуживания прерывания. Для поддержки механизма прерываний в шине управления

необходимы индивидуальные линии запроса прерывания, механизм предоставления прерывания с аппаратными и программно настраиваемыми приоритетами и механизм формирования вектора прерывания – адреса перехода при предоставлении прерывания.

3) Если растут объемы или частота обмена информацией между периферийными блоками и блоками памяти, то процессор оказывается загружен непроизводительными операциями: не сразу от пассивного устройства А к пассивному устройству Б, но каждый раз от А к процессору, затем от процессора к Б. Для таких задач более подходит принцип прямого доступа к памяти. В этом случае процессор перестает быть единственным активным устройством на системной магистрали, появляется контроллер прямого доступа к памяти (КПДП). Он может захватить магистраль и управлять передачей данных сразу между ячейками с заданными адресами. Предварительно настраивается (программируется) количество передач, адреса приема и передачи, механизм автоматического изменения адреса, система приоритетов для захвата/освобождения магистрали и связь с системой прерываний, для организации реакции по завершению заданного количества передач.

Программный ввод/вывод и ввод/вывод по прерываниям являются основными механизмами внутрисистемного ввода/вывода в МПУ. Их использование для интерфейса с внешними устройствами сдерживается значительным количеством линий, что усложняет разводку плат, делает их дорогими.

Механизм прямого доступа аппаратно еще более сложен, поэтому распространен только в высокопроизводительных процессорных устройствах.

В *AVR* МК на внутрисистемном уровне в подсемействах *AT90, ATtiny, ATmega* используется ввод/вывод программный и по прерываниям; в новом подсемействе *ATxmega* добавился механизм прямого доступа к памяти. На уровне связи вне кристалла используются прежде всего параллельные порты, входы прерывания, таймеры/счетчики, устройства аналогового и последовательного интерфейса; в моделях с числом выводов (ног) не менее 40 добавляется внешняя магистраль программного доступа к памяти данных.

3.3 Параллельный порт AVR

См. п.1.6. Для наблюдения и управления каждым 8-разрядным портом (например, портом А) используются три 8-разрядных регистра в субпространстве РВВ – *DDRA, PORTA* и *PINA*. Это значит, что каждый из них подключен к внутренней системной магистрали по шине данных, имеет свой селектор адреса (коды адреса из файла описания МК: *PORTA = 0x1b, DDRA = 0x1a, PINA = 0x19*). Причем регистры *DDRA, PORTA* доступны по чтению и записи, то есть формально их можно использовать как ячейки оперативной памяти. Регистр *PINA* доступен только по чтению, так как служит лишь для наблюдения за состоянием 8 выводов *PA0…PA7 независимо* от настройки выводов на ввод или вывод.

3.4. Внешняя магистраль памяти данных AVR МК

Для уменьшения числа выводов используется мультиплексирование шины данных (8 разрядов *D0…D7*) и адреса (16 разрядов *A0…A15*). Мультиплексирование заключается во

временном разделении – сначала по 8 линиям AD0…AD7 передаются 8 младших бит адреса, затем – 8 бит данных, для синхронизации смены адреса на данные используется выход ALE (Address Latch Enable), для синхронизации вывода используется выход WR, для синхронизации ввода – выход RD. Всего 19 выводов МК AVR (рис. 13) позволяют иметь внешнее адресное пространство 64 килобайта, в нем могут располагаться как ячейки памяти данных, так и устройства ввода/вывода.

Для подключения внешних устройств (SRAM объемом 2^{16} байт на рис. 3.1) необходима микросхема регистра-защелки для фиксации 8 младших разрядов адреса по сигналу ALE и разрешение работы с внешней памятью данных (бит SRE в регистре MCUCR). В этом случае линии порта A PA0…PA7 используются как AD0…AD7, линии порта C PC0…PC7 – как A8…A15, PD6 = WR, PD7 = RD. Микросхема регистра-защелки по сигналу ALE записывает младшие разряды адреса (на входах D0…7) и фиксирует их на своих выходах (Q0…Q7) до следующей операции на внешней магистрали. Микросхема внешней памяти данных (SRAM) или внешнее устройство ввода/вывода с подобным интерфейсом, подключается двунаправленным портом D[7:0] к линиям порта A, входы адреса младшие разряды A[7:0] получают с регистра защелки, старшие A[15:8] – с выходов порта C, входы записи и чтения – к одноименным выходам МК.

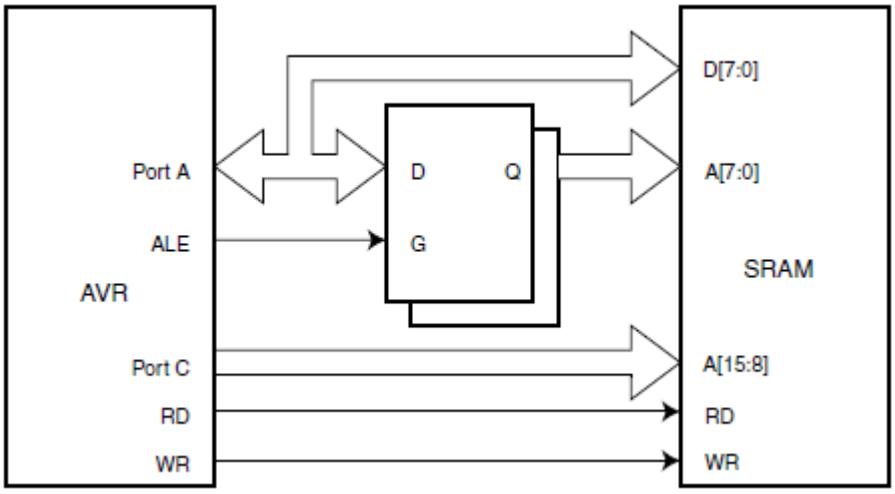


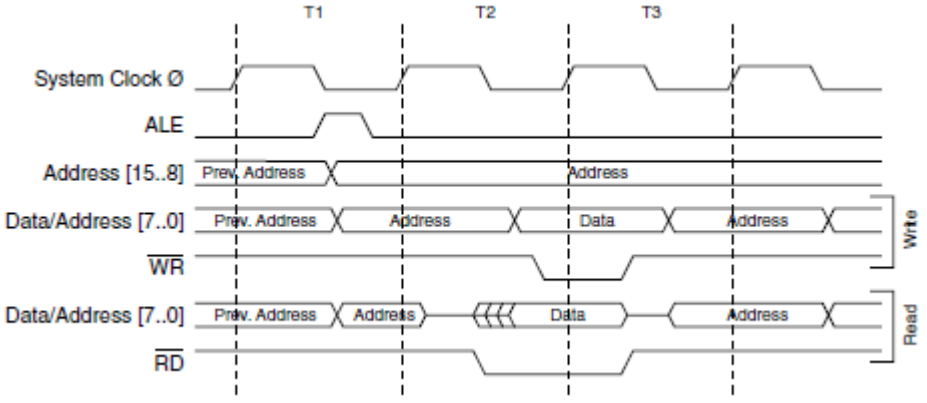
Рис. 3.1

При соблюдении требований по времени выборки адреса обмен по внешней магистрали также может выполняться за одну команду (`LDx/STx`), для незначительного увеличения длительности цикла чтения или записи по внешней магистрали используются биты настройки SRW регистра MCUCR. На рис. 3.2, представлены временные диаграммы сигналов на линиях внешней магистрали. Обычный цикл обмена (рис. 3.2, а) занимает 3 машинных цикла, первый служит для фиксации младших разрядов адреса в регистре-защелке, во втором начинается, в третьем заканчивается процедура записи или чтения. Увеличенный цикл (рис. 3.2, б) занимает 4 машинных цикла, «пустой» цикл увеличивает интервал между началом и концом процедуры записи или чтения.

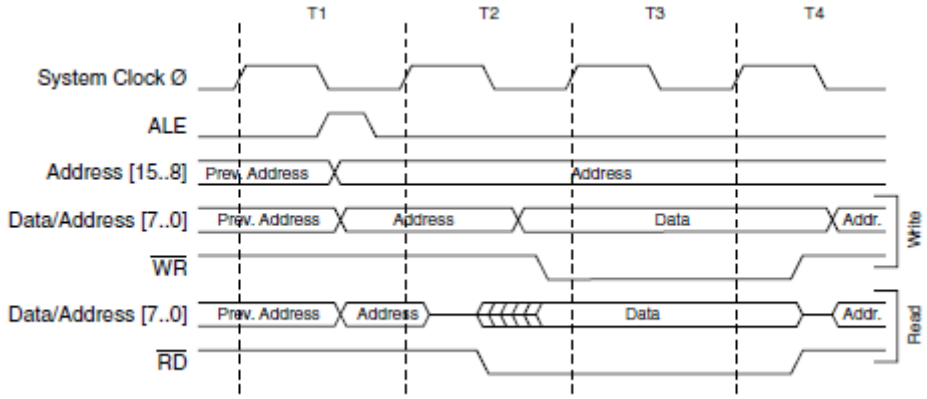
Обращаем внимание, что сигналы записи WR и чтения RD работают в инверсной логике – фронт сигнала начинается с перехода к низкому уровню, срез – к высокому.

В среде моделирования Proteus VSM есть схемно-программный пример подключения внешней памяти данных объемом 64 КБ с временными диаграммами сигналов в процедурах записи и чтения. Меню «Справка → Примеры проектов» < \

Sample \ VSM for AVR \ AVR External Memory \ extram.dsn >.



а



б

Рис. 3.2

Для увеличения адресного пространства свыше 64 КБ можно использовать не задействованные линии параллельных портов – каждая линия увеличивает размер пространства вдвое. В этом случае для смены полного адреса необходимо менять сигналы на выходах портов. Пример подключения внешней памяти 512 КБ рассмотрен в [Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. – М.: Издательский дом «Додэка-XXI», 2004. – 288 с.].

3.5. Принцип и средства ввода/вывода по прерываниям

Показания к применению: уменьшение времени реакции и выполнение действий, асинхронных к периоду и фазе основного цикла.

Основные *этапы* обработки прерывания, формирование *запроса* и *вектора (*адрес в таблице векторов), сохранение и восстановление *контекста*, временные затраты – задержка реакции и задержка фоновой программы. *Флаги* прерывания, *маскирование* общее и индивидуальное, система *приоритетов* – аппаратная (жесткая) и программная (гибкая), вложенные прерывания. Контроллер прерываний, аппаратные и программные прерывания, *внутренние и внешние прерывания*.

Входы прерываний: событие (переход и уровень), регистры управления и состояния, структура ПО (пример).

Сторожевой таймер: аппаратная структура, регистры управления и состояния, пример расчета времени и использования команд (обратить внимание на *ISR* и циклы ожидания готовности).

Механизм прерываний

Механизм прерываний *[interrupt]* удобен для быстрой программной реакции на события (сигналы) асинхронные с основным циклом управляющей программы. По

запросу прерывания *[interrupt inquiry]* приостанавливается выполнение текущей программы (фоновой программы), выполняется короткая подпрограмма обслуживания данного запроса, далее продолжается выполнение фоновой программы.

Применение прерываний оправдано, когда время выполнения всех подпрограмм обслуживания прерываний много меньше периода основного цикла. Основным критерием эффективности механизма прерываний является снижение максимального времени реакции на событие, породившее запрос. При этом не должно быть заметного роста времени реакции остальных задач.

В конце каждого машинного цикла при наличии общего разрешения прерываний процессор проверяет наличие запросов прерывания. Если данное прерывание разрешено, процессор выполняет действия аналогичные вызову подпрограммы с заранее определенным адресом (вектором), и сохраняет в стеке адрес прерванной программы. В конце подпрограммы обслуживания прерывания *[Interrupt SubRoutine - ISR]* должна находиться команда возврата из прерывания *[Return Interrupt]*. При ее выполнении происходит возврат в точку прерывания фоновой программы.

Выполнение подпрограммы обслуживания прерывания должно быть «прозрачным», т. е. незаметным для прерываемой (фоновой программы) – не должно нарушаться содержимое регистра состояния (флаги процессора) и тех РОН, которые используются фоновой программой. Названные ресурсы называются *контекстом*. Если *ISR* вынуждена делить ресурсы с другими программами, требуется при входе в подпрограмму сохранять содержимое используемых ресурсов в стеке, а перед выходом из *ISR* – восстанавливать из стека в обратном порядке. В высокопроизводительных процессорных устройствах слово состояния процессора сохраняется аппаратно контроллером прерываний, в 8-разрядных это забота программиста.

Общим правилом является стремление к минимизации времени выполнения *ISR*. Поэтому непосредственно в *ISR* выполняются только самые необходимые быстрые действия, а выполнение долгих переносится в подпрограммы основного цикла за счет взаимодействия через общие (глобальные) переменные. По возможности следует избегать арифметических и логических действий в *ISR*, они изменяют содержимое флагов процессора.

Прерывания бывают внешние – формируются сигналами определенного вида на внешних входах МК и внутренние – формируются внутренними, в том числе периферийными узлами МК. Каждый источник прерывания имеет индивидуальные вектор, флаг, маску и уровень приоритета.

Вектором называется фиксированный адрес начала *ISR*. Обычно область *ISR* размещается в начале памяти программ, сразу после ячейки с адресом 0 прерывания по пуску/перезапуску МК. В области адресов обычно располагаются команды перехода к подпрограммам обслуживания прерываний, иногда – подпрограмма обслуживания (если она умещается в 1-2 команды)

Флагом запроса прерывания называется бит в РСФ, устанавливаемый в «1» аппаратурой МК при формировании условия прерывания. Сброс этого бита обычно выполняется автоматически при вызове *ISR* или программно, если прерывание не разрешено.

Маской называется бит в РСФ, служащий для разрешения индивидуального

прерывания. *ISR* начнет выполняться тогда и только тогда, когда будут установлены биты флага, маски и бит глобального (общего) разрешения прерывания.

При одновременном поступлении более чем одного разрешенного запроса прерывания вступает в действие механизм приоритетов, который выбирает для обслуживания только один запрос. В простых 8-разрядных МК система приоритетов ограничена аппаратным распределением уровней – чем меньше численное значение вектора (адреса), тем выше уровень приоритета.

В более сложных МК в дополнение к аппаратным существуют программные уровни приоритета. Они позволяют объединять прерывания в группы, внутри которых действует аппаратный механизм распределения приоритетов.

Прерывание, не получившее права обслуживания (по приоритету, по маске, по общему запрету), сохраняет в высоком состоянии бит флага запроса до разрешения или до принудительного программного сброса.

Работа с прерываниями на языке *C/C++* имеет ряд особенностей, связанных с оформлением вызова *ISR* и обмена данными.

Механизм прерываний не входил в стандарт языка *C/C++*, поэтому в разных трансляторах встречаются отличия в оформлении механизмов прерывания. Общим правилом является обозначение *ISR* как функции с пустыми *[void]* списками входных и выходного параметров и объявление вектора прерывания (адреса) как символьной константы (как правило, после служебного слова *interrupt*). Закрывающая фигурная скобка данной функции заставляет транслятор завершать текст подпрограммы командой возврата из прерывания [*reti* для *AVR*].

Для обмена данными используется механизм глобальных переменных.

Сохранение и восстановление контекста (как и вообще работа с РОН) выполняется транслятором без участия программиста. Но это не избавляет программиста от заботы по сокращению времени выполнения *ISR* за счет отказа или минимизации вычислительных процедур в теле *ISR*.

Генерация команд общего разрешения/запрещения прерываний выполняется либо с использованием механизма ассемблерных вставок *[#asm("sei") в CodeVisionAVR]*, либо в виде специальных функций *[__enable_interrupt(); в IAR-C/C++]*.

3.6. Принцип прямого доступа к памяти

Показания к применению: значительная производительность обмена данных между блоками памяти и устройствами ввода/вывода.

Такие задачи решаются, как правило, с использованием высокопроизводительных процессорных устройств (16/32 разрядные МК и ЦСП). Обмен с прямым доступом требует от процессора только настройки адресов приемника и передатчика, задания формата данных (байт, слово и пр.), определения количества передач и сигнала синхронизации передач, возможности авто-увеличения или уменьшения адресов приемника и передатчика, разрешения запроса прерывания после завершения заданного количества передач. Далее все действия выполняются аппаратно контроллером прямого доступа к памяти (ПДП) *[Direct Memory Access - DMA]* в фоновом режиме относительно

работы процессора, для чего используются системная магистраль в интервалах, не занятых процессором, либо дополнительная магистраль при ее наличии.

В современных МК устройства ПДП различаются по своим характеристикам и называются по-разному: контроллер событий *[Event Controller]*, массив процессоров событий *[Event Processor Array]*, сервер периферийных транзакций *[Peripheral Transaction Server]* и пр.

Рассмотрим основные черты контроллера ПДП ЦСП.

Регистры управления и состояния:

- *общий регистр управления* ПДП (РУ ПДП) определяет режим работы контроллера и характер изменения адресов источника и приемника (инкремент, декремент, синхронный);

- *регистры адресов* источников (РАИ) и приемников (РАП) для каждого канала ПДП;

- *регистры счетчиков пересылок* (РСП), управляющие размером кадров и блоков пересылок по каждому каналу ПДП;

- *регистр разрешения прерываний* (РРП) ЦПУ/ПДП.

Структура передаваемых данных:

1) Элемент программно выбираемого размера (1/2/4 байта);
2) Кадр, состоящий из N элементов (обычно до 64 К);
3) Блок, состоящий из К кадров (обычно до 64 К, т.е. до 4 G элементов).

Пересылка элемента в каждом канале выполняется за 2 шага:

1) Чтение элемента данных по адресу РАИ;
2) Запись прочитанного элемента по адресу РАП.

После этого модифицируются значения РАИ и РАП по правилам, выбранным в РУ ПДП и выполняется декремент РСП. По окончании пересылки блока генерируется прерывание.

Синхронизация каналов ПДП выполняется по внешним или внутренним прерываниям. Программно выбирается одно из прерываний и один из четырех способов синхронизации:

1) нет синхронизации – все прерывания игнорируются, ПДП выполняет чтения/записи в максимально возможном темпе;
2) синхронизация источника – чтение по выбранному прерыванию;
3) синхронизация приемника – запись по выбранному прерыванию;
4) синхронизация кадра – начать пересылку кадра по прерыванию.

Генерация адресов ПДП выполняется с использованием индивидуального набора индексных регистров: базовый адрес, индекс элемента (смещение на 1/2/4 байта) и индекс кадра (смещение при переходе к новому кадру).

Приоритеты ПДП/ЦПУ: приоритеты между каналами ПДП – фиксированные (0 - старший, 3 - младший), для каждого канала программно задается приоритет по отношению к ЦПУ; порядок разбора состязания - сначала выявляется самый старший из активированных каналов ПДП, затем сравнение его приоритета с процессором. Устройство не получившее право доступа ставится в очередь. Для сохранения данных от источника в очереди используется буфер *FIFO* (9 ячеек) и еще два регистра задержек.

Часть 4. Прерывания

4.1. Механизм прерываний в *AVR* и его программирование

Прерывания в МК серии AVR

Таблица прерываний (см. Табл. 1) начинается с нулевой ячейки памяти программ (.cseg) и имеет шаг в одну или две ячейки. Размер шага определяется объемом памяти программ – если объем не превышает 2^{12} = 4096 байт, то для перехода к любой ячейке памяти достаточно команды *rjmp a12*, она занимает в памяти одну ячейку; при большем объеме ПП требуется команда *jmp a16* размером в две ячейки.

Размер таблицы равен числу источников прерываний плюс один и зависит от модели МК. Однотипные источники прерываний в разных моделях МК одного семейства нередко имеют разные вектора. Для удобства программиста в системах программирования для каждой модели МК поставляются файлы описания символьных констант (например, *2323def.inc* для ассемблера в среде *AVR-Studio* или *90s2323.h* для языка *C* в среде *CodeVision*), включающие символьные имена векторов, их написание несет смысловую нагрузку.

В первой ячейке (адрес 0) располагается первая команда, выполняемая после рестарта МК. Как правило, это команда перехода к этапу инициализации для обхода таблицы прерывания. Если в вашей программе предусмотрено общее разрешение прерываний, то по адресам векторов располагаются либо команды перехода к началу соответствующей *ISR* (*rjmp/jmp VECTOR_N_SUBROUTINE*), либо команда выхода из прерывания *[reti]*.

ATtiny15L				ATmega128			
Вект.	Адрес ПП	Источник	Описание	Вект.	Адрес ПП	Источник	Описание
1	0	RESET	External Reset, Power-on Reset, Brown-out Reset, and Watchdog Reset	1	0	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	1	I/O Pins	External Interrupt Request 0	2	2	INT0	External Interrupt Request 0
3	2	Tim1,CompA	Pin Change Interrupt	3	4	INT1	External Interrupt Request 1
4	3	Tim1,Ovf	Timer/Cnt.1 Compare Match A	4	6	INT2	External Interrupt Request 2
…	…	…	…	…	…	…	…
9	4	ADC	ADC Conversion Complete	35	0x44	SPM READY	Store Program Memory Ready

Флаг глобального прерывания находится в регистре состояния процессора *SREG.7*, имеет символическое имя *I [Interrupt]*, для общего разрешения прерываний используется команда *sei*, для запрещения - *cli*.

На этапе инициализации, как правило, устанавливаются биты регистров масок прерывания *[*MSK]*, для излишней надежности сбрасываются биты запросов прерывания *[*FR]*. Последней командой перед входом в основной цикл выполняется общее разрешение прерываний.

Впрочем, это не мешает в процессе выполнения основного цикла и подпрограмм обслуживания прерывания менять состояние масок и общего разрешения. Общий запрет прерывания *[cli]* следует выполнять перед некоторыми специфическими процедурамии программирования,

называемыми *[atomic]*, в которых определенное состояние триггеров аппаратных узлов удерживается на ограниченном числе машинных циклов. К таким процедурам относятся чтение или запись 16-разрядных регистров ввода/вывода, при этом используются скрытые (теневые) регистры.

Прерывание разрешается к выполнению, если в момент активации запроса есть общее разрешение (*I* бит в *SREG*) и индивидуальное разрешение (маска). При одновременном возникновении двух разрешенных запросов буде обрабатываться запрос с более высоким приоритетом. Приотритет определяется адресом в таблице векторов, меньшее значение адреса имеет больший приоритет. При общем запрете прерываний индивидуально разрешенные запросы запоминаются до общего разрешения, далее обрабатываются с учетом приоритетов.

Процедура предоставления прерывания МК серии *AVR* детально состоит из следующих этапов.

1) Этап аппаратного предоставления прерывания: в конце текущей команды при наличии разрешения процессор сохраняет в стеке адрес следующей команды прерываемой (фоновой) программы, сбрасывает флаг запроса прерывания, загружает в счетчик команд значение вектора прерывания и сбрасывает бит *I* регистра *SREG* (для запрета обслуживание вложенных прерываний). Время выполнения этого этапа зависит от длительности прерываемой команды и длительности перехода на начало *ISR* и составляет для МК серии *AVR* 5...6 машинных циклов.

2) Команда перехода из таблицы векторов к первому оператору *ISR* – на ассемблере программист по адресу прерывания (вектору) размещает команду перехода к тексту подпрограммы (2…3 м.ц.), на языке *C/C++* эта команда не видна, она порождается строкой вызова функции прерывания.

3) Этап сохранения контекста – на ассемблере при необходимости программируется разработчиком – состоит в сохранении значений *SREG* и тех РОН, которые приходится "делить" между *ISR* и фоновой программой/программами. Если файл РОН недостаточен для разграничения между подпрограммами и основной программой, следует сохранять значения контекста в стеке *[push]* в определенном порядке. На языке *C/C++* транслятор сам определяет необходимость и порядок сохранения контекста.

4) Этап выполнения собственно *ISR*, т. е. запланированных разработчиком действий.

5) Этап восстановления контекста – на ассемблере в обратном порядке из стека восстанавливаются значения РОН и *SREG*, на *C/C++* обо всем заботится транслятор.

6) Этап выхода из прерывания выполняется с помощью команды *RETI [RETurn Interrupt]* – из стека восстанавливается значение счетчика команд (4…5 машинных цикла), возобновляется выполнение прерванной программы, причем, первая команда обязательно выполняется ($I = 0$), после чего устанавливается бит *I* общего разрешения прерывания и может начаться обслуживание следующего прерывания.

Таким образом, минимальная задержка между установкой флага запроса прерывания и первой командой обработки прерывания составляет 7…9 м. ц. Минимальное время, на которое задерживается выполнение фоновой программы, составляет 13…15 м. ц. (если отсутствует сохранение/восстановление контекста и «содержательная» часть *ISR* состоит из 2 м. ц.).

Из описанной последовательности ясно, что не предусмотрено автоматическое прерывание выполняемой в данный момент подпрограммы обслуживания прерывания.

Тем не менее, программист может в ходе выполнения подпрограммы обслуживания установить бит SREG.I, разрешая тем самым прерываивние текущей подпрограммы.

Прерывания не были предусмотрены при разработке языка С, поэтому реализация зависит от транслятора (в деталях синтаксиса) и его настроек оптимизации.

В *IAR-C* и *CodeVision-AVR* подпрограмма обслуживания прерывания оформляется как функция со специальным модификатором *interrupt*, за которым в квадратных скобках следует вектор прерывания, а затем уже имя функции без входных (*void*) и выходных (*void*) параметров:

```
#pragma vector = VECTOR // IAR-C, line 1
__interrupt void name_functiont(void) { /* тело функции */ } // IAR-C, line 2

interrupt [VECTOR] void name_function(void) {/*тело функции*/}
//CodeVision-AVR

interrupt [TIM0_OVF] void timer0_ovf_isr(void) { //CodeVision-AVR
  TCNT0 = -156;//Период 200 Гц = 8MHz/256/78
  flag200Hz=1;
} // ******************************
```

В *GCC (WinAVR)* применяется более короткая запись *[*здесь *ISR – Interrupt SubRoutine]*:

```
ISR[VECTOR] { /* тело функции */  }
```

Транслятор размещает по адресу вектора прерывания команду перехода к телу функции, а в нем – команды сохранения контекста, далее – команды, полученные в результате трансляции тела ППОП, далее – команды восстановления контекста и, наконец, команду *RETI.*

Для передачи параметров в/из ППОП существует только механизм глобальных переменных, для запоминания значений между вызовами – статические переменные.

Для минимизации времени выполнения прерывания надо стремиться писать короткие программы обслуживания с минимальным использованием локальных и глобальных переменных, особенно при большом количестве одновременно разрешенных прерываний или их относительно высокой частоте повторения.

4.2. Входы прерываний *INTx* и *PCINTx*.

Внешние прерывания

Внешние прерывания в МК серии *AVR* ранних моделей состоят только из альтернативных функций *INTx*. Они поддерживают аппаратное выявление фронта, среза, пр. и формирование индивидуального запроса прерывания, такими функциями оснащались от 2-х до 8-ми отдельных выводов портов. В более поздних моделях к этим функциям добавились функции аппаратного выявления изменения состояния вывода *PCINTn [Pin Change INTerrupt]*, которыми оснащаются все выводы портов.

Оба механизма ориентированы на автоматизацию обслуживания входов, то есть контроля состояния внешних сигналов. Но это не мешает использовать их для контроля изменения состояния выводов в режиме выхода, то есть формировать программно управляемые прерывания.

В МК *AT90S8515* имеется два входа прерывания *INT0/PD2, INT1/PD3*. Событием

может быть (табл. 4.1): низкий уровень, фронт, срез, либо любое изменение (фронт/срез). Тип события задается программно битами *ISCX1, ISCX0* в регистре *MCUCR* (табл. 4.2).

Таблица 4.1. Внешние прерывания МК *at90s8515*

Источник	Вектор	Флаг	Маска	Выбор формы сигнала
INT0 / PD2	*INT0_vect = 0x02*	*GIFR:INTF0*	*GIMSK:INT0*	*MCUCR:ISC01, ISC00*
INT1 / PD3	*INT1_vect = 0x04*	*GIFR:INTF1*	*GIMSK:INT1*	*MCUCR:ISC11, ISC10*

Таблица 4.2. Настройка регистра MCUCR на событие на входах INT0, INT1

ISCx1	*ISCx0*	Форма прерывающего сигнала
0	0	Низкий уровень
0	1	Резерв
1	0	Срез (переход от 1 к 0)
1	1	Фронт (переход от 0 к 1)

Прерывание по фронту или срезу устанавливает флаг в регистре *GIFR*, его сброс выполняется автоматически при вызове подпрограммы прерывания (*ISR*), если данное прерывание запрещено флаг следует сбросить программно записью единицы в бит флага.

Прерывание по низкому уровню отличается от других отсутствием триггера запоминания, поэтому это прерывание активно все время, пока на входе присутствует низкий уровень, и «забывается», если к моменту предоставления прерывания уровень стал высоким.

Программное обеспечение прерываний состоит из подготовки (инициализации) и собственно *ISR*. На этапе инициализации программируются регистры маски [*GIMSK*], настройки на событие [*MCUCR*], настройки соответствующего вывода на вход [*DDRD*]. Последним действием этапа инициализации обычно выполняется команда общего разрешения прерывания [*sei*].

Обязательными элементами *ISR* являются целевые действия и команда выхода из прерывания [*reti*].

Ниже приведен пример использования входов прерывания *INT0* и *INT1* с реализацией на ассемблере и на *GNU-C*, схема примера и результаты измерений в симуляторе *AVR-Studio*, находятся в папке Primer / p2 Interrupt & TimerCounter / Int0cnt.

Задание к Примеру

Вход *INT0/PD2* – счет импульсов, если частота выше порога 200 Гц, то включить красный светодиод на выходе *PD4*, иначе переключать светодиод с частотой 10 Гц.

Вход *INT1/PD3* – по срезу кнопки на этом входе переключать зеленый светодиод на выходе *PD0*

Задержка выполнения измерялась от изменения состояния *PIND.2* / *PIND.3* до первой команды / оператора, выполняющей заданное действие (или выбор действия), то есть время реакции на событие. Время выполнения показывает задержку основного цикла на обслуживание данного прерывания.

Преимущество варианта на ассемблере перед свободно распространяемым транслятором *GNU-C (avr-gcc)* объясняется оптимальным расходом на сохранение/восстановление контекста.

Любопытные могут увидеть машинный код в файле листинга <*.*lss*>.и сравнить его с ассемблерным вариантом. Подпрограмма `ISR_INT0` использует в качестве ячейки «глобальной» памяти РОН, для сохранения регистра состояния используется еще один РОН. Любой транслятор поместит «глобальную» ячейку в *SRAM*, для сохранения контекста использует стек в *SRAM*. Подпрограмма `ISR_INT1` использует команды, которые не используют РОН и не меняют содержимое регистра состояния.

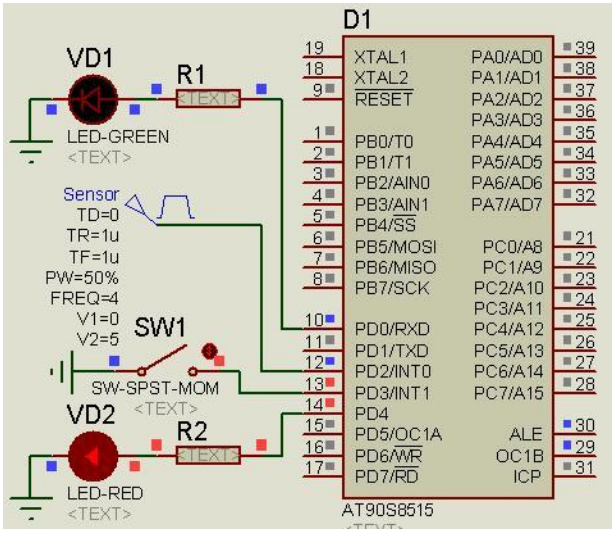
Пример на Ассемблере

```
.include <8515def.inc>
.macro LOAD ;@0, @1
       ldi r16,@1
       out @0,r16
.endm
.def cnt = r17
.def tmp_int0 = r0
.equ N = 10
       rjmp START
.org INT0addr
       rjmp   ISR_INT0
       rjmp   ISR_INT1
.org INT_VECTORS_SIZE
START:
       LOAD SPL,LOW(RAMEND)
       LOAD SPH,HIGH(RAMEND)
       LOAD DDRD,(1<<PD0)|(1<<PD4)
       LOAD PORTD, (1<<PD3)|(1<<PD0)
       LOAD GIMSK, (1<<INT0)|(1<<INT1)
       LOAD                              MCUCR,
(1<<ISC11)|(1<<ISC01)|(1<<ISC00)
       clr    cnt
       sei
LOOP:
       ldi           r18, 100
       rcall  WAIT_ms
       cpi           cnt, N
       brlo   LL
       sbi           PORTD,PD0
       rjmp   MM
LL:
       sbic   PORTD,PD0
       rjmp   BB
       sbi           PORTD,PD0
       rjmp   MM
BB: cbi              PORTD,PD0
MM:    clr           cnt
       rjmp   LOOP
; ********************************
ISR_INT0:
       in     tmp_int0,SREG
       inc cnt
       out    SREG,tmp_int0
       reti
; ====================================
ISR_INT1:
       sbic   PORTD,PD4
       rjmp   AA
       sbi           PORTD,PD4
       reti
AA: cbi              PORTD,PD4
       reti
; ====================================
```

Задержка выполнения действия
  в ISR_INT0:   10 мкс, 10 м.ц.
  в ISR_INT1:   7 мкс, 7 м.ц.
Время выполнения
  в ISR_INT0:   16 мкс, 16 м.ц.
  в ISR_INT1:   15 мкс, 15 м.ц.
Период основного цикла 100318 мкс

Пример на GNU-C

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/delay.h>
#define N 10
char cnt;
ISR(INT0_vect)  {  cnt++; }

ISR(INT1_vect)  {
  if(PORTD & (1<<PD4)) PORTD &= ~(1<<PD4);
  else PORTD |= (1<<PD4);
}

int main()  {
  DDRD = (1<<PD0)|(1<<PD4);
  PORTD = (1<<PD3)|(1<<PD0);
  GIMSK = (1<<INT0)|(1<<INT1);
  MCUCR = (1<<ISC11)|(1<<ISC01)|(1<<ISC00);
  GIFR = (1<<INTF1)|(1<<INTF0);
  sei();
  while(1) {
    _delay_ms(100);
       if(cnt > N) {
         PORTD |= (1<<PD0);
       } else {
         if(PORTD   &   (1<<PD0))   PORTD   &=
~(1<<PD0);
         else   PORTD |= (1<<PD0);
       }
       cnt=0;
  }
} //**********************************
```



Fclk = 1 МГц

Задержка выполнения действия
  в ISR_INT0:   23 мкс, 23 м.ц.
  в ISR_INT1:   33 мкс, 33 м.ц.
Время выполнения
  в ISR_INT0:   48 мкс, 48 м.ц.
  в ISR_INT1:   85 мкс, 85 м.ц.
Период основного цикла 103540 мкс

В МК более поздних разработок (*ATmega48*/*88*/*168*/*328* и др.) входы прерывания *INTx* имеют расширенный набор выявляемых событий, все выводы параллельных портов оснащены функцией выявления изменения состояния [*PCINT0…23*].

Отличия в работе входов прерывания *INTx* сводятся к использованию ранее зарезервированной комбинации битов выбора события (*ISCx1* = 0, *ISCx0* = 1) для выявления любого изменения вывода, а также к смене символьных имен регистров [*MCUCR, GIFR, GIMSK*] на соответствующие им [*EICRA, EIFR, EIMSK*].

Механизм аппаратного выявления изменения состояния входа PCINTn выявляет изменение сигнала на выводе МК, но не выявляет тип (фронт или срез). В МК указанных выше серий такой функцией оснащены 24 вывода трех параллельных портов. Прерывания представлены тремя векторами PCINT0,1,2, один вектор на группу из 8 входов, общим на группу является и флаг PCIFR. Маскирование прерываний выполняется как общее на группу (например, PCICR.PCIE0 – для PCINT0…7), так и индивидуальное (PCMSK0.PCINT0).

Рассмотрим подробнее состав и назначение регистров механизма PCINTn.

Регистр PCICR [Pin Change Interrupt Control Register] содержит три бита: PCIE0 [.. Interrupt Enable] – разрешение прерывания при изменении состояния на 8 выводах PCINT0…7, PCIE1 – на выводах PCINT8…15, PCIE1 – на выводах PCINT16…23.

Регистр PCIFR [Pin Change Interrupt Flag Register] содержит три бита PCIF0 [Pin Change Interrupt Flag] – флаг прерывания по выводам PCINT0…7, PCIF1 – на выводах PCINT8…15, PCIF2 – на выводах PCINT16…23.

Регистры PCMSK0, 1, 2 [Pin Change Mask Register] содержит биты PCINT0..7 [Pin Change Enable Mask], PCINT8..15, PCINT16..23 соответственно для индивидуального разрешения прерываний по каждому выводу.

Данный механизм ориентирован для сравнительно медленных сигналов. Для уточнения причины прерывания: фронт или срез и на каком из 8 выводов группы произошло изменение состояния, необходимо иметь память предыдущего состояния текущего порта и затратить время на анализ. Эти действия не нужны, если разрешен только один из 8 выводов данной группы и не имеет значение тип события (фронт или срез), тогда временные затраты не отличаются от механизма INTx.

Часть 5. Таймеры/счетчики в задачах формирования и измерения импульсов

5.1. Задачи формирования и измерения временных интервалов

В большинстве задач управления время выступает как важный фактор. Период исполнения основного цикла обычно достаточно сильно варьируется и не может служить надежным счетчиком времени. В то же время в ряде задач необходимо достаточно строго выдерживать интервалы времени.

1) Формирование временных интервалов:

а) синхронизация исполнения подпрограмм по времени;

б) формирование на выходах МК одиночных и периодических импульсов с постоянными и переменными параметрами – частота/период T/f, длительность импульса/ паузы ti/tп, фаза импульса φ, число импульсов n (рис. 5.1).

2) Измерение временных интервалов событий на входах МК – частоты/периода f/T, длительности импульса/ паузы ti/tп, фазы импульса φ, числа импульсов n.

Задача типа 1.а требует формирования данных об изменении даты/времени и периодического сканирования этих данных для выявления заданного момента. Период сканирования не должен превышать точность задания момента времени.

Задача формирования однократного импульса состоит из формирования фронта (перевод выхода в высокое состояние), формирование длительности $t_и$ (например, программной задержкой) и формирования среза (перевод выхода в низкое состояние).

Формирование последовательностей периодических импульсов на выходах МК сводится к выполнению повторяющихся циклов и традиционно называется *генерацией импульсов*. Такие задачи возникают при управлении электроприводами, преобразователями электрической энергии, тактировании последовательных интерфейсов.

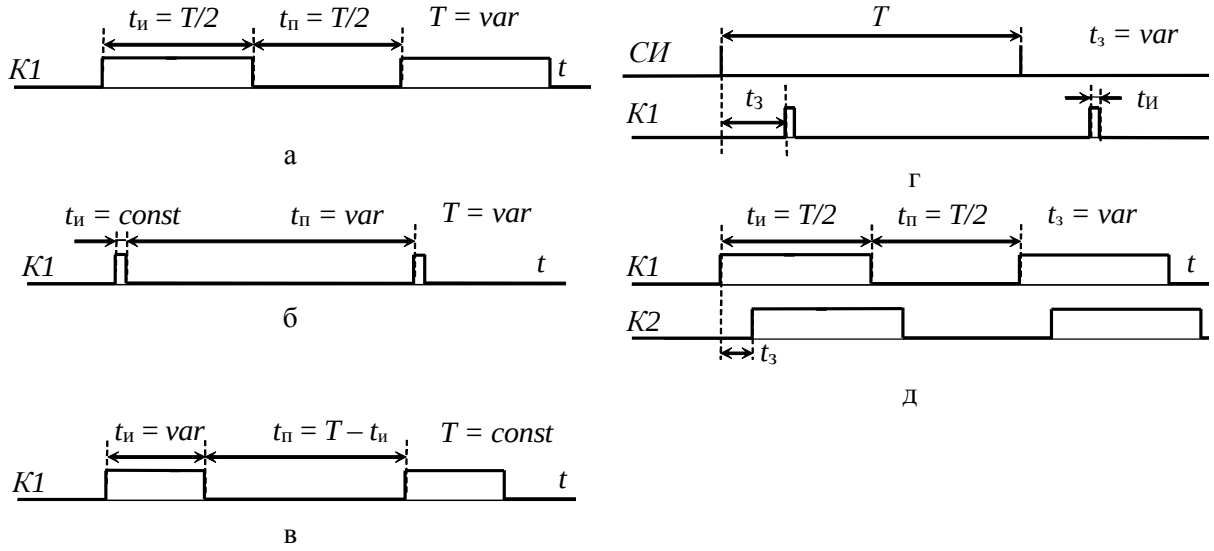


Рис. 5.1

Генерация импульсов состоит из формирования фронтов и срезов на выходах МК и формирования временных интервалов между фронтами и срезами (длительность импульса $t_и$ и длительность паузы $t_п$). При малых значениях периода или длительности импульса (менее тысячи машинных циклов) необходимо учитывать время программного формирования фронта и среза.

Импульсы одноканальные (рис. 5.1, а-в) состоят из интервала с высоким уровнем

(собственно импульс длительностью $t_и$) и с низким (пауза длительностью $t_п$), период сигнала $T = t_и + t_п$, частота $f = 1/T$.

Обычно варьируется один, реже – два из перечисленных параметров.

Вариция периода (частоты) T = 1 / f = var при постоянной абсолютной tи = const (рис. 5.1, б) или относительной tи/T = const (рис. 5.1, а) ширине импульса называется частотной модуляцией (ЧМ). Ее широко применяют для регулирования в преобразователях электрической энергии, работающих на резонансный контур.

Вариация ширины импульса tи = var при постоянной частоте (рис. 5.1, в) называется широтно-импульсной модуляцией (ШИМ). Ее широко применяют для плавного регулирования импульсных преобразователей электрической энергии.

Общий алгоритм генерации одноканальных импульсов представлен на рис. 5.2, а. Здесь подразумевается бесконечная генерация, реализация заданного закона вариации временных интервалов не рассматривается.

Способы формирования задержки могут быть самыми разными, основные будут расмотрены ниже. Все способы используют программный или аппаратный отсчет тактов фиксированной длительности.

Импульсы многоканальные (рис. 5.1, г-д) кроме длительностей импульса и паузы для каждого канала характеризуются задержкой или сдвигом фазы между каналами.

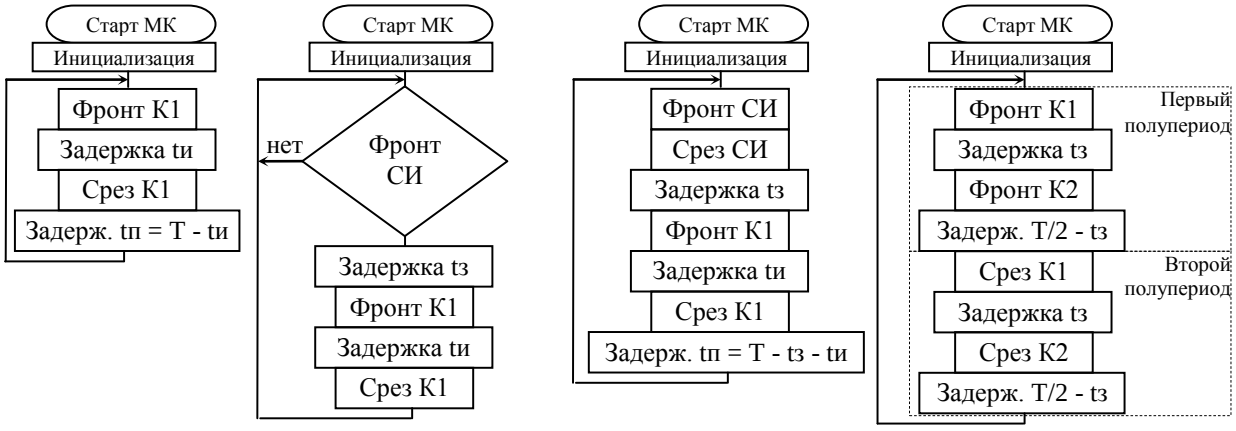
На рис. 5.1, г фронт импульсов К1 имеет задержку tз по отношению к фронту синхроимпульсов СИ, период импульсов обычно незначительно меняется.

В случае внешних синроимпульсов алгоритм генерации должен «ловить» фронты СИ и формировать задержки для генерации фронта (tз) и среза (tи) импульсов К1 (рис. 5.2, б). Период синхроимпульсов обычно варьируется незначительно.

Генерация импульсов, синхронизируемых внешним периодическим сигналом находит прменение в регулируемых выпрямителях и инверторах, ведомых сетью.

В случае внутренних СИ заданы период Т (или частота), задержка tз и длительность импульсов tи, алгоритм генерации представлен на рис. 5.2, в.

Импульсы на рис. 5.1, д обычно имеют одинаковую, но варьируемую частоту (период) и постоянную или варьируемую задержку tз между фронтами импульсов К1 и К2. Величина задержки пропорциональна периоду и измеряется в его долях значением угла сдвига φ = tз / T. Алгоритм генерации представлен на рис. 5.2, г.



а б в г

Рис. 5.2

Для управления мощными преобразователями электрической энергии широко используется мостовая схема, она требует два канала импульсов, сдвинутых по отношению к друг другу строго на пол периода. Для управления трехфазными электродвигателями требуется три/шесть каналов импульсов со сдвигом на одну треть/шестую часть периода.

Обращаем внимание, что варьирование частоты генерируемых импульсов в МК практически всегда заменяется варьированием периода. При незначительном диапазоне изменения (fmax - fmin) * 2 / (fmax + fmin) < 0.3 – 0.7 это не приводит к значительным погрешностям, при большем следует каждый раз вычислять значение периода для варьируемого значения частоты *Ti = 1 / fi*.

Начало последовательности импульсов привязано к внутреннему времени или логике программы, либо к заданному изменению входного сигнала (фронт или срез). Длина последовательности импульсов задается либо числом импульсов *n*, либо внешними условиями, либо таймером. В первом случае удобно организовать цикл из *N* периодов. Во втором и третьем при запуске устанавливается флаг генерации *f_gen*, разрешающий генерацию, его сброс обеспечивается фронтом или срезом внешнего сигнала или таймером.

Задачи измерения параметров внешних сигналов разделяются на задачу измерения длительности импульса или периода и на задачу измерения частоты. В любом случае необходимо выявлять фронты и/или срезы импульсов.

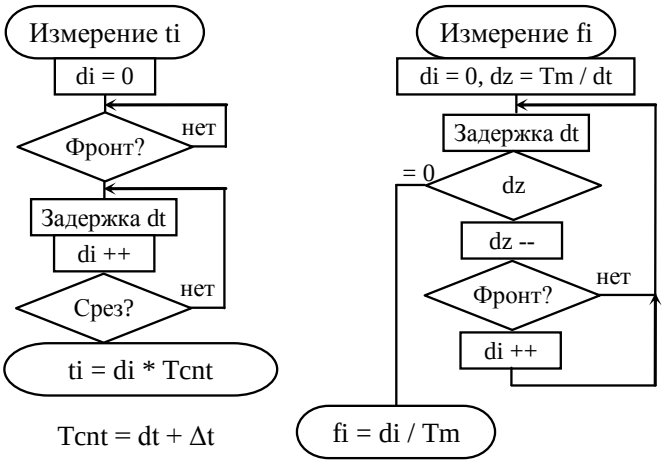
Измерение длительности импульса *ti* или периода *Ti* кроме выявления фронтов и/или срезов требует измерения длительности промежутка между этими двумя событиями. Для измерения длительности следует тактировать промежуток времени импульсами, период которых *Tcnt* много меньше интервала и вести счет этих импульсов *di = ti / Tcnt* или *dper = Ti / Tcnt*. Точность измерения в этом случае будет не лучше периода тактирования *Tcnt*. На рис. 5.3, а представлен алгоритм измерения длительности импульса ti, здесь интервал Tcnt состоит из задержки dt и команд выполнения цикла (di++, Срез?). Для измерения периода Ti достаточно заменить выявление среза на выявление фронта или наоборот.

Измерение частоты импульсов fi сводится к подсчету числа импульсов *di* за фиксированный интервал времени *Tm*: *fi* [Гц] = *di* / *Tm* [имп/с]. Здесь МК приходится решать параллельно решению две задачи:

- формирование интервала измерения *Tm = const*,
- выявление фронтов (или срезов) импульсов на входе МК и их подсчет суммированием *(di++)*.

По окончании очередного интервала *Tm* значение счетчика импульсов *di* копируется в ячейку результата (при необходимости масштабируется), счетчик импульсов *di* обнуляется для нового цикла измерений.

Выбор интервала измерения *Tm* определяет дискретность измерения частоты – чем больше импульсов за интервал измерения, тем меньше погрешность измерения. Точность измерения сильно зависит от точности задания и стабильности интервала измерения. При высоких частотах измеряемого сигнала желателен аппаратный счет импульсов.



а б

Рис. 5.3

На рис. 5.3, б представлен алгоритм измерения частоты. Здесь интервал Tm сформирован из фиксированного числа задержек dt, длительность dt должна быть много меньше периода измеряемой частоты.

Измерение сдвига фазы между импульсами одинаковой частоты f_0 (рис. 5.1, д), например, между меандрами напряжения и тока резонансного контура, сводится к измерению задержки между фронтами $t_з$ (или срезами) и периода T_0 импульсов на входах МК. Значение фазы линейно зависит от частоты $\varphi = f_0 * t_з = t_з / T_0$, поэтому при малых изменениях частоты ее можно не измерять.

5.2. Принципы программого формирования/измерения временного интервала

Программная реализация таких задач базируется на *программных задержках*, формируемых, как правило, с использованием команды *nop*, и *циклах* с фиксированным или варьируемым числом повторений. Такой способ вполне удовлетворителен для коротких временных интервалов или при монопольном использовании процессора только для этой задачи.

Небольшие фиксированные задержки (единицы – десятки машинных циклов) формируются из требуемого количества «пустых» операторов *nop*. Напомним, что эта команда присутствует в системе команд любого встраиваемого МК и выполняется за один машинный цикл. Число операторов $n = t_з / T_{CLK} = t_з * f_{CLK}$, где $t_з$ – время задержки; T_{CLK} и f_{CLK} – период и частота тактирования ядра МК.

Значительные по длительности задержки формируются из простых или вложенных циклов, содержащих требуемые по длительности малые фиксированные задержки из тех же операторов *nop*. В этом случае расчет длительности должен учитывать время выполнения всех операторов, составляющих цикл. Изменение числа циклов позволяет получать варьируемые по длительности задержки.

В библиотеках среды программирования на языке C/C++, как правило, есть готовые функции delay_us/delay_ms(unsigned int), которые реализуют формирование задержек в микро/миллисекундах. Входной параметр должен быть константой, что при заданной частоте тактирования (в настройках транслятора) позволяет транслятору рассчитать требуемое число машинных циклов.

При фиксированной частоте тактирования МК не сложно самостоятельно написать и отладить подпрограмму, реализующую задержку с варьируемой длительностью.

Недостатком таких способов формирования временных интервалов является нерациональное использование производительности МК (в ходе отработки задержки процессор не выполняет других задач).

Кроме «пустых» действий задержки можно заполнять и «полезными» кусками программ или подпрограмм, но в этом случае сложнее синхронизировать программные действия из-за наличия ветвлений, приводящих к непостоянному времени выполнения подпрограмм.

Большинство управляющих программ состоят минимум из одного «основного» цикла, в котором обслуживаются самые «медленные» задачи. Для обслуживания более «быстрых» задач внутри основного цикла организуются вложенные циклы. Период основного цикла и периоды вложенных можно сделать кратными между собой.

Для формирования импульсов с постоянной частотой со скважностью $Q = T/tu = 2$ на выходе *Px.y* (рис. 5.1, а) достаточно через равные промежутки времени $T/2$ инвертировать выход:

```
while(1) {
  PORTx ^= (1<<y); // инверсия выхода Px.y
  _delay_ms(200); // задержка на пол периода
}
```

Для генерации импульсов с варьируемой частотой используем принцип заполнения периода относительно короткими интервалами dT. Если заданы диапазон частот fmin … fmax и число квантов N, то короткий интервал должен быть равен или кратен dT = (Tmax – Tmin) / N = (1/fmin – 1/fmax) / N. Представим его целой константой в микросекундах, для этого при вычислении периода заменим 1/f на 1000000/f. Тогда для формирования импульсов с частотой fset в диапазоне fmin … fmax потребуется формировать задержку T/2 = NPPER * dT:

```
#define fmin 1000
#define fmax 2000
#define N 10
#define fset 1500 /* Заданное значение частоты */
#define dT ((1000000/fmin – 1000000/fmax)/N)
// (1000 – 500)/10 = 50
#define NPPER (1000000/fset/2/dT)
// 1000000/1500/2/50 = 6
unsigned char n;
while(1) {
  PORTx |= (1<<y); // уст выход Px.y
  n = NPPER;
  while (n) {n--; _delay_us(dT); } // задержка на пол периода
  PORTx &= ~(1<<y); // сбр выход Px.y
  n = NPPER;
  while (n) {n--; _delay_us(dT); } // задержка на пол периода
}
```

Заметим, что данный пример иллюстрирует принцип формирования заданной частоты, но не может «на ходу» изменять ее. Это связано с тем, что подпрограмма `_delay_us(dT)` в качестве аргумента `dT` требует константы (оганичение транслятора).

Этот и другие примеры из этого раздела представлены в папке Primer \ p2 Interrupt & Timer-Counter \ Progr_delay.

Проект с примером находится в папке fvar_dt в файле fvar_dt.aps.

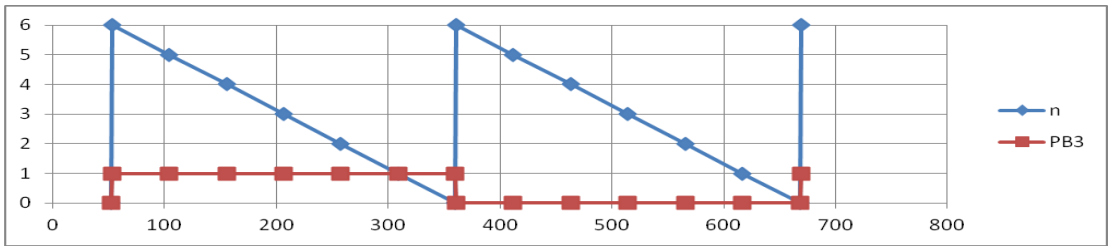


Рис. 5.4

При тестировании в симуляторе AVR Studio поставим точки прерывания (<F9>) на строки изменения состояния выхода Px.y (PB3), перед каждым очередным запуском (<F5>) просматриваем и обнуляем значение StopWatch в окне Processor. При fclk = 8 МГц получаем значение полупериодов 307.63 + 308.38 мкс, период 616.0 мкс, частота 1623 Гц (погрешность 8%). Временная диаграмма представлена на рис. 5.4.

Если требуется варьировать частоту на каждом периоде, закон изменения частоты (точнее, периода), должен находиться в цикле. Например, для формирования 10 импульсов с периодом 400, 380, 360. … 220 мкс, шаг изменения полупериода 10 мкс:

```
unsigned char n_imp = 10, npper = 20, n;
while(n_imp) {
  n_imp--;
  PORTx ^= (1<<y); // инверсия выхода Px.y
  n = npper;
  while (n) {n--; _delay_us(10); } // задержка на пол периода
}
```

Вариант одноканальной ШИМ требует отдельного формирования длительности импульса ti = var и длительности зависимой паузы tп = T – ti, T = const. Например, функция для формирования n_imp импульсов с варьируемой длительностью импульса от 10 до 990 мкс, задаваемой переменной nti от 1 до 99 соответственно и постоянным периодом (задается переменной nT в том же масштабе):

```
void gen_PWM(unsigned char n_imp, unsigned char nti, unsigned char nT) {
  unsigned char n;
  while(n_imp) {
    n_imp--;
    PORTx |= (1<<y); // фронт импульса на выходе Px.y
    n = nti;
    while (n) {n--; _delay_us(10); } // задержка на импульс
    PORTx &= ~(1<<y); // срез импульса на выходе Px.y
    n = nT - nti;
    while (n) {n--; _delay_us(10); } // задержка на паузу
  }
}
```

При тестировании в симуляторе AVR-Studio (папка pwm_delay) с fclk = 8 МГц время выполнения функции gen_PWM(5, 15, 100); составило 5138.5 мкс, период импульса 1027.25 мкс (погрешность 2.7%), длительность импульса 154.63 мкс (погрешность 3%). Для функции gen_PWM(5, 85, 100); время выполнения 5138.5 мкс, период импульса 1027.25 мкс (погрешность 2.7%), длительность импульса 872.13 мкс (погрешность 2.2%). Для функции gen_PWM(5, 5, 10); время выполнения 526.0 мкс, период импульса 104.75 мкс (погрешность 4.7%), длительность импульса 52.13 мкс (погрешность 2.1%).

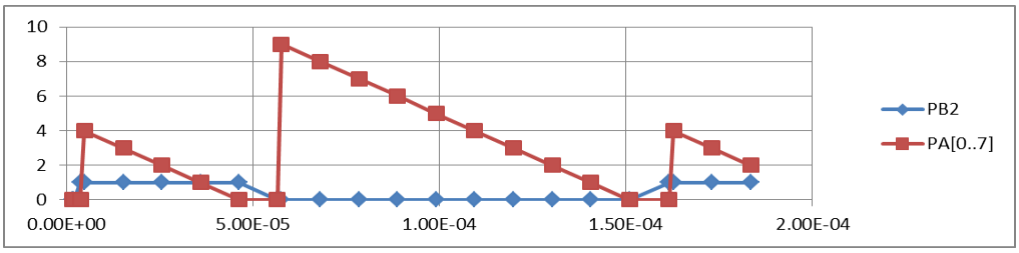


Рис. 5.5

На рис. 5.5 представлена временная диаграмма выполнения функции gen_PWM(5, 5, 15) с программным выводом переменной n на порт A. В системе моделирования Proteus VSM (файл pwm_8515.dsn) значения выхода PB2 и числа на выходах PA[0..7] выведены в окно графика Digital analysis, экспортированы в текстовый файл ddd.dat, затем импортиованы в Excel с разделителем «запятая», значения PB2 преобразованы SLO = 0, SHI = 1, построен график (Вставка – Точечная). Конечно, быстрее и проще получить лог-файлы для портов A и B в AVR-Studio, но их постобработка в Excel'е несколько длинее.

Измерение длительности однократного импульса на входе Pxy состоит из выявления фронта импульса, затем – выявление среза импульса с одновременным подсчетом времени. Первое действие состоит из частного цикла чтения входа и сравнения с предыдущим состоянием, если предыдущее состояние 0, а текущее 1, фронт выявлен. Второе действие аналогично, но срезу соответствует предыдущее состояние 1, текущее 0. Кроме того, при выявлении среза нужно подсчитывать число циклов сканирования – это и будет длительность импульса, выраженная в длительностях этого второго цикла.

```
//измерение длительности однократного импульса на входе PB0
static unsigned char n_cycle=0, tek=0, pred=1;
DDRA=0xff;
do { pred=tek; tek=(PINB & (1<<0)); }
while(pred || !tek);//ожидание фронта

do { pred=tek; tek=(PINB & (1<<0)); n_cycle++; PORTA= n_cycle; }
while(!pred || tek);//ожидание среза  и подсчет циклов
```

В данном примере (ismer_dlit_imp) при частоте тактирования 10 МГц длительность первого цикла составляет 0.8 / 1.0 мкс, второго цикла – 0.9 / 1.3 мкс, при подаче на вход импульса длительностью 10 мкс `n_cycle` = 11, что соответствует 11 * 0.9 + 1.0 = 10.9 мкс.

Измерение периода отличается от измерения длительности импульса только тем, что надо мерить длительность интервала между последующими фронтами или срезами.

Такой способ измерения нельзя назвать удобным, так как шаг измерения зависит от разрядности переменных и транслятора, то есть требует тщательной подгонки кода под требуемую точность. Кроме того, подобный способ выявления фронта/среза опасен зацикливаниями в отсутствии импульса.

Решать подобным способом задачу измерения частоты вовсе неудобно. Гораздо лучше для счета импульсов использовать вход прерывания INT0, а интервал измерения по-прежнему формировать на задержках. Подобный подход уже рассматривался в примере раздела 4.2 (Папка Int0cnt).

При низких измеряемых частотах $f_и$ и высоких требованиях к дискретности и точности целесообразно измерение частоты заменить измерением периода $T_и$ этого же сигнала и программным расчетом значения частоты $f_и = 1/T_и$.

5.3. Таймер/счетчик с прерыванием по переполнению

В многозадачном режиме процессора эффективность и точность чисто программной реализации временных интервалов быстро снижается. В этом случае гораздо выгоднее использование специальных аппаратных узлов – *таймеров/счетчиков* – имеющихся сейчас практически в любом современном МК. Основу такого узла образует аппаратный счетчик, как правило, работающий на инкремент (увеличение на 1) по тактовым импульсам. В режиме *таймера* такты получают из импульсов внутренней синхронизации МК, в режиме *счетчика* – из внешних импульсов на выделенном входе МК. В момент переполнения счетчика (перехода от максимального значения TOP к нулю) устанавливается специальный *флаг переполнения,* он может порождать запрос прерывания по вектору переполнения.

Текущее значение аппаратного счетчика доступно по чтению и записи в любой момент времени. Это позволяет следить за счетом импульсов и управлять ходом счета.

Режим таймера в задачах формирования временных интервалов удобно использовать, управляя длительностью интервала переполнения, в задачах измерения длительности удобно использоваь чтение значения счетчика.

Режим счета внешних импульсов удобно использовать в задачах измерения частоты - аппаратное выявление фронтов и/или срезов и счет.

В микроконтроллерах серии *AVR* используется несколько разновидностей таймеров/счетчиков. В одном МК находится от одного до шести таймеров/счетчиков различного типа TC*x* (*x* = 0, 1, … 5). Они различаются разрядностью аппаратного счетчика (8 или 16 бит) и набором дополнительных функций.

Основным элементом каждого ТС является *базовый счетчик*, который ведет счет на сложение (инкремент) в регистре TCNTx. При его переполнении формируется соответствующий запрос прерывания. Счетные (тактовые) импульсы в режиме таймера формируются внутри МК (непосредственно сигнал синхронизации МК частотой fclk или через предделитель), в режиме счетчика – по фронту или срезу внешнего сигнала на выделенном для этого входе *Tx*.

При наличии дополнительных регистров базовый счетчик может выполнять *дополнительные функции*: *захвата, сравнения или широтно-импульсного модулятора* (ШИМ) и *счета реального времени*.

В состав МК *at90s8515* входят два таймера/счетчика: 8-разрядный без дополнительных функций *TC0* и 16-разрядный с дополнительными функциями захвата и сравнения/ШИМ *TC1*. В боле современных моделях с подобным числом выводов *ATmega48/A(88-168-328)* присутствуют три таймера/счетчика, из них два 8-разрядных и один 16-разрядный, они имеют 6 блоков сравнения.

Структурная схема таймера/счетчика *TC0* представлена на рис. 5.6. Она состоит их программно-доступных элементов и скрытых от программиста аппаратных узлов. С точки зрения программиста таймер/счетчик *TC0* представлен четырьмя регистрами или отдельными битами РСФ, доступных по 8-разрядной шине данных по чтению и записи:

– регистр управления *TCCR0* [*Timer/Counter Control Register 0*] содержит три бита CS02…CS00 [Clock Select] выбора источника тактирования (см. Табл. 5.1);

– счетный регистр *TCNT0* [*Timer/CouNTer 0*] – основной элемент таймера/счетчика, по приходу очередного тактового импульса его содержимое инкрементируется, то есть выполняется счет на увеличение с учетом переполнения;

– бит переполнения *TOV0* [*Timer OVerflow 0*] регистра флагов таймеров *TIFR* [*Timer Interrupt Flag Register*];

– бит разрешения прерывания по переполнению *TOIE0* [*Timer Overflow Interrupt Enable 0*] регистра масок прерываний от таймеров *TIMSK* [*Timer Interrupt MaSk Register*].

Тактовый сигнал микроконтроллера *CLK* поступает в пересчетную схему ПС (*prescaler*), представляющую собой десятиразрядный счетчик, где выполняется деление тактового сигнала на 8, 64, 256 и 1024. Сигналы с четырех выходов пересчетной схемы поступают в схему управления СУ (схема выделения фронта или среза по входу *T0* и мультиплексор). Эти же сигналы используются для тактирования таймера /счетчика ТС1.

В схему управления поступают также тактовый сигнал *CLK* и сигнал из внешнего источника, принимаемый на вход *T0* (совмещенный в МК 8515 с выводом *PB0*).

Схема управления в зависимости от комбинации состояния разрядов *CS00, CS01, CS02* регистра управления *TCCR0* передает один из 6 поступающих сигналов на счетный вход базового счетчика *TCNT0*, ведущего счет на сложение. Сигналы, используемые в счетчике *TCNT0* при различных комбинациях значений в разрядах регистра *TCCR0*, указаны в табл. 5.1.

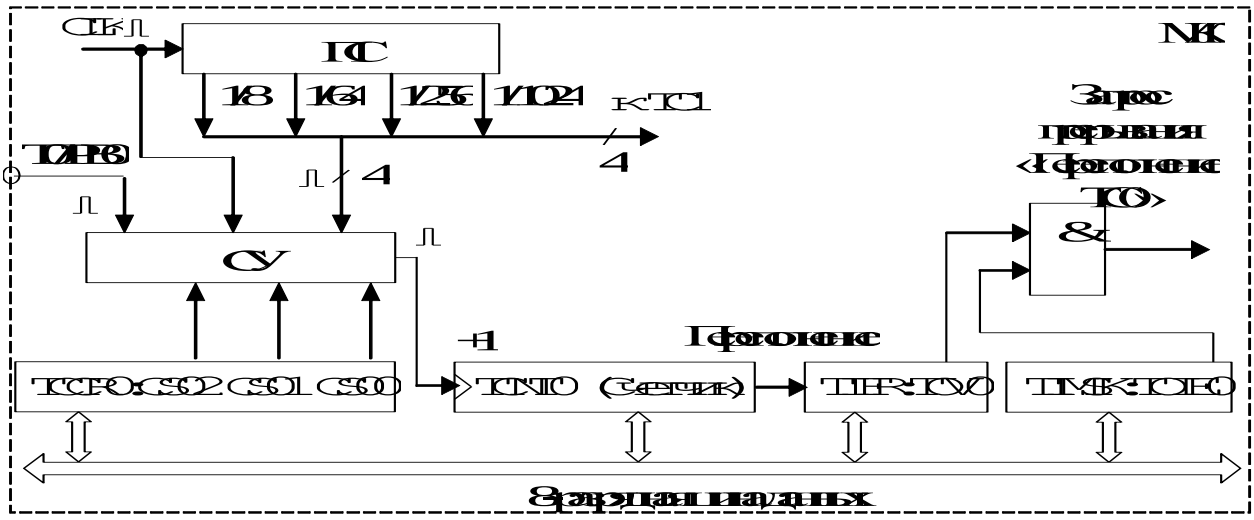


Рис. 5.6

Таблица 5.1. Код выбора тактирования ТС0

CS02	CS01	CS00	Сигнал
0	0	0	Нет
0	0	1	CLK
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	Отрицательный фронт (срез) на входе Т0
1	1	1	Положительный фронт на входе Т0

После достижения счетчиком *TCNTx* максимального значения *TOP* (равного $2^8 - 1 =$ $= 0xff = 255$) происходит переполнение *[overflow]* его разрядной сетки - переход к значению *BOTTOM* = 0. При переполнении устанавливается в единичное состояние разряд

TOV0 регистра *TIFR* и, при единичном состоянии разряда *TOIE0* регистра *TIMSK*, в блок прерываний поступает запрос прерывания по переполнению таймера/счетчика ТС0.

Сброс разряда *TOV0* в нулевое состояние осуществляется аппаратно при выполнении первой команды ППОП (*ISR*) по переполнению ТС0 или программно записью логической единицы (!) непосредственно в разряд *TOV0*.

Рекомендации по примененииюе базового таймера-счетчика.

1. Использование переполнения в режиме таймера для формирования временных интервалов.

После запуска счета в режиме таймера с выбранной частотой тактирования f_{TC0} (биты *CS02…00* в регистре *TCCRx(B)*) через каждые *TOP* периодов тактирования будет происходить переполнение таймера. Длительность интервала переполнения составляет $t\_ov = TOP / f_{TCx}$.

Если разрешить прерывания по переполнению, то в подпрограмме обслуживания можно вести счет переполнений (интервалов *t_ov)* и выполнять программные действия с интервалом равным или кратно большим *t_ov*.

Можно уменьшить длительность интервала переполнения до значения $t\_ov = n / f_{TCx}$, где $n < TOP$, если сразу после переполнения или перед запуском таймера (если счет был остановлен) изменить содержимое *TCNTx* на значение $TOP+1 - n$. Эквивалентно записи *" –n"*, если переменная *n* объявлена как беззнаковая и ее разрядность совпадает с разрядностью счетного регистра.

Для *генерации периодических импульсов* на произвольном выходе/ах МК удобно заставить таймер аппаратно вести отсчет длительности очередного интервала между фронтами и срезами, а в прерываниях по переполнению программно формировать эти фронты и срезы и загружать в счетный регистр число для формирования следующего интервала. Выбор события (фронт / срез на том или ином выходе) соотвествующей длительности интервала удобно организовать по программному счетчику числа переполнений *static cnt_ov*, число состояний которого равно числу формируемых событий.

Для генерации простого сигнала, период которого состоит из импульса и паузы, достаточно различать фронт и срез, то есть всего 2 состояния счетчика *cnt_ov*. Для формирования несинхронных импульсов на двух выходах может потребоваться до 4 состояний счетчика *cnt_ov* и так далее. Если длительность очередного интервала постоянна и мала (менее 100 машинных циклов), удобнее сформировать ее в прерывании (с использованием *nop*'ов).

При каждом входе в подпрограмму прерывания следует по значению *cnt_ov* выбрать текущую ветвь, в ней записать значение, пропорциональное длительности ближайшего интервала в регистр TCNTx, сформировать фронт или срез на соответствующем выводе и изменить значение счетчика *cnt_ov* на следующее.

На рис. 5.7 приведена временная диаграмма формирования одноканальных импульсов на выходе Px.y с использованием 8-битного таймера (TCNTx) и прерывания по переполнению (TOVx). Для формирования одноканальных импульсов можно вместо переменной *cnt_ov* использовать значение PORTx.y, это уменьшит объем и длительность подпрограммы прерывания.

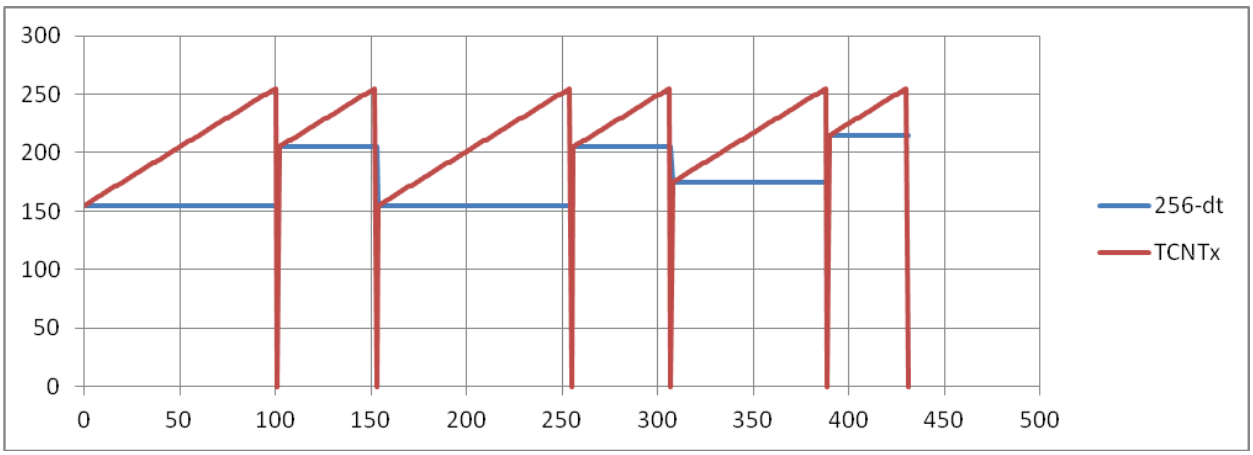


Рис. 5.7

Для генерации импульсов с переменными временными параметрами (период, длительность, фаза и пр.) выбор частоты тактирования $f_{TCx} = f_{CLK} / N$ (N = 1/8/64/256/1024) определяет минимальный шаг изменения временного интервала $T_{TCx} = Tclk * N = N / fclk$, разрядность таймера/счетчика – максимальное число шагов по времени (TOP + 1). Например, при максимальной для некоторых новых моделей МК AVR-8 частоте тактирования fclk = 20 МГц, N = 1 шаг вариации интервала составит $T_{TCx} = Tclk = 1 / fclk$ = 50 нс, максимальная длительность интервала – ti = Tclk * (TOP + 1) = 20 нс * 256 = 5.12 мкс. При fclk = 8 МГц и N = 1 шаг вариации - T_{TCx} = 125 нс, максимальная длительность ti = 32 мкс; при N = 1024 T_{TCx} = 128 мкс, ti = 32.768 мс.

Такой способ генерации импульсов имеет явные преимущества перед способом с задержками на циклах процессора и может быть рекомендован при частоте переполнений, не превышающих нескольких килогерц, иначе прерывания излишне тормозят выполнение основного цикла и растет погрешность в отработке дискретности временных параметров генерируемых импульсов (см. ниже).

Следует избегать выполнения расчетов в подпрограмме обслуживания прерывания. Переменные для загрузки изменяющихся значений в регистр *TCNTx* должны вычисляться в основном цикле в момент изменения задания.

Для формирования *программных таймеров*, синхронизирующих работу отдельных модулей (функций) основного цикла удобно в прерывании по переполнению таймера, настроенному на период, равный шагу (кванту) программного таймера, устанавливать флаг, например, *flag1ms*. В подпрограмме основного цикла, обслуживающей программный таймер, организуется проверка установления флага. При выявлении этого события флаг *flag1ms* сбрасывается и выполняется инкремент программного таймера.

2. Использование счетного входа *Tx* для подсчета внешних импульсов.

Этот способ является альтернативой счету импульсов с использованием входа прерываний, основное преимущество счетного режима таймера – снижение частоты прерываний в (*TOP* + 1) раз.

Режим выбирается с помощью битов *CS02-CS00* регистра управления данного таймера счетчика *TCCRx(B)* с комбинацией (*0x07*) для выбора фронта [*rise*] или среза (*0x06*) [*fall*] внешнего импульса. Требование к внешним импульсам – длительность импульса и паузы должна быть не мене 4 периодов тактирования процессора T_{CLK}.

Если существует вероятность переполнения таймера/счетчика, следует разрешить прерывания по переполнению, в подпрограмме обслуживания которого вести счет переполнений *cnt_sig + (TOP+1) --> cnt_sig*, размерность переменной должна быть

соответствующей.

Для измерения частоты внешних импульсов потребуется два таймера/счетчика. Первый в счетном режиме, импульсы поданы на счетный вход *Tx*. Второй будет формировать период измерения (в режиме таймера), по его переполнению следует считывать значения регистра *TCNTx* первого таймера/счетчика в переменную *cnt_sig*, затем обнулять его.

3. Использование таймера для измерения длительности временных интервалов.

Выбор периода тактирования в режиме таймера T_{TCx} определяет шаг (дискретность) измерения, разрядность таймера/счетчика – максимальное измеряемое число тактов счетчика (*TOP+1* без программной поддержки переполнения).

При измерении длительности однократных временных интервалов программа должна выполнить два действия: выявить начало интервала и запустить счет таймера (предварительно обнуленного), затем выявить конец временного интервала, после чего остановить тактирование таймера и считать полученное число тактов в *cnt*, при необходимости подготовки следующей измерительной процедуры - обнулить *TCNTx*. Абсолютное время можно вычислить по формуле $t = cnt / f_{TC}$.

При измерении длительности внешнего импульса можно использовать для выявления фронта и среза импульса вход прерывания *INTx*, настроенный сначала на фронт, затем на срез.

При измерении периода внешнего сигнала достаточно настроить прерывание только на срез или на фронт, по любому из них сохранять значение *TCNTx* в переменной, затем обнулять *TCNTx*.

Однозначная связь значений периода и частоты внешнего сигнала f = 1/T позволяет выбирать способ измерения, «удобный» для МК. При высокой частоте период слишком мал, удобнее заменить измерение периода измерением частоты и последующим вычислением значения периода. Правда, в этом случае растет задержка измерения. Аналогично, при измерении частоты слишком низкочастотного сигнала проще заменить его измерением периода, затем вычислить значение частоты, здесь задержка измерения уменьшается.

Примеры использования таймера\счетчика с прерыванием по переполнению.

Пример генерации с ЧМ.

Сформировать бесконечную последовательность импульсов, имеющих вид (рис. 5.8, *а*) с частотной модуляцией, заданной законом $T = var$ = 100, 99, … 50, 100, 99, … мкс и $\tau_И = const$ = 5 мкс.

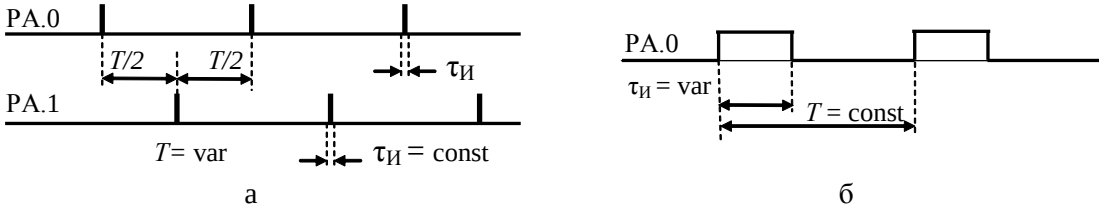


Рис. 5.8

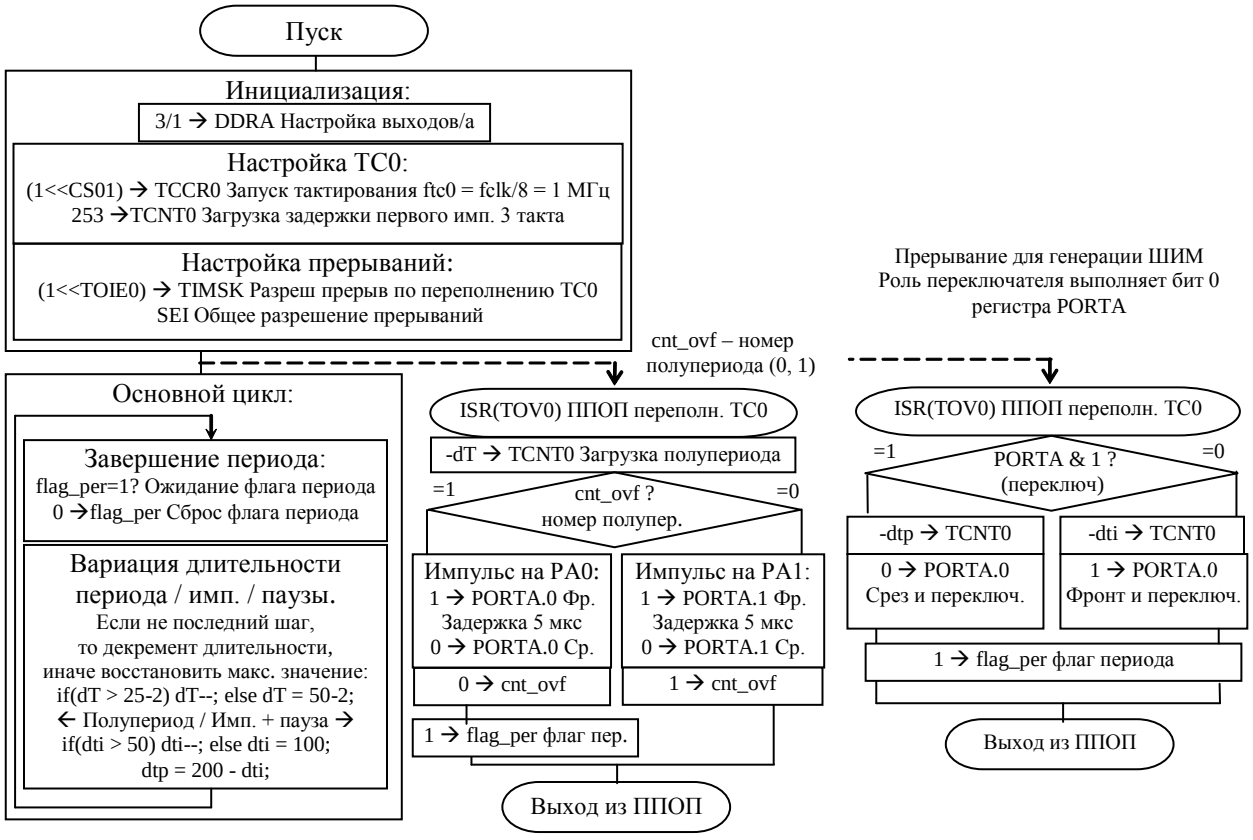
Пример генерации с ШИМ.

Сформировать бесконечную последовательность импульсов, имеющих вид (рис. 5.8, *б*) с широтной модуляцией, заданной законом $\tau_И = var$ = 100, 99, … 50, 100, 99, … мкс и *T*

= *const* = 200 мкс.

На рис. 5.9 представлена блок-схема алгоритма формирования последовательности импульсов ЧМ и ШИМ. Оба алгоритма имеет похожую структуру и отличаются лишь в деталях. Изменения состояния выходов выполнятюся программно в подпрограмме обслуживания прерывания (ППОП) по переполнению таймера ТС0, а в фоновой программе выполняется вариация уставки значения периода или длительности импульса.

На рис. 5.9, а представлены этапы инициализации, основного цикла (с вариациями для ЧМ и ШИМ), пунктирной стрелкой указана обработка прерывания таймера для ЧМ, на рис. 5.9, б – прерывание таймера для ШИМ.



а б

Рис. 5.9

На этапе инициализации выполняются: настройка выхода/ов порта, настройка ТС0 на режим таймера с необходимым коэффициентом делителя (регистр *TCCR0*), в регистр данных (*TCNT0*) заносится начальное значение числа, от которого пойдет счет до переполнения и разрешаются прерывания по переполнению ТС0 (регистр *TIMSK* и общее разрешение прерваний).

В прерывании выполняется перезагрузка регистра данных счетчика (*TCNT0*) для формирования следующего временного интервала, после чего выполняется программное изменение соответствующего выхода. Переменная «Переключатель» принимает периодические значения (0, 1, 0, 1, …) и обеспечивает выполнение по одному прерыванию различных действий: формирование импульсов на выходе PA0 или PA1 (прим. 5.1, переменная cnt_ovf), либо формирование импульса или паузы с различными длительностями (прим. 5.2, в роли переменной – бит 0 регистра PORTA).

Аппаратный счетчик *TCNT0* ведет счет на увеличение (инкремент) от нуля или от загруженного в него числа *N*, поэтому длительность формируемого временного интервала

прямо пропорциональна разности между максимальным значением и этим числом: 256 – *N* (см. рис. 5.9), для 8-разрядного числа это значение эквивалентно –*N* (унарный минус).

Для обеспечения требуемой дискретности (шага) изменения периода или длительности импульса необходимо выбрать частоту тактирования ядра МК и коэффициент деления предделителя (пересчетной схемы ПС). Заданная в прим. 5.1 и 5.2 дискретность периода 1 мкс однозначно определяет частоту тактирования таймера/счетчика ТС0: f_{TC0} = 1/(1 мкс) = 1 МГц. Для максимального использования производительности МК частоту тактирования ядра f_{CLK} необходимо выбирать как можно ближе к максимально допустимой по паспорту ($f_{CLK\ MAX}$ = 8 МГц для МК 8515). Требуемый коэффициент деления определяется как $kdiv = f_{CLK\ MAX} / f_{CLK\ TC0}$ и выбирается из перечня доступных (1, 8, 64, 256, 1024). Для нашего задания подходят f_{CLK} = 8 МГц и коэффициент деления *kdiv* = 8.

Проект с примером ЧМ находится в папке T0_gen_tov \ T0_Tvar в файле gen_Tvar.aps. В таблице приведены расчетные и измеренные в симуляторе AVR-Studio значения периода Т в микросекундах. Значение абсолютной погрешности не зависит от длительности периода и определяется задержкой между переполнением и загрузкой нового значения в регистр TCNT0 (21 [мц] или 2.625 мкс), к этому времени значение TCNT0 уже не 0, а 2.

В последних графах таблицы («dT-2» и «Абс») показаны результаты измерения при использовании коррекции значения, загружаемого в счетный регистр.

Табл. 5.2

№ пер.	dT	Расч., мкс	AVR-Studio	Абс.погр.	dT-2	Абс.
1	50	100	103	3	98.88	-1.12
2	49	98	101	3	97.0	-1
3	48	96	99	3	95.13	-0.87
4	47	94	97	3	93	-1
5	46	92	95	3	91	-1
6	45	90	93	3	88.88	-1.12

Проект с примером ШИМ находится папке gen_tov \ T0_pwm в файле T0_pwm_8515.aps, в таблице – результаты тестирования. В качестве переменной-переключателя здесь используется значение бита 0 в регистре PORTA.

Табл. 5.3

№ пер.	dti	Dtp	Ti	tp	T
1	100	100	103.38	101.63	205.00
2	99	101	102.50	102.63	205.13
3	98	102	101.25	103.75	205.00
4	97	103	100.38	104.50	204.88
…					
50	50	150	53.50	151.63	204.88

Пример синхронизации выполнения функции таймером.

Задание: Время выполнения основного цикла варьируется от 0.5 до 4 мс. Действия в подпрограмме 1 должны выполняться 100 раз в секунду (f1), действия в подпрограмме 2 – 10 раз в секунду (f2).

Минимальный кратный шаг времени в этих задачах – 10 мс, настроим один свободный таймер на переполнение с этим периодом. Пусть наш МК имеет частоту

тактирования *fclk* = 8 МГц, *fclk* / *f1* = 80000, максимальный коэффициент деления *kdiv* = 1024, следовательно, переполнение таймера должно происходить через 80000 / 1024 = 78 тактов, для этого достаточно 8-разрядного таймера. В *ISR TOV0* требуется записать в регистр *TCNT0* значение -78, далее вести счет тактов по каждому счетчику cnt_10ms[]--; и при его обнулении установить флаг задачи. Задача в основном цикле проверяет состояние флага, если не установлен, не выполняет никаких действий, иначе сбрасывает флаг и выполняет действия.

```
// ******************************************
#define N_TASK_MAX 2
char flag_task[N_TASK_MAX], cnt_10ms[N_TASK_MAX],
set_task[N_TASK_MAX];
void init_tim_task(void) {
  char i;
  for(i=0; i< N_TASK_MAX; i++) {
    flag_task[i]=0;
    cnt_10ms[i]=0;
  }
  set_task[0]=0; //1 такт
  set_task[1]=9; //10 тактов
  TCNT0=-78;
  TCCR0 = (1<<CS02)|(1<<CS00);
  TIMSK = (1<<TOIE0);
} // ****************************************
ISR(TOV0) {
  char i;
     TCNT0 = -78;
     for(i=0; i<N_TASK_MAX; i++) {
       if(!flag_task[i])
         if(cnt_10ms[i]) cnt_10ms[i]--;
         else cnt_10ms[i]=set_task[i];
     }
} // ****************************************
MAIN_LOOP:
  if(flag_task[0]) {
    flag_task[0]=0;
    subroutine_1();
  }
  if(flag_task[1]) {
    flag_task[1]=0;
    subroutine_2();
  }
```

1. Измерение периода (затем длительности) внешнего сигнала.

Измерить период внешнего сигнала T = 25…100 мкс с дискретностью (временным шагом) 1 мкс. Это означает, что мы должны получить целочисленные значения от 25 до 100. Прежде всего выберем такое тактирование таймера/счетчика, чтобы он изменял свое состояние каждую микросекунду. Тогда разность значений регистра *TCNTx* в моменты времени, соответствующие фронтам (или срезам) входного сигнала будет численно равна значению периода в микросекундах. Считывание значений регистра производится программно в подпрограмме обслуживания прерывания по фронту (срезу) на входе *INTx*.

Для этого измеряемый сигнал надо подать на вывод INT0/PD2 (8515)

Такое решение обеспечивает минимум программных расходов и максимальную точность. Погрешность этого способа связана только с задержкой выполнения команды чтения *TCNTx* по отношению к фронту сигнала, она составляет от 7 до 25 машинных циклов. При частоте тактирования 8 МГц это составит от 0.87 до 3.12 мкс, то есть до 3 % от 100 мкс.

Проект измерения периода находится в папке ismer_T0 \ ism_per в файле ism_per_T0_8515.aps. На этапе инициализации следует настроить вход прерывания *INTx* на фронт и разрешить его прерывания, выбрать тактирование таймера-счетчика, при необходимости настроить другие аппаратный узлы и переменные, последним действием выполнить общее разрешение прерываний. С этого момента потенциально возможен вызов подпрограммы обслуживания прерывания по входу *INTx* (*ISR INTx*). В ней первым действием выполняется чтение регистра *TCNTx* и вычисление значение периода в микросекундах *d_per = TCNTx – pred_cnt*, затем выполняется сохранение текущего значения счетчика для измерения на следующем периоде *pred_cnt = TCNTx*, далее – выход из *ISR.*

Переменная *pred_cnt* должна быть объявлена в *ISR* с параметром *static*, переменная *d_per* – глобального типа, в основном цикле можно выполнять ее чтение. Размер переменных *pred_cnt* и *n_per* должен совпадать с размером регистра *TCNTx* – для 8-разрядного таймера/счетчика требуется тип *unsigned char*, для 16-разрядного – *unsigned int*. Это гарантирует правильность вычислений, невзирая на периодическое переполнение счетного регистра, при условии однократного переполнения. Если возможно более одного переполнения, луше использовать счетчик боьшей разрядности либо программно «увеличить» разрядность. В этом случае надо разрешить перывания по переполнению и вести их счет…

В основном цикле переменная *d_per* может использоваться для чтения, например, в алгоритмах индикации и контроля состояния периода.

Для тестирования в симуляторе AVR-Studio удобно использовать ранее подготовленный файл стимуляции (Таблица 5.4, < PORTD2_per-25-26-99-100.sti >, расположен в папке с пректом). Он содержит данные, имитурующие периодический сигнал для порта D, ножки PD2/INT0 (числа 0x00 и 0x04), шаг по времени задан в машинных циклах. Длительность машинного цикла при частоте тактирования fclk = 8 МГц составляет 1/8 микросекунды, период 25 мкс составляет 200 мц, 100 мкс – 8000 мц. Длительность импульса в данной задаче не имеет значения, выбрана 100 мц, то есть 12.5 мкс.

Для «ручного» тестирования ставим точку прерывания на последнюю строку подпрограммы прерывания ISR(INT0_vect) и наблюдаем состяние переменной *d_per* в окне Watch. Значения Cycle Counter и переменной *d_per* представлены в файле <per.txt> (Табл. 5.4). Значения в первой строке соответствуют первому фронту, период еще не прошел, следующие строки демонстрируют полное совпадение численного значений периода заданному. Для автоматизации сохранения данных можно использовать файл логгирования (Табл. 5.6, < PORTA-8MHz.log >), генерируемый симулятором в сеансе отладки той же длительности, данные (0x19 = 25, 0x1A = 26, 0x63 = 99, 0x64 = 100) получены на порте A за счет вывода значения переменной *d_per* в основном цикле.

Для измерения длительности импульса требуется вычислять разность значений счетного регистра в конце импульса (на срезе) и в начале (по фронту). По сравнению с

предыдущим примером меняется только структура *ISR INTx*. По фронту (когда бит состояния входа *INTx – PINy.z* в единице) сохраняется текущее значение счетного регистра в переменной *pred_cnt*, по срезу вычисляется разность текущего и предыдущего *n_ti*, численно равная длительности импульса в микросекундах.

Таблица 5.4

PORTD2_per-25-26-99-100.sti	PORTA-8MHz.log	per.txt	
000000400:04	000000000:00	CycleCounter	d_per, us
000000500:00	000000454:2F	429	47
000000600:04	000000654:19	629	25
000000700:00	000000859:1A	838	26
000000808:04	000001654:63	1629	99
000000900:00	000002454:64	2429	100
000001600:04	000000000:00		
000001700:00			
000002400:04			
999999999:00			

Данный вариант алгоритма рассчитан на вход прерывания *INTx* имеющий разные настройки для выявления фронта и среза импульса. В более современных моделях МК есть настройка для выявления любого изменения входного сигнала, тогда не требуются действия по перенастройке в *ISR*.

2. Измерение частоты f периодического сигнала

Алгоритм состоит из двух подзадач:

- формирование интервала измерения, здесь – с использованием прерывания по переполнению;

- счет импульсов:

- при частоте $f < 1$ кГц можно использовать вход прерывания *INTx* настроенный на фронт (или срез);

- при частоте $f > 1$ кГц удобнее использовать счетный вход *Tx* и режим счетчика свободного таймера/счетчика.

Задание 2.1. Измерить частоту импульсов в диапазоне от 5 до 400 Гц с точностью до 1 Гц.

Для счета импульсов используем вход *INT0* с настройкой на фронт, переменная счетчика в *ISR INT0* должна иметь размерность два байта во избежание переполнения.

Точность измерений задает длительность интервала измерения – 1 секунда, для тактирования периода 1 с используем 8-разрядный таймер-0. При выборе максимального значения делителя *kdiv* = 1024 переполнение таймера будет происходить с частотой *ftov0* = *fclk* / *kdiv* / *TOP_8* = 8 МГц / 1024 / 256 = 30.52 Гц. При 30 переполнениях инервал будет 0.983 с (-1.7 %), при 31 – 1.016 с (+1.6 %). Дискретность измерения по заданию 1/400, то есть 0.25 %. Для программного отсчета секунды в *ISR* TOV0 следует организовать циклический счет 30 переполнений без коррекции значения счетного регистра, на 31-ом - записать значение для интервала 17 мс (0.017 * 8000000 / 1024 = 132.81, то есть -132). Симулятор AVR-Studio показывает значение интервала 999936 мкс, погрешность 0.0064 %!

Проект с примером находится в папке ismer_T0 \ ism_freq_INT0 в файле <ism_freq_T0_INT0.aps>. Там же находится файл <ism_freq_T0_INT0_8515.dsn> для тестирования примера в Proteus VSM, наличие модели генератора импульсов D_Pulse

гораздо удобнее формирования длинных файлов стимуляции!

При частоте импульсов 5 Гц значение частоты, выводимое в основном цикле на порты A (младший байт) и B (старший байт) 0b 0101 = 0x05 = 5, при частоте 100 Гц – 0b 0110 0100 = 0x64 = 100, при частоте 399 Гц – 0b 1 1000 1111 = 0x18f = 399, при частоте 400 Гц – 0b 1 1001 0000 = 0x190 = 400. Погрешность отсутствует.

Задание 2.2. Измерить частоту импульсов в диапазоне от 50 до 400 кГц с точностью до 0.1 кГц.

Здесь требуется два таймера/счетчика – один для счета измеряемых импульсов, другой – для формирования интервала измерения.

Диапазон измеряемых значений от 500 до 4000, поэтому для счета импульсов используем 16-разрядный таймер/счетчик TC1 (вход *T1\PB1*), для настройки в TCCR1B следует установить биты CS12, CS11, CS10 (счет по фронту).

Период измерений 10 мс (*fm* = 100 Гц) формирует 8-разрядный TC0 с перезагрузкой в прерывании по переполнению счетного регистра значением *Nm = fclk / kdiv / fm* = 8 МГц / 1024 / 100 Гц = 78.125, то есть -78 – в регистр TCNT0 (9984 мкс, отклонение 0.16 %). Для настройки в TCCR0 установить биты CS02, CS00, разрешить прерывание по переполнению.

Тестирование в Proteus VSM: при частоте генератора тактов D_Clock f = 50 кГц (ширина импульса Width=1e-05 составляет половину периода) на выходах портов C(ст) и A(мл) 0b1 1111 0011/0010 = 0x1f3 / 0x1f4 = 499 / 500, при f = 400 кГц (Width=1.25e-06) 0b 1111 1001 1001/10 = 0xf99 / 0xf9a = 3993 / 3994 (-7/-6). Относительная погрешность измерения по частоте 7 *100 / 4000 = 0.175 % близка к отклонению интервала измерения.

Ограничения базового таймера/счетчика

Точность формирования и измерения временных интервалов с использованием только таймеров с прерыванием по переполнению и входов прерываний быстро снижается при работе с сигналами, частота которых приближается к частоте тактирования (процессора) менее чем на три порядка.

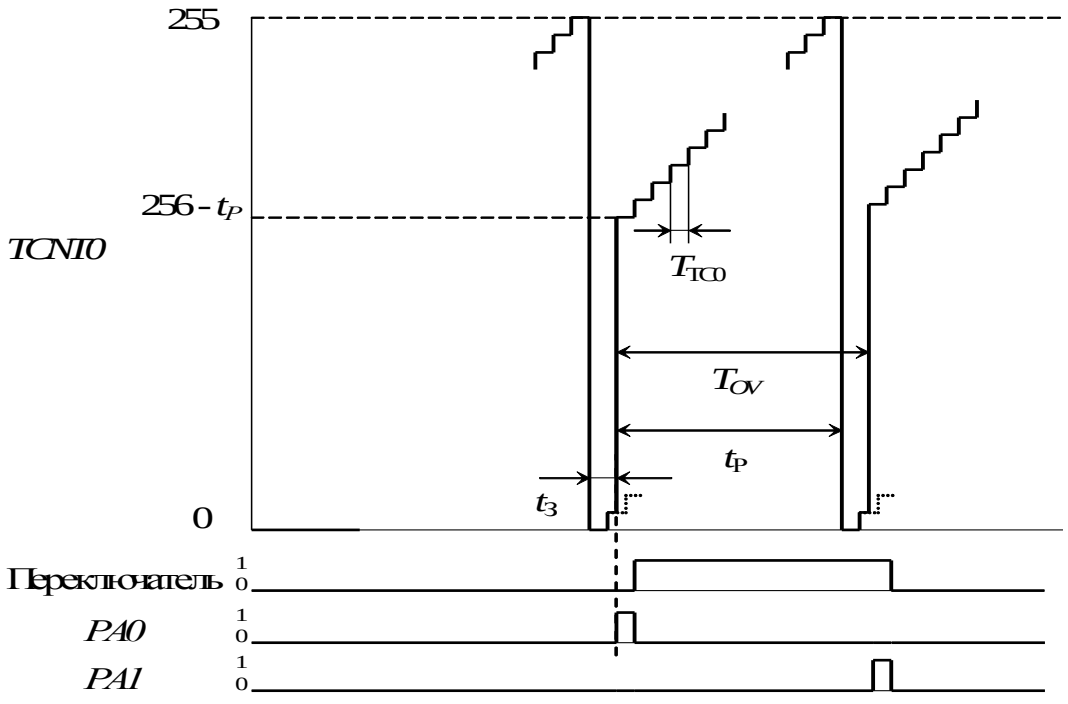


Рис. 5.10

Во-первых, как видно из рис. 5.10 (иллюстрация к примеру генерации с ЧМ), реальный период переполнения T_{OV} не совпадает с расчетным интервалом переполнения t_P на величину $t_З$, определяемую временем предоставления прерывания (5-7 м.ц.) и выполнением команд ППОП, предшествующих перезагрузке ТС0 (сохранение контекста и ветвления). Это можно попытаться учесть соответствующей коррекцией числа, загружаемого в счетчик, но тут есть два препятствия, осложняющих точный расчет этой коррекции. Первое – это случайный разброс времени предоставления прерывания равный +/–1 машинному циклу (T_{CLK}), что вызвано разбросом длительности выполнения прерываемой команды (1…3 машинных цикла для *AVR* МК). Второе возникает при использовании одного прерывания для формирования различных по длительности временных интервалов – невозможно точно уравнять расходы времени на выбор используемой на очередном шаге уставке длительности.

Во-вторых, наличие других прерываний может увеличить формируемый временной интервал на длительность выполнения других ISR.

И, наконец, при большой частоте повторения прерываний становится значительной относительная доля времени, связанная с расходами на прерывания, т. е. резко снижается производительность процессора в фоновом режиме.

При формировании и измерении сравнительно высокочастотных сигналов гораздо эффективнее использовать таймеры/счетчики с расширенным набором блоков и функций.

5.4. Таймер/счетчик с дополнительными узлами захвата и сравнения

Дополнительные узлы захвата и сравнения обеспечивают таймеру/счетчику преимущества в задачах формирования и измерения импульсных сигналов

Таймер/счетчик TC1 в МК *at90s8515* состоит из 16-разрядного базового таймера и дополнительных узлов, выполняющих функции сравнения/ШИМ и захвата. Его упрощенная структурная схема представлена на рис. 5.6.

Прежде всего, как и *TC0*, он может использоваться для формирования временных интервалов или подсчета числа внешних событий по входу *T1*, но в 16-разрядном варианте, т. е. переполнение происходит при переходе от числа *TOP* = $2^{16} – 1$ = `0xffff` = 65535 к числу 0. [`Normal top = 0xffff`].

Набор источников тактирования не отличается от 8- разрядного варианта – *fclk* / *K*, *K* = 1, 8, 64, 256, 1024, фронт или срез внешних ипульсов на входе *T1*. Выбор канала тактирования осуществляется набором бит *CS12, CS11, CS10* аналогично Табл. 5.1.

Узел сравнения/ШИМ сравнивает текущее значение счетного регистра *TCNT1* с числом в регистре сравнения *[Output Compare Register – OCR1A/B]*, при совпадении формирует на специальном выходе (*OC1A/B*) заданное изменение состояния (фронт или срез или переключение) и запрос прерывания *OCF1A/B*. Точность формирования фронта или среза в этом случае определяется только стабильностью и фазой тактирующих импульсов *ftc1* и не содержит программных задержек.

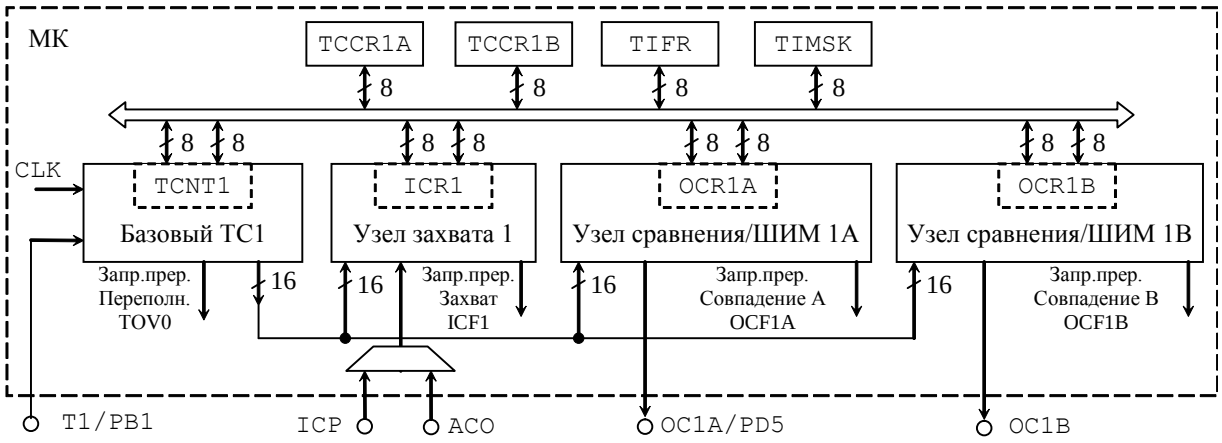


Рис. 5.11

Узел захвата при заданном событии – изменении состояния специального входа захвата (*Input Capture Pin – ICP)* сохраняет текущее значение счетного регистра *TCNT1* в регистре захвата (*Input Capture Register – ICR1*) и формирует запрос на прерывание *ICF1*. Точность захвата определяется тактирующими импульсами *ftc1*, для снижения влияния импульсных помех есть программно выбираемый механизм подавления дребезга (*ICNC1*), проверяющий условие перехода на 4 последовательных тактах (машинных циклах). Биты настройки позволяют выбрать тип события на входе ICP (бит ICES для выбора фронта или среза), либо выбрать в качестве события изменение сигнала на выходе аналогового компаратора ACO (описание компаратора см. в разделе 6).

Прерывания включают кроме уже знакомого прерывания по переполнению (*TOV1*) еще три новых: по захвату (*ICF*), и по сравнению (*OCF1A/B*). Прерывание по захвату обычно используется для сохранения значения регистра *ICR* в программной переменной, иначе следующий захват затрет предыдущее значение. Прерывание по сравнению может понадобиться для синхронизации очередной записи регистр сравнения или для счета формируемых периодов. Как и в прерывании по переполнению, сброс флага запроса прерывания выполняется аппаратно при вызове *ISR* или программно записью единицы в регистр флага.

Работа с 16- разрядными регистрами

Каждый из 16-разрядных регистров представлен в адресном пространстве РСФ парой 8-разрядных регистров, имеющих подобное имя, но с добавкой *H* [*high*] или *L* [*low*] соответственно.

При записи в эти 16-разрядные регистры необходимо первым записывать старший байт (например, *OCR1AH*), затем младший (*OCR1AL*). При чтении последовательность действий обратная – сначала чтение младшего, затем старшего. Соблюдение данного порядка и наличие скрытого 8-разрядного регистра гарантирует цельность данных при чтении и одномоментность изменения при записи. Разработчики МК называют операции чтения или записи двухбайтных регистров атомными (то есть не делимыми по времени) и рекомендуют на время их выполнения запрещать прерывания.

При программировании на языке *C/C++* допустимо оперировать с 16-битными регистрами (обычно в формате *unsigned int*), правильная последовательность побайтной отработки операций чтения и записи гарантируется транслятором.

Таблица 5.5. Регистры и биты TC1.

TCCR1A – Timer/Counter1 Control Register A								
Бит	7	6	5	4	3	2	1	0
	COMA1	**COM1A0**	**COM1B1**	**COM1B0**	**FOC1A**	**FOC1B**	**WGM11**	**WGM10**
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W
Init val	0	0	0	0	0	0	0	0
TCCR1B – Timer/Counter1 Control Register B								
Бит	7	6	5	4	3	2	1	0
	ICNC1	**ICES1**	**-**	**WGM13**	**WGM12**	**CS12**	**CS11**	**CS10**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Init val	0	0	0	0	0	0	0	0
TIMSK – Timer/Counter Interrupt Mask Register								
Бит	7	6	5	4	3	2	1	0
	OCIE2	**TOIE2**	**TICIE1**	**OCIE1A**	**OCIE1B**	**TOIE1**	**OCIE0**	**TOIE0**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Init val	0	0	0	0	0	0	0	0
TIFR – Timer/Counter Interrupt Flag Register A								
Бит	7	6	5	4	3	2	1	0
	OCF2	**TOV2**	**ICF1**	**OCF1A**	**OCF1B**	**TOV1**	**OCF0**	**TOV0**
Read/Write	R/W	R/W	R/W	R/W	R|W	R|W	R/W	R/W
Init val	0	0	0	0	0	0	0	0

В таблице 5.6 приведены названия, биты для выбора и некоторые особенности режимов работы многорежимного 16-битного таймера/счетчика. В первой графе представлен номер режима, в следующих четырех – значения битов выборки режима (размещаются в регистрах TCCR1A, TCCR1B), далее – имя режима, в графе «TOP» – константа или регистр для размещения значения верхнего предела счета, в графе «Update of OCR1x» - значение счетного регистра, при котором обновляются значения регистров сравнения (при буферизации, «Immediate» - без буферизации), в графе «TOV1 Flag Set on» - значение счетного регистра, при котором устанавливается флаг прерывания по переполнению.

Таблица 5.6. Настройки режимов TC_{16}

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Все режимы можно разделить на многочисленную группу ШИМ-режимов (генерация широтно-модулированного сигнала на выходах сравнения) и группу из

режимов 0, 4 и 12.

Режим 0 называется *Normal*, счетчик *TCNT1* ведет счет на увеличение от *Bottom* = 0 до *MAX* = $2^{16} - 1 = 65535$, далее переполнение (флаг *TOV1*) и т.д. Период переполнения без перезаписи *TCNT1* равен *Tov1* = 65536 * *K* / *fclk*. Одновременно можно использовать функции захвата и сравнения.

Функция захвата по фронту или срезу на входе *ICP1* сохраняет значения счетчика *TCNT1* в регистре *ICR1*, что удобно при измерении длительности периода или длительности импульса. Для настройки этой функции используйте биты: *ICES1* – выбор фронта (1) или среза (0), *ICNC1* – подавление шума (ожидание четырех тактов повторения), *TICIE1* – разрешение прерываний по захвату (регистр *TIMSK*), *ICF1* – флаг прерывания (регистр *TIFR*).

Так, при настройке входа захвата на фронт разность последнего значения регистра *ICR1* и предыдушего буде пропорциональна периоду сигнала, поданного на вход *ICP1*: *T* = (*ICR1 – предыдущее_ICR1*) / f_{TC1}. Запоминание предыдущего и вычисление разности следует выполнять в подпрограмме прерывания по захвату, пересчет в требуемый масштаб лучше выполнять в основном цикле (см. пример…). Для измерения длительности импульса алгоритм требует дополнительной перенастройки бита *ICES1* в прерывании.

Функция сравнения в момент совпадения значений TCNT1 и OCR1A/B формирует запрос прерывания (OCF1A/B) и событие на выходе сравнения (OC1A/B). Для выбора события на выходе узла сравнения используют пары бит COM1x1 и COM1x0 (x = A / B), см. Табл. 5.7. Для разрешения прерываний по совпадению служат биты OCIE1A/B регистра TIMSK.

Таблица 5.7. Биты управления выходами сравнения в режимах 0, 4, 12 (не ШИМ)

COM1x1	COM1x0	Описание
0	0	Выход OC1x отключен
0	1	Переключать [Toggle] выход OC1x
1	0	Обнулить выход OC1x (однократно)
1	1	Установить выход OC1x (однократно)

Для генерации непрерывных импульсов подходит только выбор переключения (код 01), в этом случае длительности импульса и паузы равны tи = tп = Tov1, период сигнала в 2 раза больше периода переполнения Tг = 2 Tov1.

Запись 1 в бит FOC1A (FOC1B) регистра TCCR1A формирует событие на выходе OC1A (OC1B) без прерывания и сброса TCNT1 (в режиме CTC).

Режимы 4 и 12 называются *CTC [Clear Timer on Compare Match]*, то есть автосброс по совпадению содержимого TCNTx и регистра OCRnA (режим 4) или ICRn (режим 12). Содержимое TCNTx растет вверх до совпадения (рис. 5.12), затем автоматически обнуляется, в этот момент на выходе OCnA формируется фронт или срез (однократные!) или переключение (многократное). Разрядность определяется разрядностью счетчика (8 или 16 бит). Устанавливается флаг прерывания OCnA или ICFn (прерывание в TOP). Для генерации импульсов подходит только выбор переключения [toggle: (1<<COM1y0)], частота импульсов f_{OCnA} = fclk / (2 * K * (OCRnA + 1)), импульс равен паузе. Использование дополнительных каналов сравнения (B и пр.) позволяет получить импульсы, сдвинутые по отношению к каналу A, при условии OCR1B < TOP.

Запись в регистр значения TOP не буферирована, поэтому рекомендуется синхронизировать запись с моментом автосброса, например, по прерыванию. Запись в момент, когда значение TCNTx близко к порогу срабатывания может приводить к появлению ложных импульсов или пропуску импульсов.

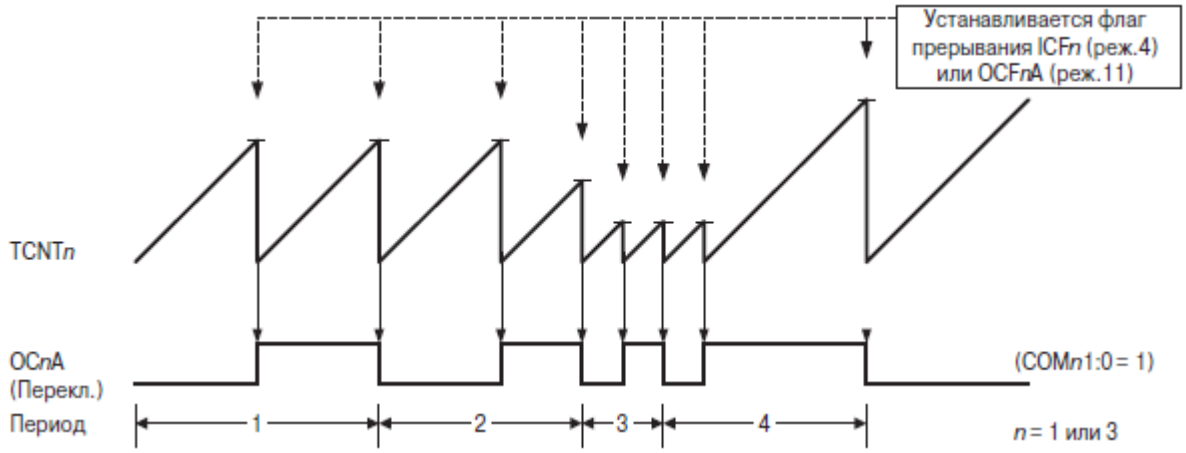


Рис. 5.12. Режим СТС

В режиме 4 можно использовать функцию захвата, аналогично режиму 0.

Все **режимы ШИМ *[PWM]*** используют буферизацию при записи в регистры сравнения (используя регистр ICRn или OCRnA), поэтому записанное значение будет использовано на следующем периоде ШИМ сигнала. Это гарантирует пропуск импульсов и отсутствие ложных импульсов.

Режимы 5, 6, 7, 14, 15 называются ***Fast PWM*** (быстрая ШИМ), имеют разрядность N = 8, 9, 10 или 16 бит, счетчик традиционно работает на возрастание до TOP, затем автосброс в BOTTOM = 0 (рис. 5.13). При совпадении значений регистров TCNTx и OCRxy на выходе OCxy формируется срез (или фронт) и устанавливается флаг прерывания по совпадению OCxy, при автосбросе – фронт (или срез) и устанавливается флаг прерывания по переполнению TOVx.

Частота импульсов f_{OCxy} = fclk / (K * (TOP + 1)), ширина импульса (или паузы) t = OCRxy * K / fclk.

В первых трех вариантах (N = 8, 9, 10) значение TOP, то есть период, фиксированные (TOP = 2^N + 1 = 255, 511, 1023 такта), в сравнении участвуют только указанное число бит регистров сравнения. 16-битная быстрая ШИМ использует в качестве значения TOP содержимое регистра OCRnA или ICRn, то есть обеспечивает подстраиваемый по числу тактов период, реально можно использовать значения от 2 (TOPmin = 0x0003) до 16 (TOPmax = 0xffff) разрядов. При обновлении значения TOP в регистре OCRnA или ICRn новое значение должно быть не меньше значений в регистрах сравнения, иначе импульса не будет. В момент автосброса устанавливается флаг прерывания по совпадению OC1A или по захвату ICF. Ответственность за соотношением значения TOP и регистра сравнения лежит на программисте. Если в качестве TOP используется регистр ICRn, запись в него не буферирована, запись в OCRnA буферирована.

Самый короткий период генерируемого сигнала получаем при минимальной разрядности (N = 8), значение TOP = 255, то есть период 256 тактов ftc. Этот вариант

удобен для построения аналогового вывода из МК с использованием внешней сглаживающей RC-цепи. В папке T1_FastPWM \ FastPWM8-AO находится проект программы FastPWM8-AO_8515.aps, и модель Proteus VSM FastPWM8-AO_8515.dsn (МК с RC-цепью на выходе OC1A и временная диагармма сигналов).

Таблица 5.8. Биты управления выходами сравнения в режиме быстрой ШИМ

COM1x1	COM1x0	Состояние выхода
0	0	Выход OC1x отключен
0	1	Если WGM13 = 1, то переключать [Toggle] OC1A, иначе выход отключен
1	0	Неинвертированная ШИМ: При совпадении выход OC1x переходит в 0, при TCNT1 = TOP OC1x переходит в 1
1	1	Инвертированная ШИМ: При совпадении выход OC1x переходит в 1, при TCNT1 = TOP OC1x переходит в 0

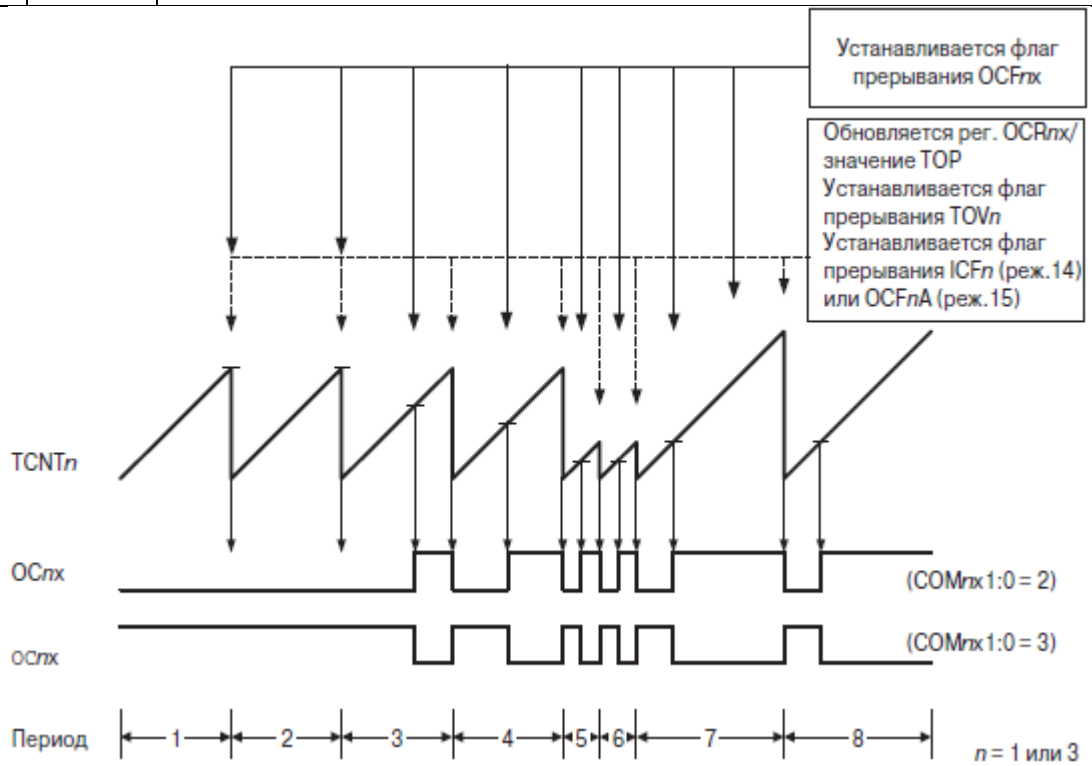


Рис. 5.13. Fast PWM

В соседней папке FastPWM-ICR-period находся проект программы FastPWM-ICR-period_m16.aps и модель: «Генерация двухтактных импульсов с ЧМ без паузы, вариация частоты за счет вариации периода T = 1/f в регистре ICR = fclk / f, канал A - инв-ные импульсы длительностью ICR/2, канал B - прямые импульсы длительностью ICR/2».

Режимы 2, 3, 4, 10, 11 называются ***Phase Correct PWM***, то есть фазово корректная или симметричная ШИМ, отличается от предыдущего тем, что цикл счетчика в два раза длиннее: сначала на возрастание до TOP, затем на убывание до BOTTOM = 0 (рис. 5.14). Фронты и срезы формируются на выходе OCxy только при совпадении значений регистров OCRxy и TCNTx (одновременно устанавливается флаг OCxy), поэтому частота импульсов в два раза ниже f_{OCxy} = fclk / (2 * K * TOP).

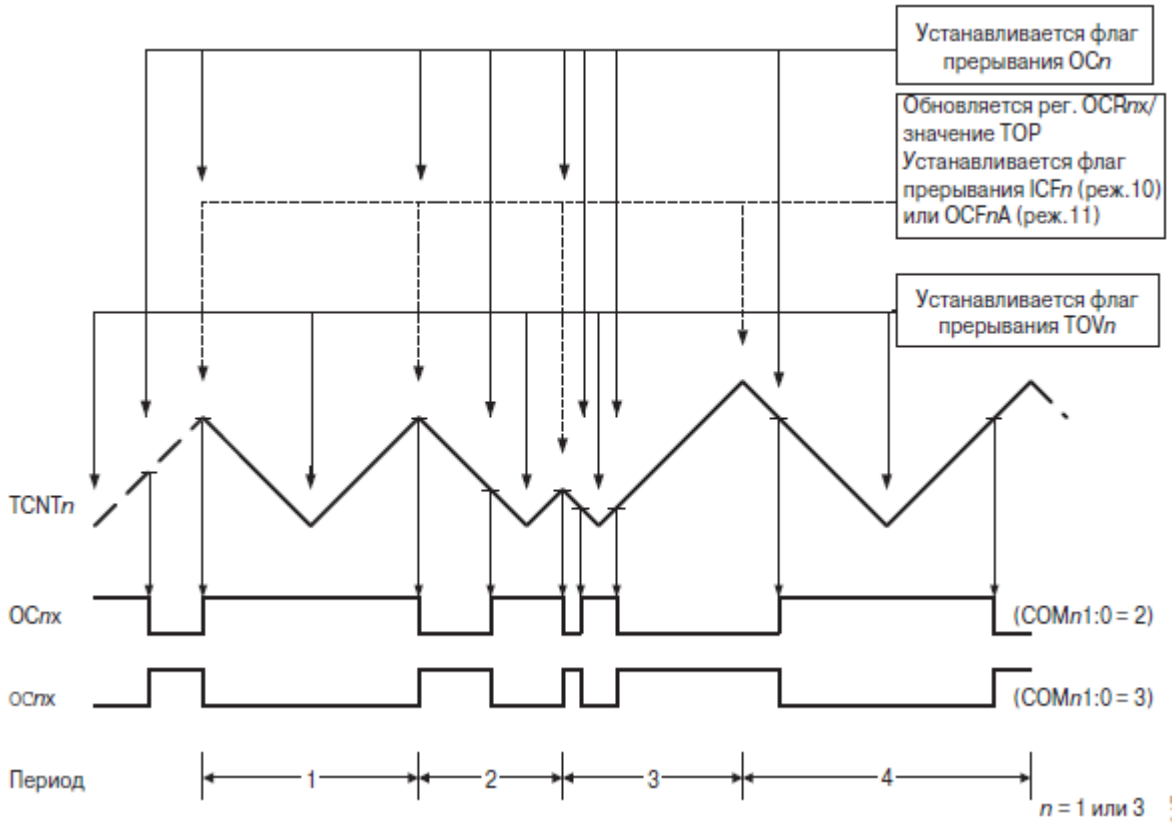


Рис. 5.14. Phase Correct PWM

Таблица 5.10. Биты управления выходами сравнения в режимах «симметричной» ШИМ

COM1x1	COM1x0	Состояние выхода
0	0	Выход OC1x отключен
0	1	Если WGM13 = 1, то переключать [Toggle] OC1A, иначе выход отключен
1	0	Неинвертированная ШИМ: При совпадении выход OC1x переходит в 0 при прямом счете или переходит в 1 при обратном
1	1	Инвертированная ШИМ: При совпадении выход OC1x переходит в 1 при прямом счете или переходит в 0 при обратном

Значение TOP, как и в Fast PWM либо фиксированное (255, 511, 1023 – по выбору разрядности), либо определяется значением, записанным в регистр OCRnA или ICRn (плюс флаг OCnA или ICFn). Флаг переполнения TOVn устанавливается в BOTTOM.

В папке T1_PhaseCorrectPWM находся проект программы PhaseCorPWM_m16.aps и модель: «Генерация двухтактных импульсов с симметричной ШИМ 8-бит, , канал A - инверсные импульсы с законом вариации OCR1A = TOP/2 + n, канал B - прямые импульсы OCR1B = TOP/2 - n, n = 0… TOP/2».

Режимы 8, 9 называются ***Phase and Frequency Correct PWM***, то есть фазово и частотно корректная ШИМ, почти тождественны режимам 10, 11, но отличаются моментом обновления регистров сравнения (рис. 5.15) – при нулевом значении счетного регистра.

В папке T1_Phase&FreqCorrectPWM находся проект программы FreqCorrPWM_fvar_m16.aps и модель: «Генерация двухтактных импульсов с ЧМ, режим 8. Период задается числом в регистре ICR1 = fclk / fset, канал A - инверсные импульсы с законом вариации OCR1A = fclk / fset / 2 + n, канал B - прямые импульсы OCR1B = fclk / fset / 2 - n, здесь n = 0 – при отсутсвии ШИМ регулирования, иначе n = 0… fclk / fset / 2».

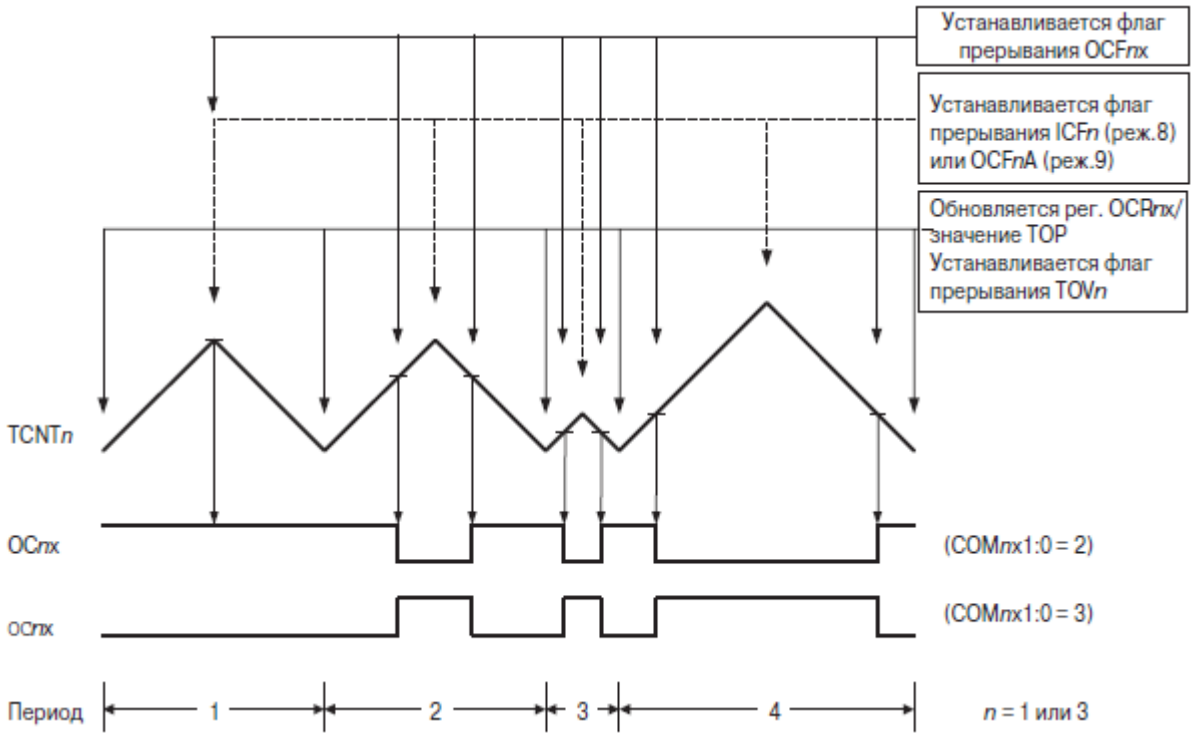


Рис. 5.15. Phase and Frequency Correct PWM

Пример генерации импульсов с варьируемым периодом (ЧМ).

Задана вариация частоты от 5 до 10 кГц – 100 квантов, 1 канал, ти = тп. При равномерном изменении периода от 100 до 200 мкс шаг составит 1 мкс, при этом хватает 8 разрядов. В режиме CTC top=OCR0 с настройкой выхода OC0 на переключения [Toggle] получим меандр с периодом T = 2*OCR0, тогда ftc = 2 МГц, для модели МК ATmega16 удобно выбрать fclk = 16 МГц и предделитель 8. Для генерации частоты от 5 до 10 кГц в OCR0 надо записать число от 200 до 100.

Пример генерации ШИМ сигнала.

Задано f_{out} = 16 кГц, $t_и * f_{out}$ = 0.25…0.75, 100 квантов, 1 канал.

Используем блок и выход сравнения OC1A таймера/счетчика-1. Выбор режима ШИМ в AT90S8515: *Phase correct PWM top = 0xff / 0x1ff / 0x3ff*. Так как интервал вариации 0.75 – 0.25 = 0.5 составляет 100 квантов, то период состоит из 200 квантов. Режим с минимальной разрядностью 256 квантов (*0xff*) определяет частоту тактирования таймера/счетчика-1 и процессора *fclk* = 16 кГц *510 = 8.16 МГц, округляем до 8 МГц.

Для получения значений $t_и * f_{out}$ = 0.25…0.75 в регистр OCR1A следует записывать значения от 0.25*256=64 до 0.75*256=192, все они помещаются в младший байт 16-разрядного регистра, то есть в OCR1AL.

Альтернативный вариант решения – использование режимов *Fast PWM top = OCR1A* (в более современных моделях МК).

Режим CTC top = OCR1A позволяет формировать на выходе OC1A (настройка Toggle) меандр с регулируемым периодом T = 2* OCR1A. При записи в регистр OCR1B числа меньшего, чем в регистре OCR1A на выходе OC1B (настройка Toggle) получаем меандр с тем же периодом T = 2* OCR1A, но с опережением по фазе. Если выходы OC1A и OC1B подключить на входы внешней логики 2И, то на выходе получим сигнал OC1A & OC1B, который имеет период T = 2 * OCR1A и ширину импульса ти = OCR1B,

максимальная ширина импульса не превышает половину периода.

Пример измерения периода

Заданы параметры входного сигнала период T = 50…100 мкс, ti = tп, измерить период с шагом $t_ш$ = 1/8 мкс.

Используем вход захвата ICP таймера/счетчика-1 настроенный на фронт, частота тактирования ftc1 = fclk = 1/ $t_ш$ = 8 МГц. За максимальный по длительности период в счетчик попадет nper_max = Tmax / $t_ш$ = 100*8 = 800 импульсов, то есть 16-разрядного регистра TCNT1 достаточно. Используем ISR прерывания по захвату для сохранения текущего значения регистра захвата и вычисления разности текущего и предыдущего в формате unsigned int.

{ ntek = ICP; nper = ntek – npred; npred = ntek; }

Переполнение счетчика не приводит к ошибке при вычислении разности, если формат вычислений совпадает с форматом регистров счетчика.

Данный пример находится в папке T1_Normal в проекте t1_mode0.aps, схема с генератором для тестирования в t1_mode0_m16.dsn, погрешность измерения отсутствует.

Контрольные вопросы по р. 2-5

Теоретические

1. Подсистема ввода/вывода в процессорных устройствах.
2. Параллельный интерфейс шинного типа – системная магистраль.
3. Подключение внешней памяти данных в 8-разрядных МК.
4. Ввод/вывод через параллельный порт.
5. Ввод/вывод с использованием системы прерываний.
6. Метод прямого доступа к памяти.
7. Программный опрос (поллинг) и типовые задачи управления.
8. Периферийные устройства 8-разрядных МК.
9. Параллельный порт в режиме ввода.
10. Параллельный порт в режиме вывода.
11. Система прерываний 8-разрядных МК.
12. Входы прерываний и их применение
13. Программное формирование постоянных и переменных временных интервалов.
14. Генерация импульсов управления с использованием таймеров/счетчиков.
15. Измерение частоты внешних импульсов
16. Измерение длительности (периода) внешних импульсов.
17. Таймеры/счетчики 8-разрядных МК в режиме счета до переполнения.
18. Функции сравнения и захвата таймеров/счетчиков 8-разрядных МК.
19. Таймеры/счетчики 8-разрядных МК в режиме формирования ШИМ сигнала.
20. Схема подключения и алгоритм обслуживания многосегментных светодиодных индикаторов.
21. Схема подключения и алгоритм обслуживания клавиатуры

Практические

1. Типовая схема подключения МК с внутренним тактированием в корпусе DIP-8.
2. Типовая схема подключения МК с кварцевым резонатором на 8 МГц в корпусе DIP-20.

3. Схема 8-разрядного МК с внешней памятью данных размером 16 кБ.
4. Схема 8-разрядного МК с внешней памятью данных размером 256 кБ.
5. Схема и алгоритм светофора (3 светодиода на одно направление и 3 на другое).
6. Схема и алгоритм светофора по требованию (кнопка и 3 светодиода).
7. Контроль частоты вращения вентилятора с индикацией снижения (<600 об/мин) на светодиоде – схема и алгоритм.
8. Алгоритм измерения частоты импульсов в диапазоне 2…250 Гц (шаг 1 Гц) с выводом на цифровой индикатор.
9. Алгоритм измерения частоты импульсов в диапазоне 50…500 кГц (шаг 1 кГц) с выводом на цифровой индикатор.
10. Матричная схема и алгоритм динамической индикации (50 Гц) четырех 7-сегментных светодиодных индикаторов (отображение цифр 0…9).
11. Алгоритм генерации импульсов $f_И$ = 700…1000 Гц с числом шагов n = 200, $t_И = t_П = T_И/2$ с использованием функции и выхода сравнения с автосбросом таймера/счетчика, выбрать частоту тактирования f_{CLK}.
12. Алгоритм измерения периода внешнего импульса с $T_И$ = 25…100 мкс $t_И = T_И/2$ с шагом 0.2 мкс, использовать прерывание по переполнению таймера/счетчика и вход прерывания, выбрать частоту тактирования f_{CLK}.
13. Алгоритм программного измерения периода импульса с $T_И$ = 1000…1250 мкс с шагом 5 мкс, выбрать частоту тактирования f_{CLK}.
14. Алгоритм формирования отсчета реального времени ММ:СС с выводом на цифровой индикатор, использовать таймер/счетчик, выбрать частоту тактирования f_{CLK}.
15. Схема и алгоритм измерения фазы (0.000, 0.001, 0.002, … 0.250 периода) между внешними импульсами Имп1 и Имп2 ($t_И = t_П$) на частоте 0.5…1 кГц, выбрать частоту тактирования f_{CLK}.
16. Алгоритм измерения периода внешнего сигнала Т = 100…140 мкс с шагом 0.2 мкс, использовать функцию захвата таймера/счетчика, выбрать частоту тактирования f_{CLK}.
17. Алгоритм измерения длительности внешнего импульса с $f_И$ = 1 кГц $t_И$ = 10…250 мкс с шагом $\Delta t_И$ = 1 мкс с использованием функции и входа захвата таймера/счетчика, выбрать частоту тактирования f_{CLK}.

Часть 6. Задачи и устройства аналогового ввода/вывода (Л12, Л13)

6.1. Задачи аналогового ввода и вывода

Процессорные устройства принципиально являются цифровыми устройствами. Программная реализация обработки или преобразования аналогового сигнала предполагает периодический ввод и вывод аналогового сигнала.

Аналоговый ввод в МК требует выполнения двух операций *дискретизации*:

1) дискретизации по уровню,

2) дискретизации по времени.

В итоге из непрерывного по уровню и времени аналогового сигнала u(t) получаем набор цифровых отсчетов n[i-1], n[i], n[i+1].... Значения отсчетов пропорциональны уровню сигнала в фиксированные моменты времени t[i-1], t[i], t[i+1]..., интервал между которыми обычно постоянный и называется шагом дискретизации tш = t[i+1] – t[i] = t[i] – t[i-1].

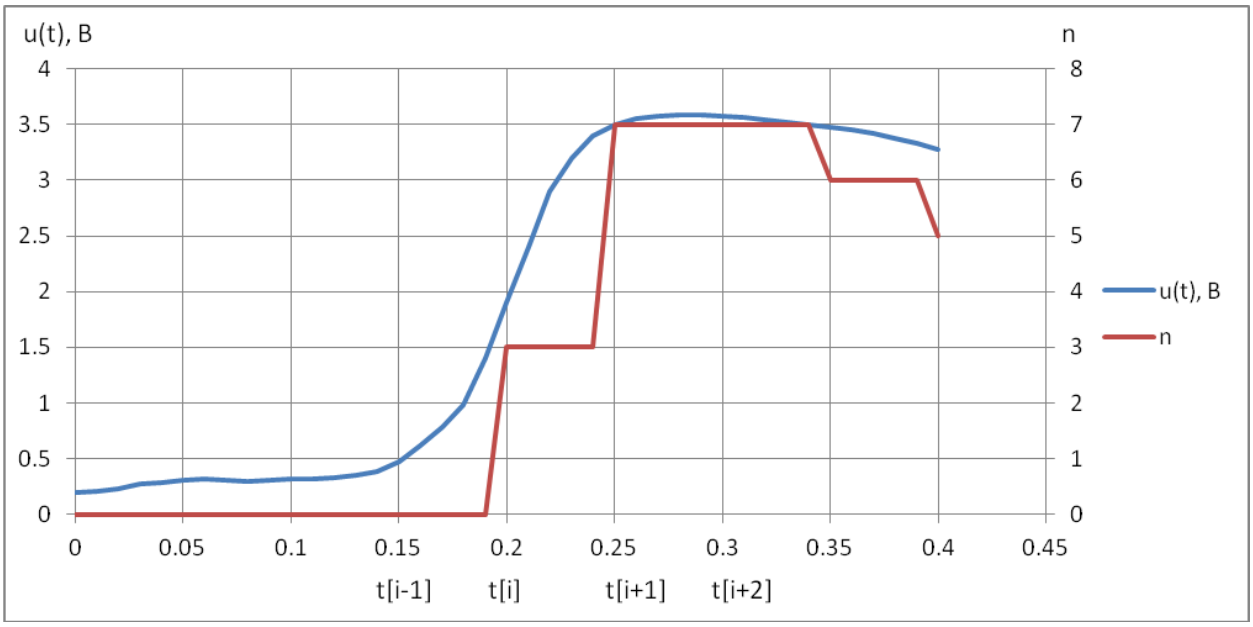


Рис. 6.1

Аналоговый вывод требует обратного преобразования цифровых отсчетов n[i-1], n[i], n[i+1]..., в близкий к непрерывному по уровню и времени сигналу u(t). Этот процесс называется *восстановлением*.

Дискретизация по уровню

Диапазон преобразования аналогового сигнала может быть однополярным [0, $U_{вх\_макс}$] или двуполярным [-$U_{вх\_мин}$, $U_{вх\_макс}$]. Пределы преобразования определяются выбором источника опорного напряжения, для однополярного сигнала $U_{вх\_макс}$ = $U_{оп}$ = *Uref* [от *reference* – опора].

Число уровней напряжения N в диапазоне от 0 до *Uref*, которые различаются на единицу в цифровом отсчете определяют дискретность преобразования. При двоичном кодировании это число кратно степеням двойки, например, 8-разрядное преобразование различает не более 2^8 = 256 уровней, 12-разрядное – 4096, то есть N двоичных разрядов задают 2^N уровней.

Закон аналого-цифрового преобразования n = (Uвх / Uref) * 2^N, здесь n – число на выходе АЦП, пропорциональное напряжению на входе Uвх.

Закон цифро-аналогового преобразования Uвых = n * Uref / 2^N, здесь Uвых – напряжение на выходе ЦАП, пропорциональное преобразуемому числу n.

Дискретизация по времени

Теорема Котельникова (в англоязычной литературе – Найквиста - Шеннона) гласит, что, если аналоговый сигнал *x(t)* имеет ограниченный спектр (от 0 до $f_в$), то он может быть восстановлен однозначно и без потерь по своим дискретным отсчетам, взятым с частотой **строго большей** удвоенной верхней частоты $f_в$: $f > 2 f_в$. Такая трактовка рассматривает идеальный случай, когда сигнал начался бесконечно давно и никогда не закончится, а также не имеет во временно́й характеристике точек разрыва. Именно это подразумевает понятие «спектр, ограниченный частотой $f_в$».

Разумеется, реальные сигналы (например, звук на цифровом носителе) не обладают такими свойствами, так как они конечны по времени и, обычно, имеют во временно́й характеристике разрывы. Соответственно, их спектр бесконечен. В таком случае полное восстановление сигнала невозможно и из теоремы Котельникова вытекают 2 следствия:

- Любой аналоговый сигнал может быть восстановлен с какой угодно точностью по своим дискретным отсчётам, взятым с частотой $f > 2f_в$, где $f_в$ — максимальная частота, которой ограничен спектр реального сигнала.
- Если максимальная частота в сигнале превышает половину частоты дискретизации, то способа восстановить сигнал из дискретного в аналоговый без искажений не существует.

Фильтрация аналоговых сигналов и наложение спектров.

Все АЦП работают путём выборки входных значений через фиксированные интервалы времени. Следовательно, выходные значения являются неполной картиной того, что подаётся на вход. Глядя на выходные значения, нет никакой возможности установить, как вёл себя входной сигнал *между* выборками. Если известно, что входной сигнал меняется достаточно медленно относительно частоты дискретизации, то можно предположить, что промежуточные значения между выборками находятся где-то между значениями этих выборок. Если же входной сигнал меняется быстро, то никаких предположений о промежуточных значениях входного сигнала сделать нельзя, следовательно, невозможно однозначно восстановить форму исходного сигнала.

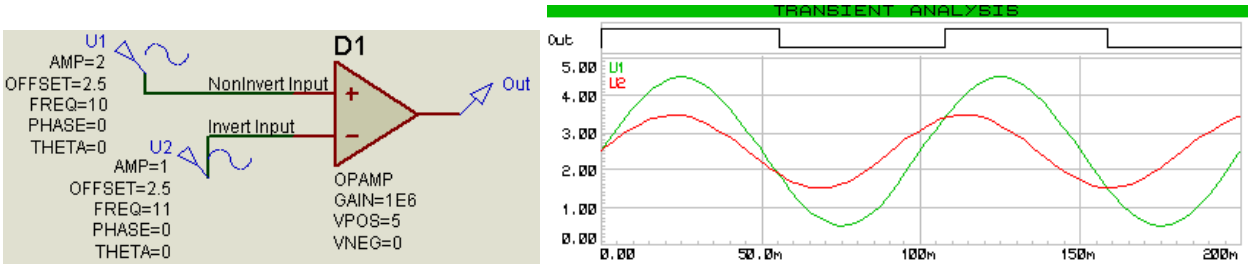
Если последовательность цифровых значений, выдаваемая АЦП, где-либо преобразуется обратно в аналоговую форму цифро-аналоговым преобразователем, желательно, чтобы полученный аналоговый сигнал был максимально точной копией исходного сигнала. Если входной сигнал меняется **быстрее**, чем делаются его отсчёты, то точное восстановление сигнала невозможно, и на выходе ЦАП будет присутствовать ложный сигнал. Ложные частотные компоненты сигнала (отсутствующие в спектре исходного сигнала) получили название *alias* (ложная частота, побочная низкочастотная составляющая). Частота ложных компонент зависит от разницы между частотой сигнала и частотой дискретизации. Например, синусоидальный сигнал с частотой 2 кГц, дискретизованный с частотой 1.5 кГц был бы воспроизведён как синусоида с частотой 500 Гц. Эта проблема получила название *наложение частот* (*aliasing*).

Для предотвращения наложения спектров сигнал, подаваемый на вход АЦП, должен быть пропущен через фильтр нижних частот для подавления спектральных компонент, частота которых превышает половину частоты дискретизации. Этот фильтр получил название *anti-aliasing* (антиалиасинговый) фильтр, его применение чрезвычайно важно при построении реальных АЦП.

6.2. Встроенный аналоговый компаратор

Аналоговый компаратор (АК) является простейшим устройством аналогового ввода. Он служит для сравнения двух аналоговых сигналов и выдает на выход высокий уровень, если сигнал на неинвертирущем входе больше, чем на инвертирующем и низкий уровень в обратном случае.

Аналоговый компаратор строится на базе операционного усилителя (рис. 6.2, а), оптимизированного по быстродействию для работы на участке насыщения (лучшие имеют десятки - единицы наносекунд). Выходной сигнал компаратора привязывается по уровню к логическим сигналам (рис. 6.2, б).



а б

Рис. 6.2

Большинство аналоговых компараторов, встроенных в МК имеют два внешних входа и "цифровое" питание, что ограничивает диапазон сравниваемых сигналов ($0 < u_{вх} <$ VCC). Выход компаратора доступен для чтения через регистр ввода/вывода и лишь в некоторых моделях МК выход одновременно выведен наружу для замыкания аналоговых и логических обратных связей.

В МК семейства AVR имеется один компаратор (рис. 6.3). Его входы обозначены AIN0 и AIN1, выход компаратора доступен для чтения через регистр ввода/вывода (ACSR.ACO). Изменение выхода компаратора (фронт, срез или любой переход) выявляется аппаратно и может порождать прерывание (ACI), а также срабатывание узла захвата таймера/счетчика 1 (ICP).

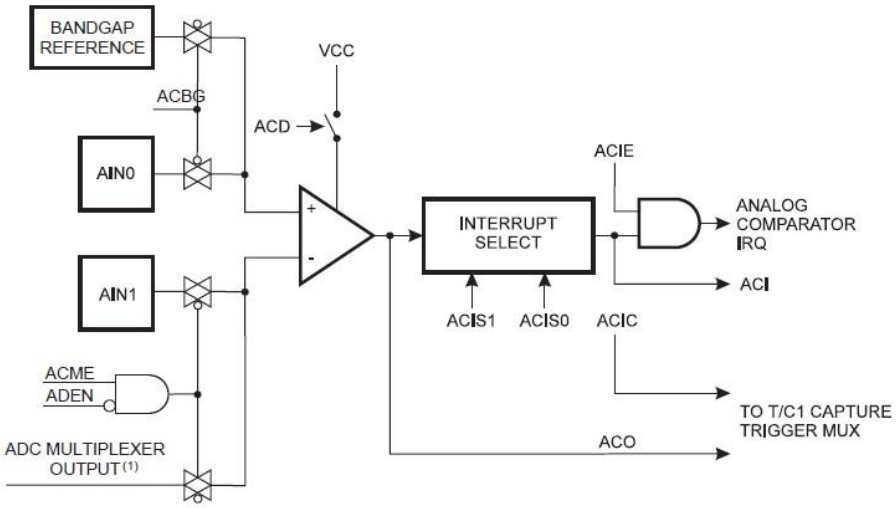


Рис. 6.3. Структура аналогового компаратора (ATmega16)

Таким образом, дискретизация по времени с использованием АК может быть выполнена в цикле поллинга (чтение выхода ACSR.ACO), либо автоматически по

(выбранному) изменению выхода (прерывание или захват "времени" в виде числа тактов таймера/счетчика).

Быстродействие встроенных компараторов оставляет желать лучшего (300-500 нс), но в последних высокопроизводительных МК встречаются модели с улучшенными значениями (30 нс в *ATxmega*). Для сравнительно медленных сигналов с зашумлением удобно использовать аппаратный подавитель шума. Его работа основана на отбрасывании слишком частых изменений выхода.

Во многих задачах необходимо сравнивать изменяющийся во времени сигнал с постоянным уровнем, поэтому один из входов аналогового компаратора может подключаться к внутреннему источнику так называемого "опорного" напряжения (Analog Reference).

Пример с АК: *система импульсно-фазового управления (СИФУ) однофазного выпрямителя на 50 Гц*. Выявляет переходы напряжения сети через ноль (синхронизация с сетью) и формирует импульсы управления с заданной задержкой по отношению к этим переходам. Параметры задания: длительность импульса 10 мкс, время задержки должно изменяться от 0 до 9.9 мс с шагом 0.1 мс и задаваться числом n от 0 до 99 через порт МК.

Для выявления моментов перехода напряжения сети через ноль используем понижающий измерительный трансформатор и встроенный аналоговый компаратор.

В модели Proteus VSM (<sifu-m16.dsn>, рис. 6.2) трансформатор и напряжение сети представлены источником напряжения V1 типа VSIN амплитудой VA = 2 В частотой FREQ = 50 Гц. К неинвертирующему входу AIN0 подключен делитель напряжения питания R1 и R2 по 1 кОм, обеспечивающий уровень VCC/2 = 2.5 В. На инвертирующий вход AIN1 подано напряжение источника VSIN со смещением 2.5 В (см. временную дигармму на рис. 6.2).

Аналоговый компаратор настроен на формирование прерываний по фронту и срезу (оба перехода через ноль) и в этом прерывании (`ISR(ANA_COMP_vect)`) перезапускает (обнуляет) таймер/счетчик TCNT1, а также переключает выход PC0 для индикации бита AC0.

Таймер-счетчик 1 настроен на режим таймера с частотой тактирования ftc1 = 1 МГц, переполнение может наступить через 2^{16} / ftc1 = 65536 мкс, но сброс каждые 10 мс гарантируется линейный рост примерно до 10000. Разрешено прерывание по совпадению значений TCNT1 и OCR1A, в этом прерывании на выходе PC1 программно формируется импульс длительностью 10 мкс.

Запись в OCR1A числа от 0 до 9900 с шагом 100 гарантирует прерывание, задержанное по отношению к переходу VSIN через ноль на требуемое по заданию время.

В основном цикле выполняется чтение числа n с порта A, преобразование этого числа в длительность задержки (в микросекундах) и запись результата в регистр сравнения OCR1A.

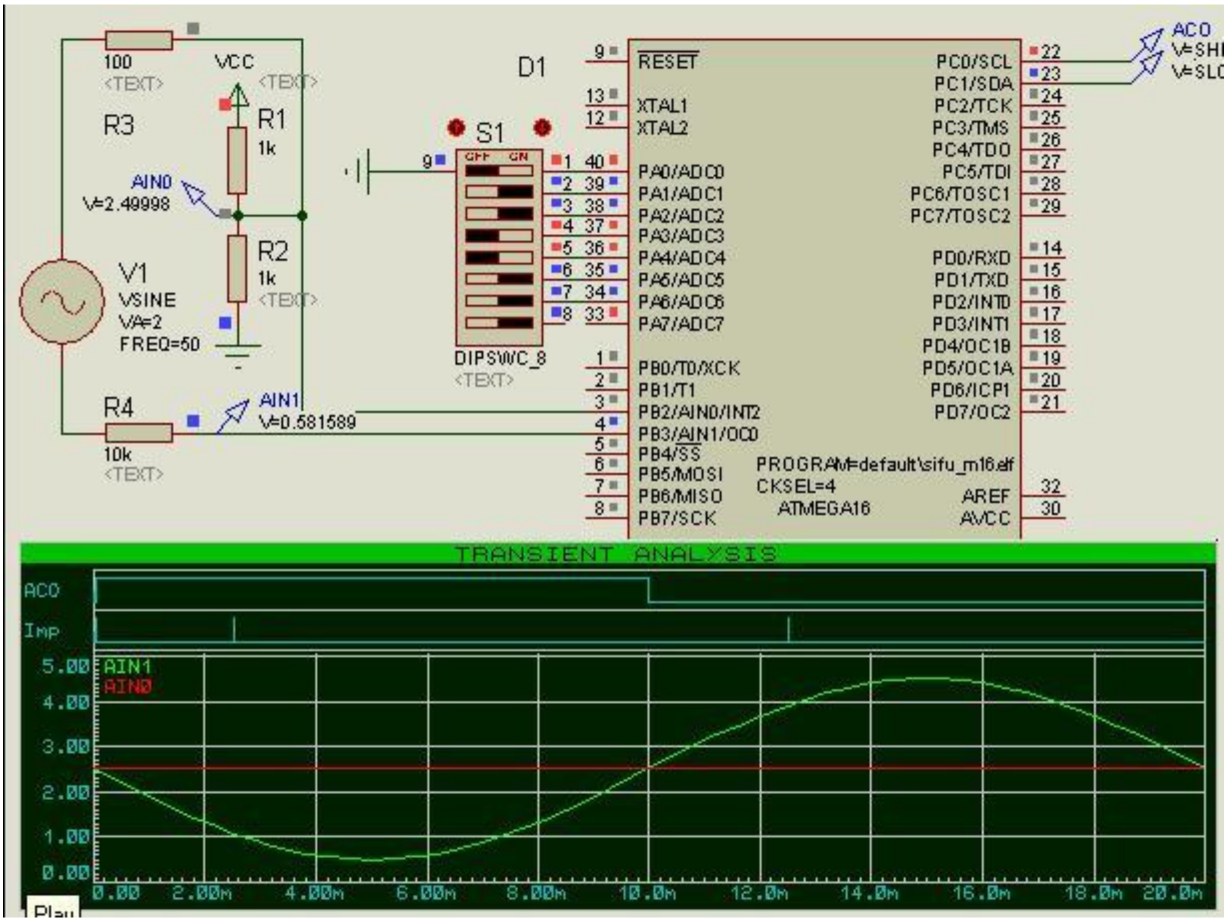


Рис. 6.4. Схема и временная диаграмма СИФУ в PROTEUS-VSM

Формируемые программой сигналы AC0 (PC0) и Imp (PC1) также приведены на временной диграмме.

Текст программы (файл <sifu_m16.c>).

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define FCLK  8000000L
#define _NOP() do { __asm__ __volatile__ ("nop"); } while (0)
#define DELAY_10us { _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP();
_NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP();
_NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP();
_NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP();
_NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP();
_NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP();
_NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP(); _NOP();
_NOP(); _NOP(); _NOP(); _NOP(); }

unsigned int ntz = 0;

ISR(TIMER1_COMPA_vect) {
  PORTC |= (1<<1);
  DELAY_10us;
  PORTC &= ~(1<<1);
} // *************************

ISR(ANA_COMP_vect)  {
```

```
  TCNT1 = 0;
  if(ACSR & (1<<ACO)) PORTC |= (1<<0); else  PORTC &= ~(1<<0);
} // *************************

unsigned int calc_ntz(unsigned char code) {
  ntz = code * 100;
  if(ntz == 0) ntz=1;
  if(ntz > 9900) ntz=9900;
  return ntz;
} // *************************

int main()  {
  ntz = calc_ntz(0);
  DDRC = (1<<1)|(1<<0);//0x03;
  PORTA = 0xff;
  TCCR1B = (1<<CS01);//ftc1 = fclk/8 = 1MHz, Ttc1=1us
  TIMSK = (1<<OCIE1A);//
  OCR1A = ntz;//
  ACSR = (1<<ACIE);//Разрешить АК, прерыв по фр и ср
  sei();

  while (1)   {
    ntz = calc_ntz(PINA & 0x7f)
    OCR1A = ntz;
  }
} // ******************************
```

Период тактирования таймера/счетчика задан шагом изменения задержки – он может быть равен или кратно меньше Ттс_макс = 0.1 мс = 100 мкс. При частоте тактирования МК fclk = 8 МГц требуемый коэффициент деления Красч = Ттс_макс * fclk = 800, такого значения в предделителе нет, при выборе К = 8 кратность составит 800 / 8 = 100, то есть ntz = n*100. Выбор данной частоты тактирования позволяет при необходимости задавать время задержки с дискретностью в 100 раз меньшей (1 мкс).

6.3. Встроенный многоканальный АЦП

Общие свойства и характеристики

Аналого-цифровое преобразование заключается в определении цифрового эквивалента n измеряемого сигнала *u(t)* по формуле $n = u(t) / Uref * 2^N$, где *Uref* - опорное напряжение, задающее диапазон оцифровки (*0...Uref* или *-Uref ... +Uref*), *N* - разрядность двоичного кодирования. Разрядность определяет количество дискрет 2^N в заданном диапазоне измерения, вместе с выбором опоры определяет разрешение. Например, при *N* = 10 и *Uref* = 5.0 В получим разрешение $Uref / 2^N = 5 / 1024 \approx 5$ мВ. Стабильность и точность АЦ преобразования в значительной степени определяются параметрами источника опорного напряжения. С ростом разрядности растут требования по точности к опоре и всем элементам аналогового тракта измерения.

Точность нормируется двумя параметрами:

- разбросом значений, обусловленным несовершенством полупроводниковой технологии (чистота основных материалов и дозирование примесей),

- влиянием температуры кристалла и напряжения питания.

Динамические характристики:

- максимальная производительность, измеряется числом преобразований в секунду [Sample per Second], средние значения – десятки тысяч преобр/с [kSPS], высокие значения – десятки миллионов преобр/с [MSPS]; реальная производительность определяется временем преобразования и алгоритмом взаимодействия аппаратных узлов и программы;

- максимальная частота входного сигнала определяется свойствами аналогового тракта АЦП, алгоритмом работы и временем преобразования.

Существует большое разнообразие типов аналого-цифровых преобразователей (АЦП). Они различаются как потребительскими свойствами – диапазоном измеряемого напряжения, параметрами точности, разрядностью, быстродействием и пр., так и типом преобразования: параллельное, последовательного приближения, сигма-дельта и пр. [12].

АЦП последовательного приближения (или поразрядного уравновешивания) имеет небольшое количество аналоговых компонентов (компаратор и ЦАП), сравнительно простую цифровую часть, поэтому получил наибольшее распространение в МК. Цикл преобразования требует тактирования, число тактов равно разрядности (или на 1 – 3 такта больше), что обуславливает средние характеристики быстродействия.

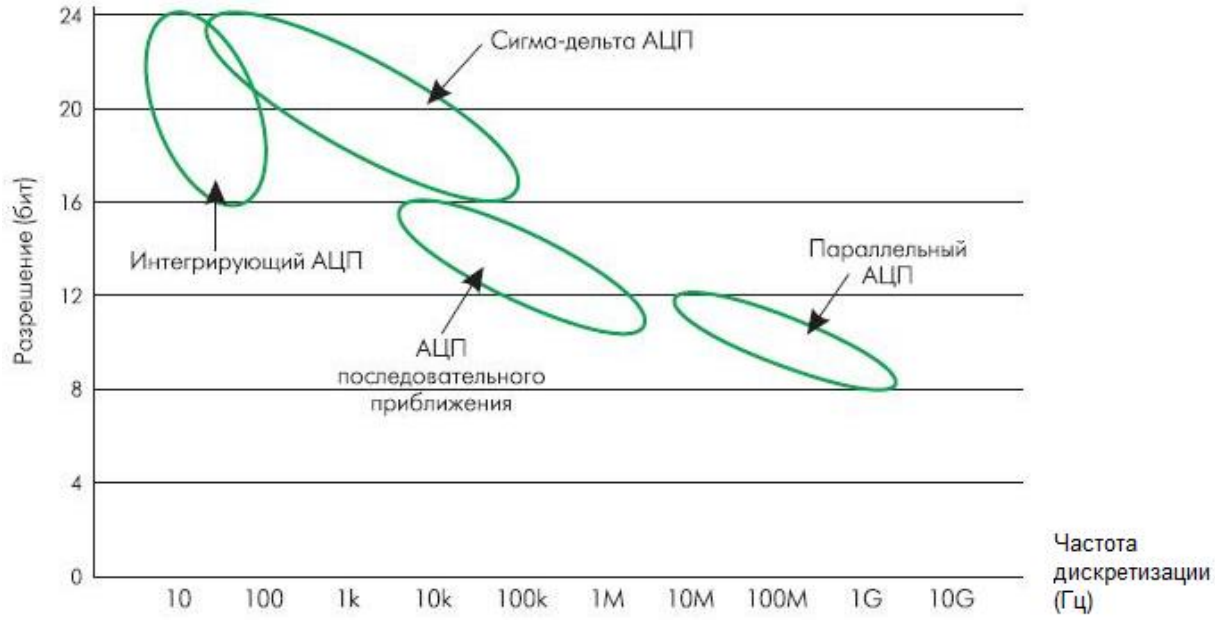


Рис. 6.5

Типовые характеристики АЦП встраиваемых в МК:

- разрядность 8…12 бит,

- время преобразования от долей до сотен микросекунд,

- точностные параметры, как правило, соответствуют разрядности, то есть отклонения реальной характеристики от идеальной не превышают одного - двух квантов.

В качестве источника опорного напряжения (AVref) используются:

- питание цифровой части МК (VCC), что удобно в мобильных приложениях при использовании резистивных датчиков, питающихся от этого же источника;

- встроенный источник (1.23 или 2.56 В) с точностью не хуже ±10%;

- внешний источник опорного напряжения, подключаемый через специальный вход, позволяет гибко выбирать источник с требуемыми параметрами уровня напряжения (ограниченном сверху значением VCC) и точности.

В большинстве случаев встроенный АЦП может преобразовывать сигнал напряжения с нескольких аналоговых входов (с разделением времени). Для выбора

требуемого входа используется набор управляемых аналоговых ключей, объединенных по выходам – так называемый *аналоговый мультиплексор*.

Принцип работы АЦП последовательного приближения заключается в следующем. На первом такте преобразования входное напряжение сравнивается с половиной опорного напряжения, если входное больше, то в старший разряд результата записывается единица, иначе - ноль. На следующем шаге преобразования входное напряжение сравнивается с суммой предыдущего шага (0.5*Uref или 0) и четвертинки опоры (0.25*Uref), в следующий разряд регистра результата записывается "1" или "0" и т.д. Таким образом, преобразование длится число тактов, не меньшее числа двоичных разрядов результата N.

Важно, чтобы в течение этого времени уровень напряжения на входе АЦП не изменялся, иначе растет вероятность неправильного кодирования. Поэтому между аналоговым мультиплексором и собственно входом АЦП располагается устройство *выборки/хранения [track|hold]*. На этапе выборки через входной ключ заряжается конденсатор хранения заряда, на этапе хранения входной ключ размыкается и идет этап АЦ-преобразования. Для выборки и перехода к хранению требуются дополнительные такты, что увеличивает время преобразования.

АЦП в AVR

Основные характеристики АЦП встроенные в МК серии AVR: разрядность 10 бит, опорное напряжение от 2 до 5 В, но не более VCC, число синфазных каналов от 4 до 16, время преобразования от 70 до 280 мкс на канал.

На одно преобразование тратится 13-14 тактов при разрядности N = 10, то есть 10 тактов на собственно АЦ-преобразование и 3-4 такта на выборку. Частота тактирования должна находиться в диапазоне 50...200 кГц, тогда время одного преобразования из 13 тактов составляет 65...280 мкс. Нижнее ограничение частоты связано с постоянной времени разряда конденсатора устройства выборки/хранения – растет погрешность, связанная с утечкой заряда, верхнее – с влиянием цифрового шума. Допустимо увеличение частоты до 1 МГц (время преобразования 13/14 мкс) при снижении эффективной разрядности до 8 бит. Поэтому в ряде современных моделей AVR МК существует возможность выбора 8-разрядного режима преобразования, что упрощает процедуру чтения (один 8-битный регистр результата вместо двух).

На рис. 6.6 приведена структура АЦП встроенного в МК ATmega16. Слева расположены выводы МК: восемь измерительных входов ADC0…ADC7, входы питания АЦП AVCC и GND (AGND), вход/выход источника опорного напряжения AREF. Сверху расположены 8-битная шина данных и программно доступные регистры – состояния и управления (ADMUX, ADCSR) и 16-битный регистр результата ADC.

АЦП состоит из 10-битного ЦАП [10-bit DAC], компаратора с устройством выборки и хранения [Sample and Hold Comparator] и регистра последовательного приближения [Conversion Logic]. Делитель [Prescaller] позволяет выбрать коэффициент деления частоты тактирования fclk для получения требуемой fadc. Дешифратор выбора канала [Mux Decoder] управляет аналоговым мультиплексором синфазных входов [Pos. Input Mux.], мультиплексором инвертирующих входов [Neg. Input Mux.], выбором предварительного усиления [Gain Amplifier] и мультиплексором выбора синфазного или дифференциального режимов [Single Ended / Differential Selection]. Trigger Select используется для выбора одного из аппаратных способов запуска очередного преобразования.

Цикл АЦ-преобразования состоит из выбора канала, запуска (программный и/или аппаратный), ожидания готовности (флаг завершения преобразования ADIF) и чтения регистра результата. Значительное время преобразования (по отношению к длительности машинного цикла) делает актуальным использование прерывания по готовности АЦП, т.е. по завершению предыдущего преобразования.

Аппаратный запуск очередного преобразования бывает двух типов. Наиболее распространен циклический запуск, когда сразу по окончании предыдущего преобразования начинается следующее. В

современных моделях МК появился аппаратный запуск по выбранному прерыванию (аналоговый компаратор, вход прерывания, таймер).

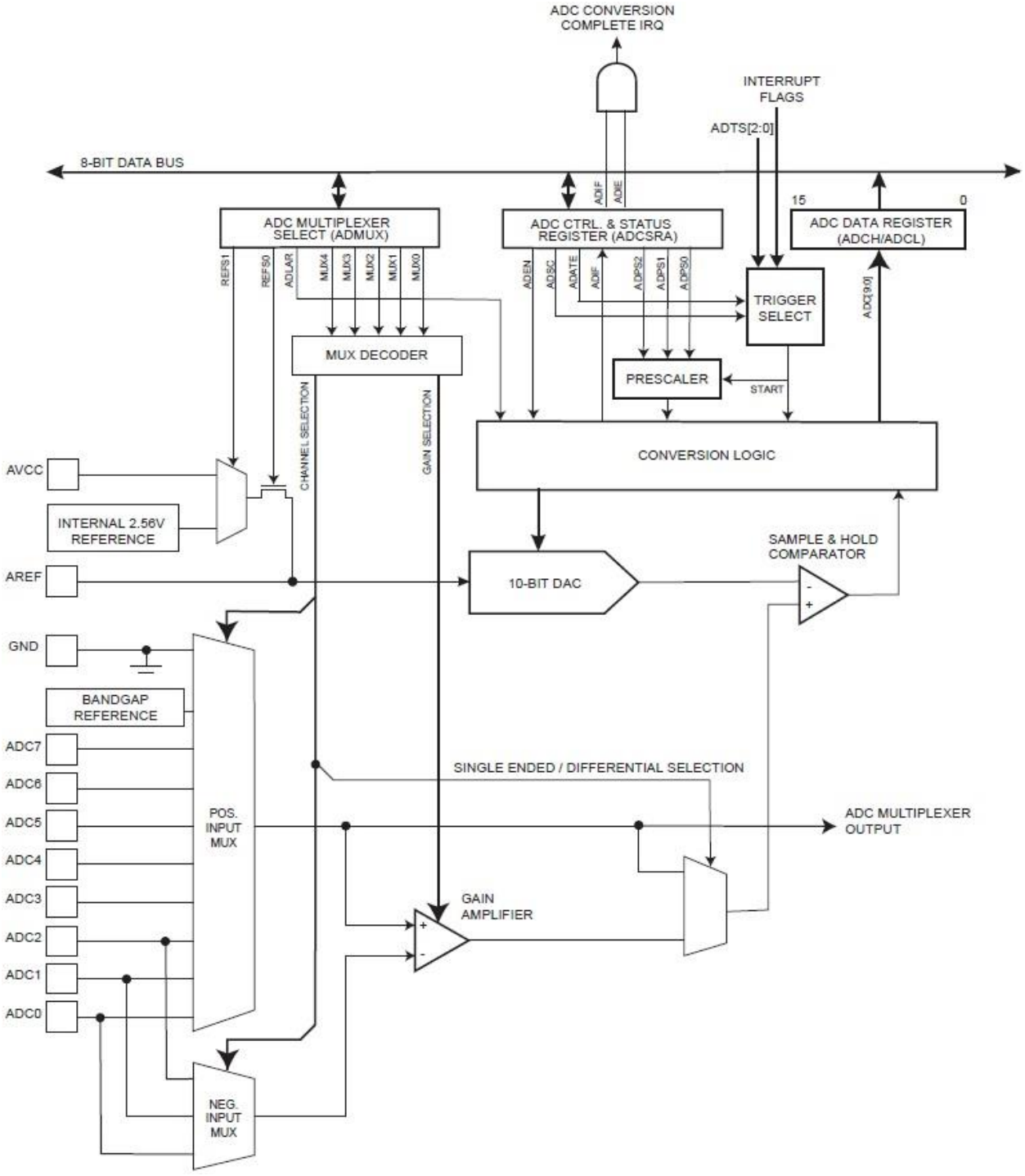


Рис. 6.6. Структура АЦП (ATmega16)

Наличие нескольких входов АЦП не всегда поддерживается соответствующим количеством регистров результата. Так в МК серии AVR единственный регистр результата должен быть прочитан программой до завершения следующего преобразования. Это требование особенно актуально при циклической работе АЦП и поочередном чтении разных аналоговых входов.

Для удобства согласования диапазона измерения АЦП с малосигнальными источниками (датчиками) современные встроенные АЦП имеют между аналоговым мультиплексором и устройством выборки/хранения предварительный усилитель с программируемым коэффициентом усиления (ky = 1, 10, 200 или другой набор).

Предварительное усиление создает опасность усиления ошибки, связанной с протеканием различных токов по «общему» проводу синфазного входа, поэтому при предусилении используются так называемые дифференциальные входы. То есть измеряемое напряжение подается не между «синфазным» входом и «общим» проводом, а между двумя входами (входной дифференциальный усилитель усиливает разностный сигнал, ослабляя тем самым, синфазную ошибку).

Реальная разрешающая способность АЦП с использованием встроенного предусиления снижается, например, для ATmega16 дифференциальный вход с усилением 1 и 10 имеет разрешение 8 бит, с усилением 200 – 7 бит.

Питание аналоговой части требует повышенной стабильности, поэтому в МК с достаточным количеством выводов выделены специальные выводы – AVCC и AGND. Питание может быть как от независимого источника, так и от цифрового через фильтр (LC или RC), соединение аналоговой и цифровой «земель» должно выполняться в одной точке печатной платы.

Для снижения влияния шума цифровых узлов МК на результат АЦ-преобразования в МК AVR предусмотрен специальный режим АЦ-преобразования, во время которого приостанавливается тактирование всех узлов МК, кроме АЦП.

Табл. 6.1. Разновидности АЦП в МК AVR

МК, корпус	Число каналов	Ku	Внутр. опора, В	tпр, мкс/ тысяч пр/с	Запуск	Прочее
tiny15 pdip8, soic8	4 синф 1 дифф	1 20	2.56		Прогр.	
tiny25/45/85 pdip8, soic8	4 синф 2 дифф	1 1, 20	1.1/ 2.56	65/ 15	Прогр., автотриг.	ДТ[1]
tiny24/44/84 soic14	8 синф 12 дифф	1 1, 20	1.1	13/ 76	Прогр., автотриг.	ДТ[1]
mega16, tqfp44/ mega128, tqfp64	8 синф 7 дифф 2 дифф	1 1 10, 200	2.56	13/ 76.9	Прогр.+ автотриг./ Прогр.	–
mega640/1280/2560 tqfp100 mega1281/2561 tqfp64	16 синф 14 дифф 4 дифф	1 1 10, 200	1.1/ 2.56	13/ 76.9	Прогр.	–
mega164/324/644p, tqfp44	8 синф 2 дифф	1 1, 10, 200	1.1/ 2.56	65/ 15	Прогр., автотриг.	–
mega164/324/644/1284a/pa, pdip40, tqfp44	8 синф 2 дифф	1 1, 10, 200	1.1/ 2.56	13/ 15	Прогр., автотриг.	–
mega16/32u4 tqfp44	12 синф 1 дифф	1 1, 10, 40,200	2.56	65/ 15	Прогр., автотриг.	ДТ[1] USB
mega16/32/64M1 tqfp32	11 синф 3 дифф	1 5, 10, 20, 40	2.56	8/ 120	Прогр., автотриг	ДТ[1] CAN
90pwm2-B so24 90pwm3-B so32	8с + 1д 11с + 2д	1, 5, 10, 20, 40	2.56	8/ 125	Прогр., автотриг	DAC10, pwm64MHz
90pwm216 so24 90pwm316 so32	8с + 1д 11с + 2д	1, 5, 10, 20, 40	2.56	8/ 125	Прогр., автотриг	DAC10, pwm64MHz

[1]) ДТ – встроенный датчик температуры

Набор регистров АЦП

Состав регистров и битов управления и состояния зависит от модели МК.

ADCSR (**ADCSRA**) - регистр состояния и управления:

7	**ADEN**	ADC Enable	Разрешение работы АЦП (1)									R/W	0
6	**ADSC**	ADC Start Conversion	Запуск преобразования записью 1									R/W	0
5	**ADFR (ADATE)**	ADC Free Run Select (ADC Auto Trigger En.)	Выбор режима: 0 – однократный, 1 – циклический (Выбор запуска: 0 – прогр, 1 – триггерный (SFIOR))									R/W	0
4	**ADIF**	ADC Interrupt Flag	Уст-ся по завершению преобразования, сбрасывается: - аппаратно при входе в ISR(ADC_vect), - программно записью 1									R/W	0
3	**ADIE**	ADC Interrupt Enable	Разрешение прерывания ADC_vect									R/W	0
2	**ADPS2**	ADC Prescaler Sel 2		0	0	0	0	1	1	1	1	R/W	0
1	**ADPS1**	ADC Prescaler Sel 1		0	0	1	1	0	0	1	1	R/W	0
0	**ADPS0**	ADC Prescaler Sel 0		0	1	0	1	0	1	0	1	R/W	0
			fadc = fclk / …	2	2	4	8	16	32	64	128		

Имена скобках – для более современных моделей, например, ATmega16.

ADMUX - регистр выбора канала и настройки источника опоры, в ранних моделях АЦП (AT90S8535) использовались только биты MUX2…0:

7	6	5	4	3	2	1	0
REFS1	**REFS0**	**ADLAR**	**MUX4**	**MUX3**	**MUX3**	**MUX1**	**MUX0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Биты выбора источника опорного напряжения (ATmega16):

REFS1	**REFS0**	Выбор источника
0	0	Внешний через вход AREF, внутренний выключен
0	1	Аналоговое питание AVCC, внешний конденсатор к AREF
1	0	Резерв
1	1	Внутренний источник 2.56 В с внешним конденсатором к AREF

ADLAR [ADC Left Adjust Result] – бит сдвига результата влево (1), 0 - вправо.

MUX4…0 – выбор канала и предварительного усиления:

0…7 – синфазный канал 0…7 с ku = 1 (достаточно **MUX2…0**),

8…29 – дифференциальные каналы с предварительным усилением 1, 10, 200,

30 – внутренний источник 1.22 В,

31 – потенциал 0 В.

ADC(H/L) – двухбайтный регистр результата. По умолчанию старший бит результата в ADC.9, младший – в ADC0; при **ADLAR=1** старший бит – в ADC15, младший – в ADC6.

SFIOR [Special Function IO Register] – три старших бита (ADTS2…0) выбирают сигнал триггерного запуска преобразования, в качестве которого используется фронт флага прерывания (только в современных моделях):

ADTS2	**ADTS1**	**ADTS0**	Выбор триггерного сигнала
0	0	0	Циклический режим
0	0	1	Аналоговый компаратор
0	1	0	Внешнее прерывание INT0
0	1	1	Совпадение блока сравнения таймера/счетчика 0
1	0	0	Переполнение таймера/счетчика 0
1	0	1	Совпадение блока сравнения В таймера/счетчика 1
1	1	0	Переполнение таймера/счетчика 1
1	1	1	Захват таймера/счетчика 1

Примеры использования АЦП

Простейшая функция выполнения преобразования в режиме программного (то есть однократного) запуска read_adc() рассмотрена ниже. Для настройки АЦП на этапе инициализации необходимо выбрать источник опоры, тактирование и включить модуль:

```
ADMUX=(1<<REFS0);//Выбор опоры +5 В с фильтрацией через вывод AREF
ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1);//ВклАЦП, fadc=fclk/Kps=8M/64=125кГц

unsigned int read_adc(unsigned char channel) {
  ADMUX = channel | (1<<REFS0);//sel ref & ch - выбор канала и опоры
  ADCSRA |= (1<<ADSC); //Start conversion - запуск преобразования
  while ((ADCSRA & (1<<ADIF))==0);//wait ready - ожидание готовности
  ADCSRA |= (1<<ADIF); //reset ready - сброс бита готовности
  return ADC;//read result - чтение результата
} // ************************************
```

Входной параметр функции представляет собой номер канала, он объединяется с выбором опоры для загрузки в ADMUX. Далее выполняется запуск преобразования (установкой бита ADSC), ожидание завершения преобразования (флаг ADIF) и сброс флага готовности, завершается чтением регистра результата.

Данная функция позволяет использовать все имеющиеся на борту каналы в любой последовательности.

При частоте тактирования МК 8 МГц, выборе тактирования АЦП 125 кГц время выполнения функции составляет 114 мкс. Большая часть этого времени тратится на ожидание готовности.

Данный пример находится в папке p3_Analog_IO \ ADC-Rpot-LED7-4 в проекте testADC-m16.aps. Там же находится файл Rpot-adc-LED7-4.dsn содержащий схему, приведенную на рис. 6.7. Ко входу PA0/ADC0 МК D1 (ATmega16) подключен потенциометр RV1 (регулируемый оператором делитель напряжения 0…VCC), вывод AVCC – к питанию VCC (здесь оно идеальное), вывод AREF – к фильтрующему конденсатору C1. Выводы PB4…PB7, PC0…PC7 управляют цифровым индикатором на светодиодах HG, на него выводится результат АЦ преобразования в десятичном формате. Значение 1023 на рис. 6.4 соответствует максимальному измеряемому напряжению 5 В.

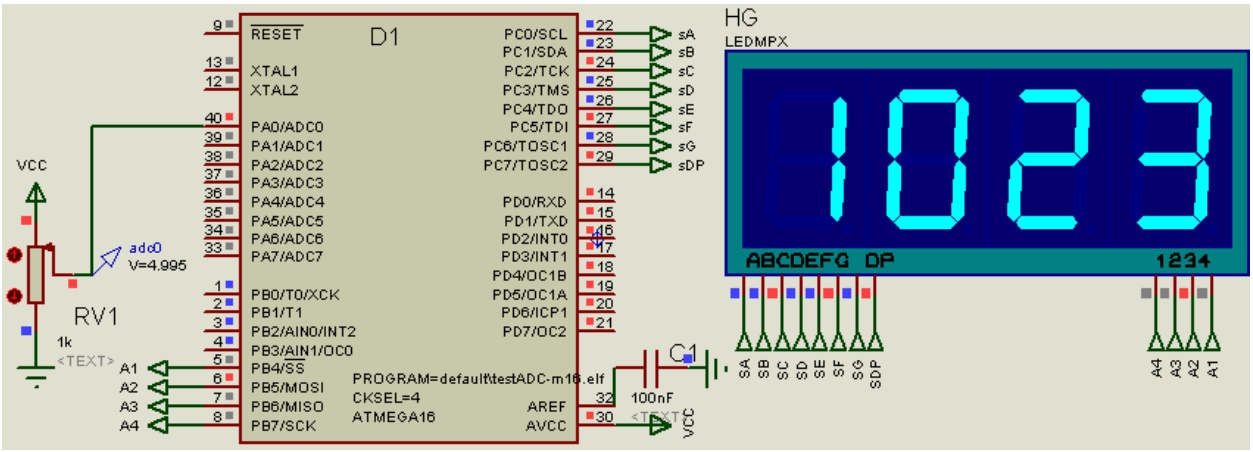


Рис. 6.7

Чаще всего требуется периодическая оцифровка одного или нескольких каналов. Для экономии времени процессора можно использовать прерывание по готовности для

копирования результатов. При необходимости, можно организовать поочередное чтение нескольких каналов в этом же прерывании.

Формально есть несколько вариантов действий с прерыванием по готовности:

1) однократный режим с прерыванием (ADCSR.ADFR=0): сохранить результат, при необходимости выбрать очередной канал оцифровки, запустить очередное преобразование; период работы АЦП – 14 тактов плюс длительность программной выборки канала и запуска;

2) циклический режим с автоперезапуском (ADCSR.ADFR = 1 или ADCSRA.ADATE = 1) является самым быстрым (13 тактов без программных «добавок»), он оптимален для оцифровки одного канала; при оцифровке нескольких каналов надо успеть за первые два такта сменить номер канала, затем прочитать результат предыдущего преобразования (что не просто!);

3) циклический режим с триггерным запуском…

Пример циклического режима с автоперезапуском в задаче последовательной оцифровки 8 каналов (ADC0, ADC1, … ADC7, ADC0, ADC1, …). Настройки инициализации и программа прерывания приведены ниже.

```
  ADMUX=(1<<REFS0);//AVCC+c(AREF), channel=0
//Вкл.АЦП, Запуск, РазрешПрерыв, fadc = fclk/64 = 8 МГц/64 = 125кГц
  ADCSRA=(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1);//+цикл.реж,прер
  sei();//Общее разрешение прерываний
  …
ISR(ADC_vect) { //Функция обслуживания прерывания по готовности АЦП
  static unsigned char channel=0;//Номер канала АЦП
  value[channel] = ADC;//read result – чтение результата
  if(channel < LAST_CHANNEL) channel++;//Выбор следущего канала
  else channel = FIRST_CHANNEL;//или первого
  ADMUX = channel | (1<<REFS0);//sel ref & ch – запись номера канала
  ADCSRA |= (1 << ADSC);
} // ************************************
```

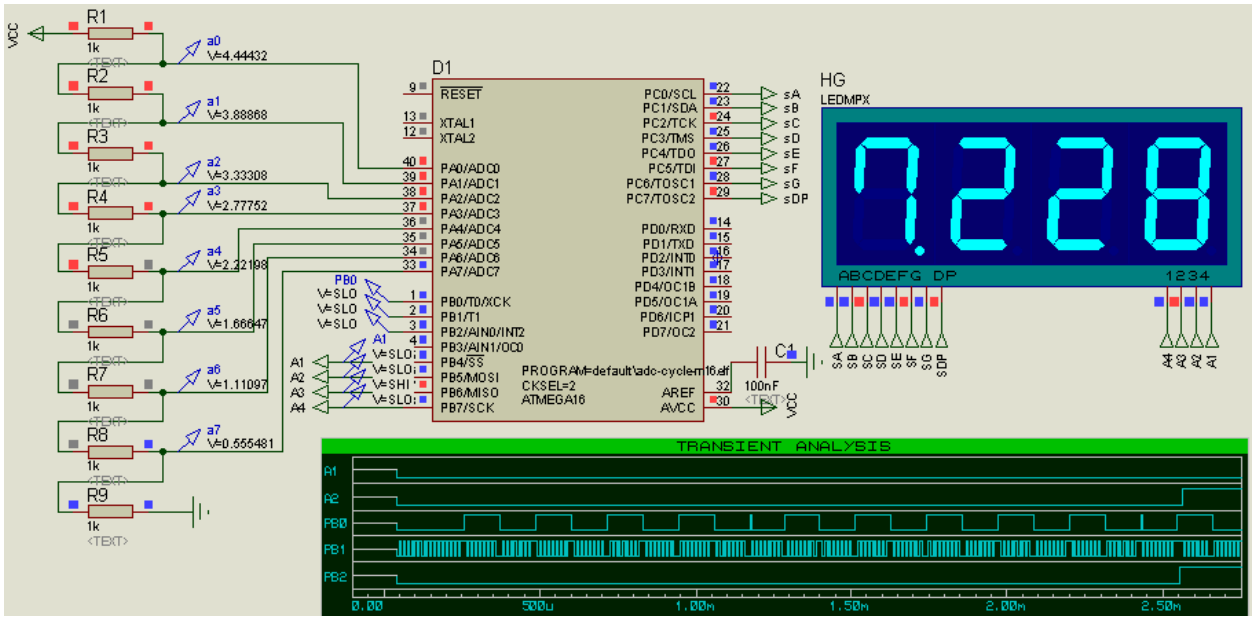


Рис. 6.8

Данный пример находится в папке p3_Analog_IO \ ADC-tst-cycle_mode в проекте adc-cycle-m16.aps. Там же находится файл adcX8-LED7-4.dsn содержащий схему,

приеденную на рис. 6.8. Резистивный делитель R1…R9 запитан напряжением VCC и подключен к восьми входам АЦП МК D1 (ATmega16). Питание АЦП, опора, цифровой индикатор на светодиодах HG подключены аналогично предыдущей схема. На индикатор выводятся результаты АЦ преобразования в формате «N.ddd», где N – номер канала (1…8), ddd – десятичное значение результата (ограниченное числом 999). Значение 228 на 7 канале (рис. 6.5) соответствует напряжению 1.11097 В (между резисторами R7 и R8).

Циклический режим удобнее всего использовать для непрерывных преобразований в одном канале, получаем максимальную частоту преобразования. Пример с этим режимом будет рассмотрен ниже, в разделе ЦАП.

6.4. Встроенный ЦАП

Цифро-аналоговое преобразование – вывод непрерывного во времени сигнала с уровнем, пропорциональным численному значению выходного сигнала u(t) = n* Uref / 2**N, где Uref задает диапазон пропорциональности, N - разрядность.

Структурно ЦАП (рис. 6.6 сверху) состоит из регистра данных РД и резистивной матрицы с аналоговыми ключами. Число n по N цифровым линям записывается в РД, стробирует запись сигнал WR. Для преобразования числа в уровень используются резиствная матрици и набор аналоговых ключей. Старший разряд данных имеет вес Uref/2, следующий – Uref/4 и так далее. Непрерывность обеспечивается регистром данных, выходы которого управляют ключами. Время преобразования определяется быстродействием цифровой части и временем установления в аналоговой цепи, на практике оно составляет десятки - сотни наносекунд.

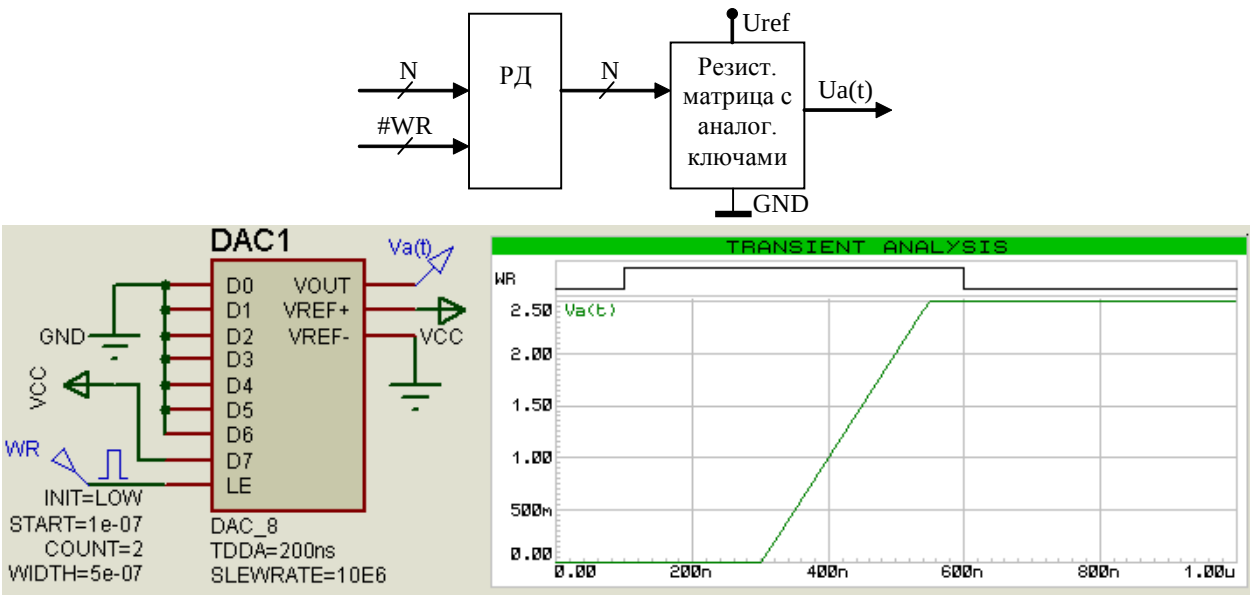


Рис. 6.9

В нижней части рис. 6.9 приведены схема подключения 8 разрядного ЦАП DAC1 и временная диаграмма сигналов в ней (файл DAC8.dsn). На семь младших входов данных (D0…D6) поданы 0, на старший (D7) – 1, на вход строба записи (LE – Latch Enable) – импульс длительностью 500 нс. Выводы VREF+ и VREf- задают диапазон аналогового сигнала на выходе VOUT. Параметр модели ЦАП TDDA= 200 ns задает задержку между цифровым входом и аналоговым выходом, параметр SlewRate = 10e-6 В/с – скорость изменения аналогового сигнала.

ЦАП до недавнего времени не были распространены в МК, что связано с повышенными требованями к питанию качественных аналоговых узлов при повышенной разрядности (более 12). Современным лидером в области встроенных в МК ЦАП (да и АЦП) является фирма Analog Devices с семейством MicroConverter с ядрами 8-разрядных (C51/52) и 32-разрядных (ARM7) микроконтроллеров.

В усовершенствованном подсемействе ATxmega (семейства AVR-8) появились встроенные модули 12 разрядного ЦАП () и 12 разрядного АЦП со значительно меньшим временем преобразования (0.5 мкс).

Для преобразований с невысокой разрядностью удобно использовать ШИМ-выход таймера/счетчика, к которому подключают интегрирующую RC-цепь с постоянной времени, много больше периода ШИМ. Напряжение на выходе RC-цепи имеет постоянную составляющую и пульсацию. Во время импульса напряжение линейно растет, в паузе спадает. Среднее значение напряжения определяется напряжением питания VCC и коэффициентом заполнения ШИМ: Um = VCC * kз = VCC * OCRx / TCNTx_max, здесь OCRx - значение в регистре ширины импульса, TCNTx_max = 256/512/1024 - число дискрет в периоде ШИМ. Частоту тактирования таймера/счетчика ftc следует брать максимальной равной fclk, это позволяет минимизировать значения постоянной времени RC-цепи.

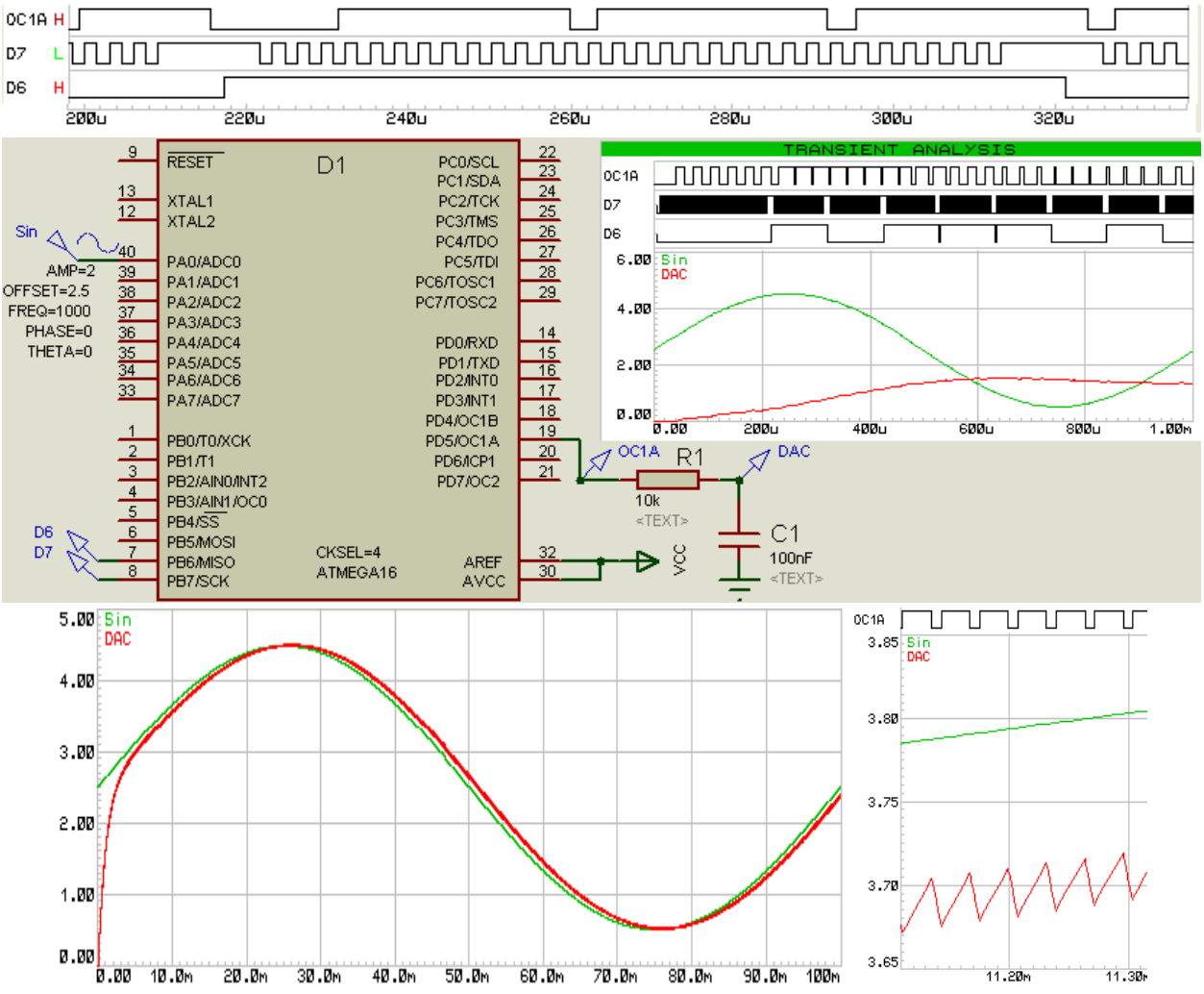
При необходимости можно использовать богатый выбор внешних ИМС ЦАП [12]. Традиционно внешний ЦАП подключается к МК по параллельному интерфейсу, что требует N выходов для передачи N-разрядного кода и не менее одного выхода управления. В последнее десятилетие кроме ЦАП с параллельным интерфейсом появилось большое количество ЦАП с разновидностями последовательного синхронного интерфейса (I2C, SPI, SPORT...), что существенно снижает требуемое количество выводов МК. Мы рассмотрим эти варианты в следующем разделе.

Пример оцифровки и восстановления синусоиды (вариант 1)

На рис. 6.10 представлены программа, схема и результаты моделирования ШИМ-ЦАП (папка adc_DAC-PWM, проект adc_dac-pwm_m16.aps и схема adc_dac-pwm8_m16.dsn). На вход АЦП ADC0 подано синусоидальное напряжение Sin частотой 1000 или 10 Гц, амплитудой 2 В и смещением 2.5 В. АЦП работает в циклическом режиме, в прерывании по готовности результат преобразования сдвигом влево делится на 4 и заносится в регистр сравнения OCR1A. Импульсы быстрой 8-битной ШИМ на выходе OC1A имеют период 256 / 8 МГц = 32 мкс, сглаживаются фильтром R1C1 с постоянной времени τ = 10 кОм * 100 нФ = 1 мс (сигнал DAC). Таким образом, ширина импульса на выходе OC1A пропорциональна напряжению на входе ADC0.

```
unsigned char value;
ISR(ADC_vect) { //Прерывание по готовности АЦП
  OCR1A = ADC>>2;//Чтение результата, преобр.к 8 битам и запись в «ЦАП»
  PORTB ^= (1<<6);//Для оценки периода АЦ-преобразования (104u)
} // ************************************

int main()  {
  DDRB=0xfF; DDRC=0xFF; DDRD=0xFF;//Выходы
  TCCR1A=(1<<COM1A1)|(1<<WGM10);//Режим FastPWM-8bit, nonivert
  TCCR1B=(1<<CS10)|(1<<WGM12);//ftc1=fclk=8МГц, период ШИМ 256/8=32мкс (32u)
```

```
  OCR1A = 128;
// ADC initialization
  ADMUX=(1<<REFS0);//AVCC+c(AREF)
  ADCSRA=(1<<ADEN)|(1<<ADSC)|(1<<ADATE)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1);
  //0x86:ВклАЦП, fadc=fclk/64=8M/64=125кГц
  sei();

  while (1)  { //-----------------------------
      PORTB ^= (1<<7);//Для оценки длительности ОЦ (1u5) и прерывания (12u5)
  }
} // ************************************
```



f = 10 Гц

Рис. 6.10.

Сравнивая графики Sin и DAC видим, что при частоте 1000 Гц напряжение после RC цепи плохо повторяет исходный сигнал, а при частоте 10 Гц – вполне удовлетворительно, хотя и заметна задержка (примерно равная постоянной времени фильтра) и пульсация напряжения (27 мВ при среднем 3.7 В, то есть около 0.7 %). . Увеличение постоянной времени фильтра даст более гладкую форму, но еще больше увеличит сдвиг фазы.

6.5. Терморегулятор

В этом примере мы рассмотрим не только алгоритм управления преобразованиями, но и схемотехнику согласования датчика сигнала со входом АЦП.

Напомним, что *терморегулятором* называется устройство, выполняющее измерение и индикацию температуры, сравнение с заданным уровнем (уставкой) и управляющее выходом по определенному закону регулирования. При использовании дискретного выхода доступны только ключевые законы регулирования (нагреватель, холодильник, в диапазоне, вне диапазона), при наличии аналогового выхода возникает возможность использовать пропорциональный (П), либо пропорционально-интегральный (ПИ) либо пропорционально-интегрально-дифференциальный (ПИД) законы регулирования. Нередко в одном корпусе объединяют несколько независимых каналов измерения/регулирования (до 8), управляемых одним МК.

Датчиком температуры называется устройство, обеспечивающее преобразование температуры в пропорциональный сигнал напряжения или тока. В общем случае делятся на контактные и бесконтактные.

Как правило, в задачах промышленной автоматики нагреваемые объекты имеют значительную тепловую инерцию, то есть значительную постоянную времени изменения температуры (от единиц секунд до десятков часов). В случае малых значений (менее секунды) приходится учитывать постоянную времени датчика температуры. Контактные датчики имеют значительную постоянную времени - от десятков секунд до 0.2 с (термопара без чехла). Бесконтактные датчики (пирометры) имеют наименьшие значения постоянной времени порядка 0.01 с.

Контактные датчики представлены тремя группами:

1) термосопротивления (ТС) или терморезисторы из металлической проволоки (ГОСТ 6651-84, медь для температур от -50 до 180 С° или платина – от -260 до +1100 С°), для меди сопротивление линейно растет с ростом температуры $R(T) = R0\ (1 + ТКС*T)$, T – значение температуры, R0 – значение сопротивления при определенной температуре, обычно при 25 С, $ТКС = 4.28\ 10^{-3}\ К^{-1}$ – температурный коэффициент изменения сопротивления меди, для платины температурная зависимость сопротивления имеет нелинейную составляющую;

2) полупроводниковые терморезисторы, сопротивление зависит от температуры, чаще всего нелинейно: термисторы (ТКС < 0) и позисторы (ТКС > 0), диапазон измерения от -55 до +155 С°,

3) термоэлектрические преобразователи (ТЭП) являются источниками ЭДС пропорциональной температуре (десятки мВ) в диапазоне от -200 до +2500 С° (ГОСТ Р 8.585-2001, МЭК 60584), состоят из спая металлических проволок двух разных металлов, для краткости называется «термопара» [thermocouple], точка спая называется «горячим концом» и располагается в зоне измерения, свободные концы проволок (длиной от 0.4 до 5 м) называются «холодными концами» и подключаются ко входу измерителя напряжения.

Полупроводниковые датчики самые дешевые и малогабаритные, широко применяются в бытовых устройствах. Термосопротивления имеют самый высокий класс точности, имеют значительные габариты и постоянную времени, платиновые имеют значительную цену. Термопары наиболее распространены в технологиях нагрева металлов. Быстродействие контактных датчиков невелико и определяется конструкцией.

Самая малая постоянная времени у бескорпусной термопары – порядка 0.2 с, но постоянная времени нагреваемых объектов обычно гораздо больше.

Бесконтактные датчики используют излучение нагретых тел в оптическом или инфракрасном диапазоне, называются пирометрами. Диапазон температур от -50 до +4000 С°. Являются сложными электронными и поэтому самыми дорогими устройствами с пропорциональным токовым (4-20 мА) или цифровым (последовательные интерфейсы в стандартах RS-232 или RS-485) выходом. Постоянная времени варьируется от 10 до 200 мс.

В библиотеке системы моделирования Proteus VSM полупроводниковые датчики находятся в *Data converters → Temperature Sensors*, термопары и терморезистор платиновый – в *Transducers → Temperature*.

Для *согласования выхода датчика со входом АЦП* надо выбрать схему подключения, определить диапазон напряжений, соответствующий заданному диапазону температур, выбрать источник опорного напряжения. Также следует не забыть о подавлении шумов (помех) хотя бы с помощью емкостного или резистивно-емкостного фильтра нижних частот, постоянная времени может приближаться к постоянной времени самого датчика.

Как привило, сигнал на входе АЦП по форме близок к постоянному, но может содержать колебания и шумы различных частот. *Аппаратный НЧ-фильтр* на входе АЦП в простейшем случае состоит из интегрирующей RC-цепи (постоянная времени $\tau = RC$, полоса пропускания ограничена частотой $f = 2\pi / \tau$) и выполняет следующие функции:

1) подавление не информативной части спектра входного сигнала, прежде всего – высокочастотные шумы (1…10 МГц), связанные с коммутационными процессами в силовых полупроводниковых вентилях (тиристорах, транзисторах) и низкочастотные шумы (50…300 Гц), связанные с промышленной сетью электропитания;

2) так называемый «антиалайзинговый» фильтр для предупреждения появления псевдочастот в спектре восстанавливаемого сигнала.

Выбор типа и конструкции датчика определяет не только диапазон измерения, но и точность измерения (вместе с АЦП и другими элементами тракта измерения). При расчете бюджета погрешностей по методу наихудшего случая общая погрешность есть сумма погрешностей всех элементов тракта измерения.

При подключении полупроводниковых датчиков для преобразования изменения сопротивления в изменение напряжения используют простой делитель напряжения, вторым плечом выступает обычный резистор соответствующей точности. Питание такого делителя удобно выполнять от цифрового питания VCC, это же напряжение используется для опоры АЦП. При расположении датчика RT(T) «сверху» (то есть между VCC и средней точкой) напряжение на входе АЦП U(T) = VCC * R / (R + RT(T)), при расположении RT(T) между нулем питания и средней точкой U(T) = VCC * RT(T) / (R + RT(T)). В параллель нижнему плечу делителя обычно ставят конденсаторы фильтра. Точность измерения в этом случае составит 5…10 %, разрешение зависит от разрядности АЦП и использования диапазона опорного напряжения, реальная разрядность вряд ли превысит 8-9 разрядов.

Использование подобной схемы при подключении высокоточного термосопротивления сталкивается с проблемой влияния падения напряжения от тока, питающего датчик. Для компенсации этого влияния используют трех и четырехпроводные схемы на базе резистивных мостов из высокоточных резисторов.

Естественно, в этом случае стоит использовать хороший внешний источник опорного напряжения. Но все эти меры оправданы при разрядности АЦП 12 и выше.

При подключении термопары требуется предварительное усиление. Как уже упоминалось, можно использовать встроенные в МК предусилитель, но эффективная разрядность 8 или 7 бит означает в лучшем случае относительную точность измерения 0.4 – 0.8 % (при 100 % использовании диапазона АЦП и отсутствии погрешности источника опоры).

Проще и надежнее использовать специализированные инструментальные усилители. Например, для термопары ХА (К) типа предназначены усилители AD595 / AD597 с коэффициентом усиления 245.5, что позволяет получить общий коэффициент передачи 10 мВ/ºС, диапазону температур от 0 до 1000 ºС соответствует напряжение от 0 до 10.0 В. Если диапазон сигнала на выходе усилителя больше диапазона АЦП удобно использовать резистивный делитель (из высокоточных резисторов) общим сопротивлением несколько килоОм, нижнее плечо шунтируется конденсаторами помехоподавляющего фильтра нижних частот. Такое решение позволяет получать точность измерения порядка 2.5…5 % при разрядности 10 (то есть дискретности в одну-две тысячные диапазона АЦП).

АЦ- преобразование занимает небольшую долю времени в цикле измерения, поэтому достаточно использовать программный запуск преобразования с ожиданием готовности. Для снижения влияния шумов следует выбрать частоту тактирования АЦП примерно в середине разрешенного диапазона, кроме того, использовать остановку работы процессора на время преобразования.

Масштабирование

Для индикации значения температуры в цифровом виде в заданном формате (целое число градусов или целое число с единицей, равной 0.1 градуса по шкале Цельсия, а также по шкалам Кельвина или Фаренгейта) в общем случае значение, полученное с АЦП необходимо умножить на дробный коэффициент и прибавить величину смещения (вспомните Пример 1.3).

Программная фильтрация последовательности значений АЦ преобразования позволяет решить следующие задачи:

1) подавление отдельных «ложных» значений (резко выделяющихся из ряда последовательных чтений), вызванных шумом и сбоями в работе цифровых узлов,

2) дополнительное подавление шумов и НЧ-колебаний, пропущенных аппаратным НЧ фильтром на входе АЦП; это, прежде всего, фильтры с полосой пропускания ниже 50 Гц и фильтры с гибко настраиваемой полосой пропускания.

При этом нельзя забывать, что любой фильтр вносит задержку. Это актуально при использовании АЦП в контуре отрицательной обратной связи, так как существует вероятность создания условий для самовозбуждения из-за сдвига фазы на пол периода.

Принципы программного измерения средних и эффективных значений сигналов со значительной переменной составляющей рассмотрены в [БДН МУ к ПЗ по МПСУ, 2007, с.8-10].

Часть 7. Задачи и устройства последовательного интерфейса

7.1. Принципы и преимущества последовательного интерфейса

Протокол

Принцип последовательного интерфейса (ПИ) заключается в передаче N бит данных по одной линии за N тактов времени.

Основное назначение ПИ – обмен данными между МК и другими микросхемами:

1) внутриплатный или внутрисистемный интерфейс, когда один ведущий МК связывается с ведомыми периферийными микросхемами (цифровыми или цифро-аналоговыми);
2) построение распределенных систем управления, когда связываются равноправные МК, состав такой сети может гибко меняться и перенастраиваться;
3) программирование и отладка устройств на базе встраиваемых МК.

В МК/МП для встраиваемых применений чаще всего данные передаются порциями по 8 бит (реже – по 16 или 32 бита). При передаче данные из регистра данных передатчика преобразуют в последовательность бит, для чего используются программные или аппаратные *операции сдвига*. При приеме с помощью тех же операций сдвига последовательность бит вдвигается в регистр данных приемника. Темп этих преобразований задают тактовые импульсы, частота тактирования определяет производительность процесса передачи.

Должны быть согласованы порядок передачи бит (начиная со старшего или с младшего), уровни и фазы сигналов данных и тактов. Для выделения нескольких байт в логически связанную группу (кадр или фрэйм) нередко используют дополнительный сигнал синхронизации.

На рис. 7.1 приведен пример временной диаграммы. Данные передаются, начиная с младшего бита (в данном примере). Линия тактов между посылками находится в низком состоянии, число тактов в одной посылке не меняется и равно n. По фронту тактового импульса передатчик выдвигает очередной бит на линию данных, по срезу приемник считывает уровень с линии данных и вдвигает его.

Вид временной диаграммы, значения временных параметров Tclk, t1, t2, t3, t4, уровни напряжения, число тактов в посылке, параметры фрейма и пр. называются *протоколом* ПИ.

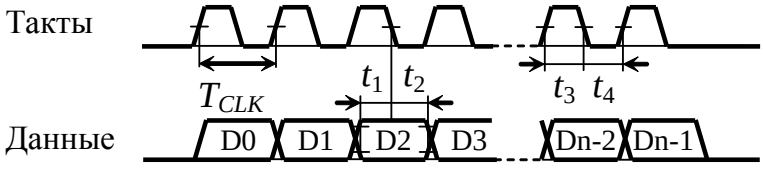


Рис. 7.1

Тактирование

Важным моментом в организации ПИ является вопрос передачи тактового сигнала от передатчика к приемнику.

Самый простой способ – передача тактов от передатчика к приемнику – требует отдельной линии связи и называется *синхронным*. В локальных сетях источник тактовых

импульсов называется ведущим [Master], приемники тактов – ведомыми [Slave], такое распределение ролей характерно для централизованных систем. В этом случае к частоте тактирования предъявляются невысокие требования – она должна находиться в коридоре, ограниченном сверху аппаратно-программными возможностями, снизу – допустимым снижением производительности. Отдельные импульсы могут даже затормаживаться в заранее оговоренном состоянии.

Недостатки синхронного ПИ заключаются не только в дополнительном расходе меди на дополнительный проводник, но в паразитных перекрестных помехах между линиями (взаимная емкость близко расположенных линий и высокая скорость изменения напряжения в линиях), что снижает дальность устойчивой связи.

Примеры синхронного ПИ:

- двухпроводный *I2C* (одна линия тактов и одна линия данных, то есть полудуплекс),
- трехпроводный интерфейс *SPI* (одна линия тактов и две линии данных, то есть дуплекс),
- шестипроводный *McBSP,* (такты, фреймы и данные в одну сторону и такой же набор в обратную).

В общем случае рост числа линий увеличивает производительность и гибкость использования ПИ. Так *I2C* рассчитан на тактирование с частотой 100 и 400 кГц, *SPI* применяется частотами до 10 МГц, встроенные контроллеры этих протоколов распространены во встраиваемых МК/МП общего назначения разрядностью от 8 до 32 бит.

Областью применения синхронного ПИ являются высокоскоростные линии и локальные сети малой размерности, обычно в пределах одной платы, максимум – в пределах одной установки. Чаще всего МК или МП выступает в качестве мастера и оснащен встроенным программно управляемым контроллером соответствующего интерфейса. В качестве ведомых широко используются готовые микросхемы с полностью аппаратной реализацией ПИ (SPI или I2C), выполняющие функции внешних периферийных устройств: ЦАП, АЦП, цифровых датчиков, энергонезависимой памяти и пр.

Многопроводный *McBSP* применяется при частотах тактирования до десятков МГц в цифровых сигнальных процессорах для обмена данными с быстрыми АЦП, ЦАП и кодеками.

Асинхронный принцип тактирования не использует дополнительную линию для передачи тактов, это позволяет экономить на проводах, снижает проблемы перекрестных связей, обеспечивает большую дальности связи. Синхронизация процессов передачи и приема обеспечивается:

- выбором одинакового значения частоты передатчика/приемника и применением высокостабильных генераторов тактов,
- синхронизацией фазы тактовых импульсов приемника за счет передачи в потоке данных дополнительных бит со строго определенными уровнями (старт и стоп биты), это незначительно увеличивает время передачи, но усложняет логику работы передатчика и приемника.

На рис. 7.2 приведена диаграмма передачи байта данных асинхронным протоколом, реализуемым контроллером UART (USART), наиболее распространенным в МК.

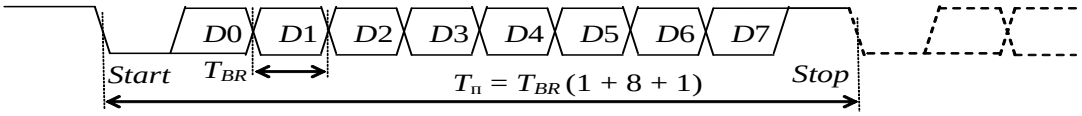


Рис. 7.2

В отсутствии данных линия находится в "1" состоянии. Передача начинается со старт-бита с уровнем "0", далее следуют 8 бит данных, начиная с младшего D0 до старшего D7. Завершается передача стоп-битом с уровнем "1" длительностью от 1 до 2 тактов. Длительность передачи байта с учетом старт и стоп битов составляет не менее 10 тактов.

Частота работы тактовых генераторов передатчика и приемника должна выбираться из стандартного фиксированного ряда (9 600, 19 200, 57 600, 115 200 бод и т. д.), отклонение частоты приемника и передатчика не должно превышать 1–2 %.

Приемник должен выполнять чтение входа данных с частотой, в целое число раз выше частоты тактирования передатчика. При программной реализации достаточно трехкратного превышения, при аппаратной реализации используют 8 или 16 кратное.

При обнаружении первого среза приемник проверяет наличие "0" примерно в середине такта, отсчитанного от среза. Если условие выполнено, приемник продолжает считывание уровней примерно в середине тактов данных и завершает прием проверкой наличия "1" на интервале ожидания стоп-бита.

Принцип асинхронной связи использовался задолго до появления электронных и полупроводниковых устройств в телеграфной связи. Поэтому в МК он был реализован в виде специализированного контроллера одним из первых (UART или USART).

На базе данной диаграммы построены последовательный порт (COM-port) в персональных компьютерах и многочисленные сетевые протоколы для объединения устройств промышленной автоматики в локальные распределенные сети (*Modbus, MPI, ProfiBus etc.*).

В усовершенствованном виде подобные принцип синхронизации и диаграмма нашли применение в протоколе CAN и его разновидностях.

Все упомянутые протоколы ориентированы на передачу одного байта. Для передачи многобайтных посылок требуется дополнительная синхронизация (кадровая), выполняется за счет дополнительных линий связи (SPI, SPORT) или за счет избыточности передаваемых данных.

Еще одной разновидностью ПИ является *однопроводный интерфейс* [1-Wire]. Он применяется для низкоскоростной связи с одним или несколькими пассивными устройствами – датчиками температуры, устройствами памяти, считывателями данных электронных ключей и пр. Здесь одна линия связи используется для синхронизации операций чтения или записи, для передачи данных, а иногда и для питаниия внешнего устройства.

На рис. 7.3 приведена схема подключения МК к микросхеме датчика температуры DS18B20. Слева изображен драйвер двунаправленной линии связи – объединение входа усилителя приемника Rx [Receiver] и выхода усилителя передатчика Tx [Transmitter]. Линия связи имеет притяжку к питанию МК через резистор 4.7 кОм, поэтому в состоянии покоя на линии высокий уровень. Внешнее устройство (1-Wire port) представлено параллельным объединением входа усилителя приемника Rx, открытым стоком драйвера выхода Tx и утечкой входа 5 мкА.

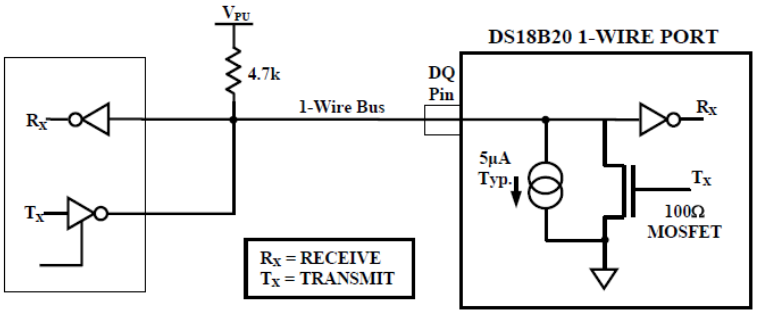


Рис. 7.3

Обмен данными всегда инициирует МК переводом выхода в низкое состояние на 480 мкс, следующие 480 мкс МК ждет ответа от ВУ в виде низкого уровня длительностью 60…240 мкс. Если ответ принят, начинается обмен данными по достаточно сложному протоколу, зависящему от типа ВУ и числа этих устройств на линии. Слот чтения/записи бита длится 100…120 мкс.

Например, для измерения температуры единственным датчиком DS18B20 требуется 480*2+2*100*8 = 960+1600 = 2560 мкс для запуска очередного измерения, время измерения зависит от программно выбранной разрядности результата и составляет 100 мс для 9-битного результата и 750 мс для 12-битного, собственно чтение результата занимает 480*2+4*100*8 = 960+3200 = 4160 мкс.

Дальность связи и помехозащищенность

При увеличении длины связей растут падения напряжения на проводах, при синфазных линиях связи токи соседних линий смешиваются в "общем" проводе. Также растет площадь контуров протекания токов (паразитная индуктивность), что значительно увеличивает вероятность наведения помех внешним магнитным полем. Близкое расположение соседних информационных линий приводит к росту паразитной емкости, приводит при передаче высокоскоростных сигналов к наведению перекрестных помех (ложные импульсы).

Для увеличения устойчивости ко всем перечисленным источникам помех используют различные способы.

На уровне схемотехники:

- увеличение диапазона сигнала до ±5...15 В в RS-232 позволяет поднять дальность связи до 15 м (в "офисных" условиях) за счет снижения влияния падений напряжения и увеличения диапазона помехозащищенности;

- дифференциальные линии связи по стандарту RS-422/485 позволяют поднять дальность до 1000 м при скоростях до 10 МБод в промышленных условиях за счет хорошего подавления синфазных помех.

На уровне линий связи применяют:

- чередование информационных линий с линиями нулевого потенциала (GND) применяется в печатном монтаже и в плоских кабелях для снижения влияния перекрестных связей;

- витая пара проводов имеет минимальную распределенную индуктивность и хорошо согласуется с дифференциальными приемниками/передатчиками;

- экранирование витой пары дополнительно снижает воздействие внешних

магнитных полей;

- коаксиальные кабели объединяют преимущества предыдущих линий связи, но в силу конструктивных неудобств применяются все реже;

- оптоволоконные кабели полностью решают проблему паразитных электромагнитных воздействий, расширяют частотный диапазон, но требуют преобразования электрического сигнала в световой и обратно.

Основные *характеристики* интерфейса:

1) производительность [бит/с = бод = *Bode*, Байт/с],
2) максимальная длина линий связи,
3) среда передачи и форма представления данных:
   - провода и кабели (печатный монтаж на плате, плоский кабель, витая пара проводов, экранированная витая пара, коаксиальный кабель), напряжение/ток, синфазный/дифференциальный сигнал (помехозащищенность), требование гальванической развязки (связь устройств с не эквипотенциальным питанием и для повышения помехозащищенности),
   - радиоканал (радиомодемы или Wi-Fi),
   - оптоволоконные линии связи;

направление передачи:

- однонаправленный – симплексный (источник – приемник),
- двунаправленный с переключением линии на прием или передачу – полудуплексный,
- двунаправленный с раздельными линиями для приема или передачи – дуплексный,

по способу синхронизации передаваемых данных:

- синхронный – синхроимпульсы передаются по отдельной линии связи от *ведущего* устройства к *ведомому/мым*, роль ведущего передается или не передается между устройствами,
- асинхронный – синхроимпульсы добавляются в поток данных (служебные биты),

по числу информационных и служебных линий связи, примеры:

- однопроводный полудуплекс 1-*Wire (Dallas Sem.*): *I/O*, *GND*;
- *двухпроводные* – *RS*-232: *TXD*, *RXD*, *GND*; *RS*-485: A, B, GND (полудуплекс); I^2C (*Phillips*) (*TWI-AVR*): SDA, SCL, GND (полудуплекс); CAN, USB;
- *трехпроводные* – *SPI*: *SCK, MOSI, MISO, #SS, GND;*
- *многопроводные* – *McBSP (multichannel buffered serial ports)*: *BCLKR, BFSR, BDR (reciver), BCLKX, BFSX, BDX (transmitter), GND (*дуплекс) и т.д.

Локальные сети, в свою очередь, имеют дополнительные характеристики:

1) число устройств, объединяемых интерфейсом:
   - два устройства - типа «точка – точка» [*Point-To-Point (PTP)*],
   - более двух устройств – сеть (*network*), в том числе локальная – для ограниченного числа устройств и расстояния),
2) топология, то есть способ объединения устройств в сеть:
   - радиальный (звезда) – центральное устройство имеет отдельные связи с каждым и обеспечивает передачу информации между ними (пример: ПК+USB),

- кольцо – каждое устройство имеет связь на прием от одного устройства и на передачу к другому,
- шина – общий канал связи, каждое устройство имеет связь на прием и на передачу со всеми устройствами сети за счет разделения времени доступа (требуется разбор коллизий),
- комбинированные,

3) способ адресации устройств в сети и размер адресного пространства.

7.2. Функции встроенного контроллера последовательного интерфейса

Несмотря на разнообразие протоколов последовательных интерфейсов, всем им свойственен ряд общих задач, которые в современных МК решаются аппаратным путем и реализуются во встроенном специализированном контроллере.

Рассмотрим типовые узлы и структуру гипотетического контроллера последовательного интерфейса в микропроцессорном устройстве (КПИ в МПУ, рис. 7.4).

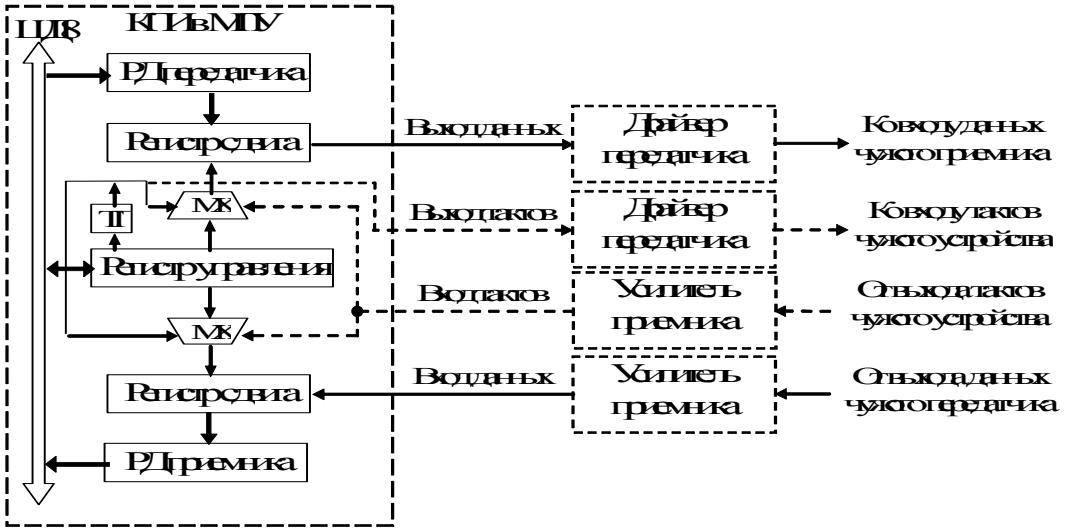


Рис. 7.4

Процедура передачи всегда начинается с загрузки байта данных в регистр данных передатчика (РД передатчика). Обычно этот регистр доступен только по записи. Если контроллер свободен, байт данных параллельным способом грузится в регистр сдвига передатчика. Под действием тактовых импульсов данные из регистра сдвига побитно (начиная со старшего или младшего бита) выводятся на вывод МК, выполняющий функцию выхода данных. После вывода последнего бита данных или замыкающего бита синхронизации процедура передачи закончена.

В этот момент устанавливается специальный битовый флаг завершения передачи в регистре состояния контроллера, может быть разрешено прерывание по этому событию.

Также встречается бит и запрос прерывания, устанавливающийся при загрузке данных из РД передатчика в сдвиговый регистр. Это сигнализирует разрешение записи в РДПер следующего байта

Процедура приема начинается с загрузки байта данных со входа данных приемника в регистр сдвига приемника РСПр под действием тактовых импульсов. По завершении загрузки данные автоматически перегружаются в регистр данных приемника РДПр.

Одновременно устанавливается специальный битовый флаг завершения приема в регистре состояния контроллера, может быть разрешено прерывание для программного чтения полученного байта.

Тактовые импульсы в асинхронных протоколах формируются встроенным тактовым генератором (ТГ) делением тактовой частоты процессора. В синхронных протоколах тактовые импульсы либо формируются встроенным ТГ и передаются через выход тактов (в режиме ведущего),.либо поступают от другого приемопередатчика со специального входа тактов (в режиме ведомого).

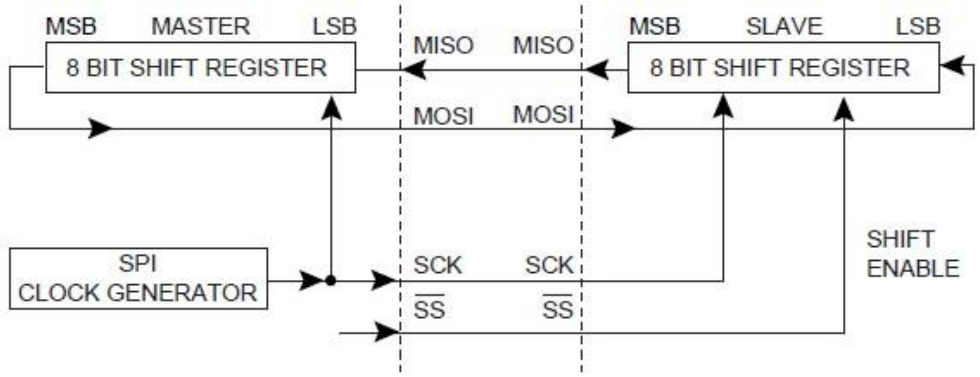
Обычно выводы приемопередатчиков автоматически настраиваются при активации функций связи контроллера, но в некоторых случаях требуется дополнительная настройка портов ввода/вывода.

Для связи на небольшие расстояния используется схемотехника обычных КМОП входов и выходов или схемотехника открытого истока с внешним подтягивающим резистором. Для формирования более помехозащищенного сигналов используются внешние устройства – драйвер передатчика и усилитель приемника.

7.3. Протокол и контроллер трехпроводного синхронного ПИ (SPI)

Протокол синхронного *последовательного периферийного интерфейса SPI* (*Serial Peripheral Interface*) обеспечивает высокоскоростной обмен данными между микроконтроллерами или между МК и переферийными устройствами. При организации локальной сети адресация выполняется аппаратно, что ограничивает размерность сети и затрудняет смену ведущего.

Передача и прием байтов происходят между двумя устройствами одновременно (полный дуплекс), за 8 тактов сдвиговые регистры обоих устройств обмениваются байтами. Для этого используется до четырех выводов (рис. 7.5): три основных – *SCK [Serial ClocK]* такты, *MISO [Master Input or Slave Output]* данные вход ведущего или выход ведомого, *MOSI [Master Output or Slave Input]* данные выход ведущего или вход ведомого, и один вспомогательный – *SS# [Select Slave]* вход выбора ведомого (низким уровнем) для адресации и синхронции кадра (фрейма) в многобайтовом обмене.



Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Рис. 7.5

Гибкость протокола заключается в возможности программного выбора:

- роли Ведущего или Ведомого,
- скорости обмена (частоты тактирования на выходе SCK в режиме ведущего),
- порядка бит в байте (от старшего к младшему или наоборот),
- выбор исходного уровня тактов (высокий или низкий),
- выбор фазы тактов (фронт или срез в момент смены данных).

В МК AVR контроллер SPI кроме высокоскоростной связи с периферийными устройствами (или другими МК) используется для загрузки/выгрузки кодов программы и данных в энергонезависимую память Flash/EEPROM (ISP при Reset = 0).

Контроллер SPI имеет три регистра специальных функций: управления *SPCR [SPI Control Register]*, состояния *SPSR [SPI Status Register]*, данных *SPDR [SPI Data Register]*, распределение бит управления и состояния в регистрах представлено в табл. 7.1.

В режиме Мастера (ведущего) частота тактирования не может превышать 1/2 (в старых моделях – 1/4) тактовой частоты процессора, выбор тактирования представлен в табл. 7.2. В режиме ведомого такты на входе SCK должны иметь высокий и низкий уровни не менее двух тактов частоты процессора.

Варианты временных диаграмм в режимах Mode 0 … Mode 3 представлены на рис. 7.6, соотвтетствие бит настройки – в табл. 7.3.

Таблица 7.1. Регистры и биты трехпроводного интерфейса

SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

7	**SPIE**	SPI Interrupt Enable	Разрешение прерывания SPI
6	**SPE**	SPI Enable	Разрешение работы SPI
5	**DORD**	Data Order	1 – первый младший (LSB), 0 – первый старший (MSB)
4	**MSTR**	Master/Slave Select	1 – ведущий, 0 – ведомый
3	**CPOL**	Clock Polarity	CPOL: 0 – Leading Edge: Rising, Trailing Edge: Falling; 1 – Leading Edge: Falling, Trailing Edge: Rising
2	**CPHA**	Clock Phase	CPHA: 0 – Leading Edge: Sample, Trailing Edge: Setup; 1 – Leading Edge: Setup, Trailing Edge: Sample
1	**SPR1**	SPI Clock Rate Select 1	Выбор делителя тактов 1
0	**SPR0**	SPI Clock Rate Select 0	Выбор делителя тактов 0

SPI Status Register – SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SPIF [SPI Interrupt Flag] – устанавливается аппаратно по завершении 8 тактов приемопередачи или обнулением входа #SS в режиме Ведущего; сбрасывается аппаратно при входе в прерывание SPI или последовательностью чтения регистра SPSR, затем регистра SPDR.

WCOL [Write COLlision flag] – устанавливается аппаратно, если запись в регистр SPDR произошла во время передачи, сбрасывается аппаратно последовательностью чтения регистра SPSR, затем регистра SPDR.

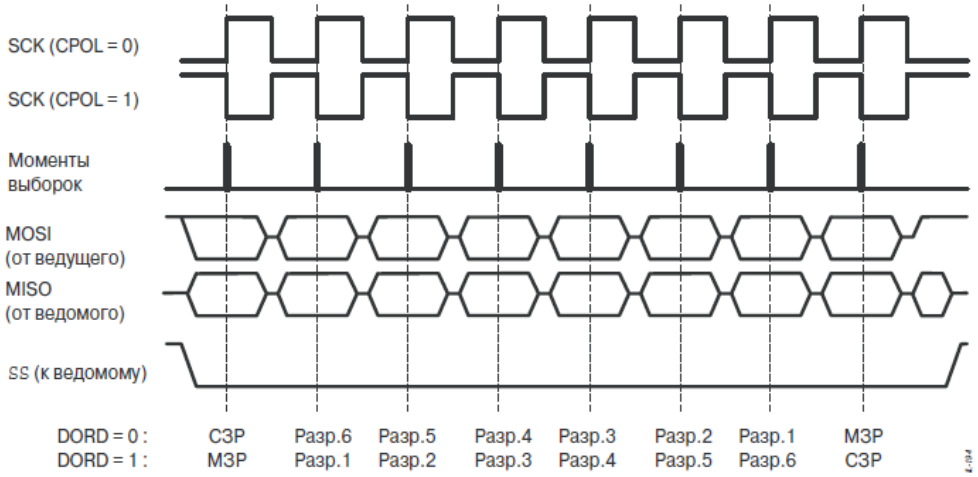
SPI2X [Double SPI Speed bit] – (отсутствует в подсемействе AT90Sxxxx), удваивает частоту тактирования в режиме Ведущего, то есть максимальная частота тактирования равна fclk/2.

Табл. 7.2 Выбор тактирования SPI

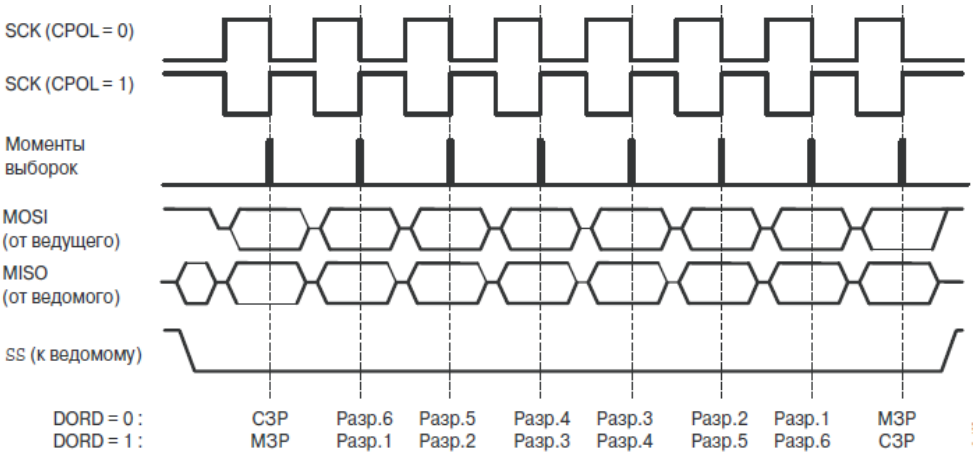
SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Табл. 7.3 Режимы SPI

	Передний фронт	Задний фронт	Режим SPI (Mode)
CPOL = 0, CPHA = 0	Выборка (фронт)	Смена данных (спад)	0
CPOL = 0, CPHA = 1	Смена данных (фронт)	Выборка (спад)	1
CPOL = 1, CPHA = 0	Выборка (спад)	Смена данных (фронт)	2
CPOL = 1, CPHA = 1	Смена данных (спад)	Выборка (фронт)	3



CPHA = 0 (Mode = 0, 2)



CPHA = 1 (Mode = 1, 3)

Рис. 7.6

Процедура обмена байтами инициируется командой записи в регистр *SPDR* на стороне ведущего и длится 8 тактов с частотой выбранной ведущим (1/4, 1/16, 1/64 или 1/128 от частоты тактирования ядра МК f_{CLK}).

В режиме ведущего запись байта в регистр SPDR запускает процесс его передачи и приема от ведомого. По окончании процесса передачи/приема устанавливается флаг SPIF, принятый байт данных доступен по чтению из регистра SPDR.

В режиме ведомого запись в SPDR должна предшествовать моменту начала обмена, инициируемому ведущим. После завершения обмена устанавливается флаг SPIF, принятый байт данных доступен по чтению из регистра SPDR.

Сброс флага SPIF выполняется аппаратно при входе в прерывание SPI или последовательностью чтения регистра SPSR, затем регистра SPDR.

При настройке SPI контроллера требуется его активировать (бит SPE регистра SPCR), выбрать роль Ведущего или Ведомого (MSTR), порядок бит в байте (DORD), режим работы (Mode 0, 1, 2, 3 – сочетание бит CPOL и CPHA), для Ведущего – выбрать коэффициент деления для выбора частоты тактирования (биты SPR1, SPR0 в SPCR и бит SPI2X в SPSR). Для использования прерывания следует установить бит SPIE.

Обратите внимание, что любая операция (передача байта, прием байта, обмен байтами) требуется одинаковой последовательности действий: запись в SPDR, ожидание завершения обмена, чтение SPDR.

Примеры функций настройки [Init] в режиме ведущего [Master] или ведомого [Slave] и универсальная функция приемопередачи без прерывания [Transmit] и в прерывании:

```
void SPI_MasterInit(void)  {
  DDRB = (1<<DD_SS)|(1<<DD_MOSI)|(1<<DD_SCK); //Настройка выходов SPI
  SPCR = (1<<SPE)|(1<<MSTR);//Вкл.SPI, Master, fspi=fclk/4, Mode=0
}
void SPI_SlaveInit(void)  {
  DDRB = (1<<DD_MISO); //Настройка выходов SPI
  SPCR = (1<<SPE); ); //Вкл.SPI
}
char SPI_Transmit(char cData)  {
  SPDR = cData; /* Запись данных запускает передачу и прием */
  while(!(SPSR & (1<<SPIF))); /* Ожидание завершения передачи/приема */
  return SPDR;/* Чтение данных */
}
ISR(SPI_STC_vect) {//В прерывании порядок «обратный» -
  buf_in = SPDR;   // сначала чтение принятого байта,
  SPDR = buf_out;  // затем запись байта для следующего обмена (если требуется)
}
```

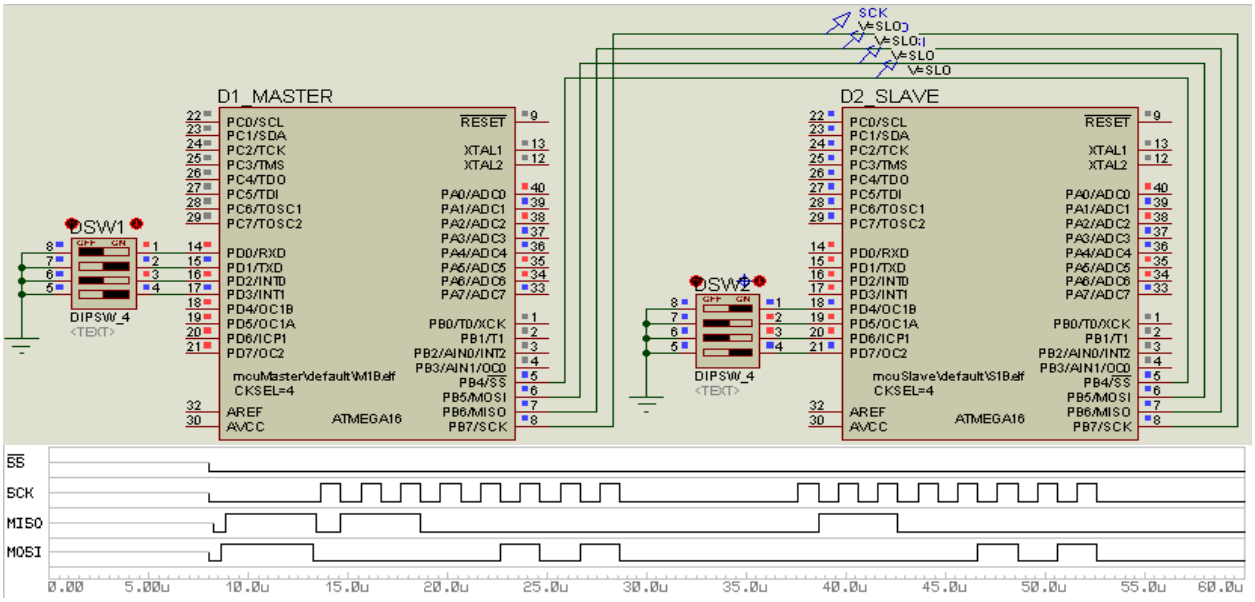
Отладка работы интерфейса в симуляторе AVR-Studio удобна только в режиме Ведущего, можно проверять временные соотношения; моделирование приема затруднительно, как и режима Ведомый.

Гораздо удобнее отладка в среде моделирования Proteus VSM, правда, только с моделями МК семейства Mega.

Пример обмена байтами между двумя МК по SPI интерфейсу

На рис. 7.7 приведена схема, временная диаграмма и тексты программ периодического обмена байтами между двумя МК. Переключатель DSW1 подключен к 4 младшим входам порта B ведущего МК, переключатель DSW2 – к 4 старшим входам

порта B ведомого МК. Выходы порта A обоих МК дублируют состояние двух переключателей. Подробно смотрите этот пример в папке <p4 Serial Interface \ SPI 2mcu>.




```
char in, out;
int main(void) {
  PORTD = 0xff;//Вкл притяжку PD0...PD7
  DDRA = 0xff;//Выходы PA0...PA7
  SPI_MasterInit();
  while (1) { //Основной цикл
    in = PIND & 0x0f; //PD0...PD3 в in
    out = SPI_Transmit(in);//Передача in, прием
out
    PORTA = (in & 0x0f)|(out & 0xf0 );//4 мл
in, 4 ст out
  } // Конец основного цикла
} // Конец функции main()
```

```
char in, out;
int main(void) {
  PORTD = 0xff;//Вкл притяжку PD0...PD7
  DDRA = 0xff;//Выходы PA0...PA7
  SPI_SlaveInit();
  while (1) { //Основной цикл
    in = PIND & 0xf0; // PD4...PD7 в in
    out = SPI_Transmit(in);//Передача in, прием
out
    PORTA = (out & 0x0f)|(in & 0xf0);//4 мл out,
4 ст in
  } // Конец основного цикла
} // Конец функции main()
```

Рис. 7.7

Частота тактирования обоих МК 8 МГц, SPI – 2 МГц, 8 тактов SCK занимают 4 мкс, функция SPI_Transmit выполняется за 9.88 мкс. Замена частного цикла ожидания готовности фиксированной задержкой потребовала 33 *nop*-а, длительность функции – 9.63 мкс.

Аналогично можно организовать обмен данными по прерыванию SPIF, хотя при высоких частотах тактирования SPI это даст незначительный выигрыш времени процессора. Подпрограмма прерывания должна содержать чтение полученного байта из SPDR, затем запись данных в SPDR для передачи.

Подключение нескольких устройств по SPI.

Для организации шины SPI (рис. 7.8) три линии (SCK, MOSI, MISO) подключаются к одноименным выводам, а вот входы выборки ведомых микросхем – каждая к своему выходу ведущего МК. Эти выводы МК не связаны с контроллером SPI, а обычные выводы портов, настроенные как выходы в высоком состоянии. Таким образом, для обмена данными одного ведущего с N ведомыми требуется 3 + N выводов ведущего.

Программа ведущего при обращении к конкретному ведомому сначала переводит его линию выборки CS в низкое состояние, затем инициирует обмен через контроллер SPI требуемого количества байт (кадр) и возвращает линию выборки в неактивное высокое состояние.

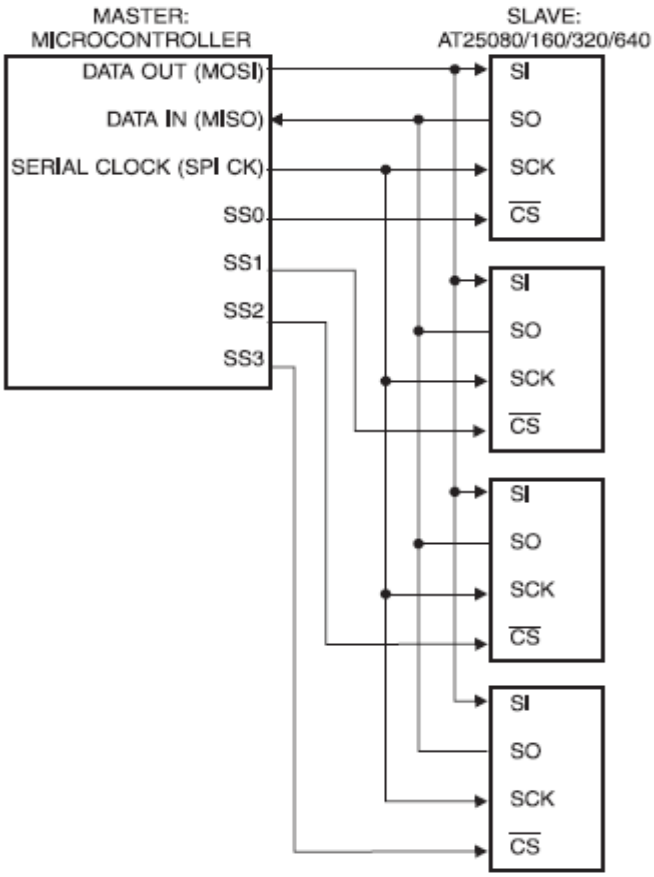


Рис. 7.8

На стороне ведомых (как и на рис. 7.8) чаще всего используются микросхемы со встроенным контроллером ведомого SPI. Это микросхемы памяти, АЦП, ЦАП, «интеллектуальные датчики температуры» и пр. В разделе 7.5 мы подробнее ознакомимся с техническими характеристиками и моделями ЦАП с последовательным интерфейсом, схемы подключения и алгоритмы обмена данными подробно рассмотрены в лабораторной работе № 4 «Последовательный интерфейс в терморегуляторе».

7.4. Протокол и контроллер двухпроводного синхронного ПИ (I2C)

Двухпроводный синхронный протокол I^2C (*Inter IC bus* – шина соединения микросхем, *Phillips*) предназначен для средне скоростного обмена данными в локальной сети (до 128 устройств) между микроконтроллерами или между МК и переферийными устройствами. Роль ведущего может меняться во времени (мультимастер).

Протокол использует только две линии: одну линию для передачи тактов (*SCL – Serial CLock*), другую – для передачи или приема данных (*SDA – Serial DAta*).

Обе линии должны иметь вне микросхем притяжку резисторами 4…10 кОм к плюсу питания (обычно +5 В). В микросхеме к каждой линии подключен приемник и передатчик типа «открытый коллектор» («открытый сток»), такое соединение называется «монтажное ИЛИ». По умолчанию активны только приемники, имеющие высокоомный вход

Если все передатчики, подключенные к линии находятся в ВЫСОКОМ состоянии (транзистор закрыт), линия тоже находится в ВЫСОКОМ. Если хотя бы один передатчик перейдет в НИЗКОЕ состояние (транзистор открыт), линия будет в НИЗКОМ. Поэтому контроллер I2C должен читать состояние линии во время своей передачи для обнаружения конфликтных ситуаций.

Двунаправленный характер передачи сигналов в линии SDA затрудняет применение гальванической развязки.

Терминология

Master	Устройство, которое начинает и заканчивает передачу, выдает такты на линию SCL (Мастер или Ведущий)
Slave	Устройство адресуемое мастером (Ведомый)
Transmitter	Устройство, которое выдает данные на шину (Передатчик)
Receiver	Устройство, которое принимает данные с шины (Приемник)

Устройство на шине I2C может находиться в одном из 4 состояний:

- ведущий передатчик [Master Transmitter],
- ведущий приемник [Master Receiver],
- ведомый передатчик [Slave Transmitter],
- ведомый приемник [Slave Receiver].

Изменение состояния линии SDA выполняется передатчиком только при низком уровне SCL, чтение данных приемником выполняется при высоком уровне *SCL*.

Цикл обмена данными (рис. 7.9) состоит из этапов:

1) формирование последовательности Start;
2) передача адресного пакета SLA+R/W;
3) передача одного или нескольких пакетов данных;
4) формирование последовательности Stop.

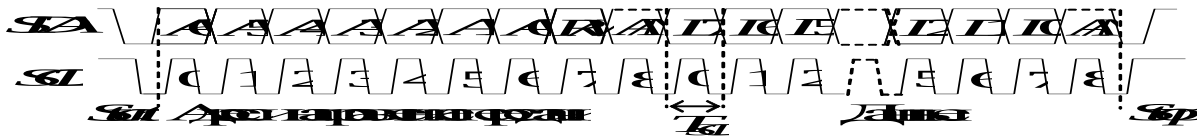


Рис. 7.9

Последовательность *Start* формируется ведущим и состоит из среза на *SDA* при высоком уровне *SCL*.

Передача адреса ведомого (SLA) и направления передачи (R/W) состоит из 9 тактов *SCL*. На SDA ведущий выставляет 7-битный адрес ведомого A6…A0 *[SLave Addr]* и бит выбора направления передачи *R*/*W* (*Write* = 0 – от ведущего к ведомому, *Read* = 1 – наоборот). Далее следует ответ ведомого *ACK*/*NACK*. На девятом такте ведущий выключает свой передатчик на линии SDA и считывает уровень. Если адресуемый ведомый отсутствует или не готов к указанной операции, то он «молчит», уровень задан притяжкой (высокий) – это NACK [*Not ACKnowledge*], если адресуемый ведомый согласен на указанную операцию он обнуляет линию – ACK [*ACKnowledge*].

После первого ответа *ACK* начинается передача байтов данных в выбранном направлении.

Передача каждого байта требует от ведущего 9 тактов SCL и завершается передачей подтверждения/отказа (*ACK*/*NACK)* от приемника данных. Если передача от ведущего к ведомому (запись), то ответ *ACK*/*NACK* формирует ведомый; если передача от ведомого к ведущему (чтение), то ответ *ACK*/*NACK* формирует ведущий, в любом случае ответ формирует приемник данных.

Количество передаваемых байт определяется протоколом более высокого уровня. Передача может быть прекращена приемником выдачей *NACK*, ведущим – выработкой

последовательности *Stop* или *Start* (повторный старт).

Последовательность *Stop* завершает цикл обмена и состоит из фронта на *SDA* при высоком уровне *SCL*.

Для смены направления передачи требуется новый цикл обмена, начинающийся последовательностью Start или Stop-Start.

Роль ведущего жестко не закреплена, любое устройство может стать ведущим, когда на обоих линиях устанавливается высокий уровень. Для разбора *коллизии*, возникающей при одновременной попытке двух МК стать ведущими, оба должны читать сигналы на линиях *SDA* и *SCL*. Устройство, которое обнаружит отличие между передаваемыми им сигналами и сигналами на линиях отказывается от роли ведущего до окончания сеанса захвата.

Контроллер I2C в МК AVR-8

Называется TWI [Two-Wire serial Interface], структурная схема на рис. 7.10, имеет 5 программно доступных регистров.

TWBR [TWI Bit Rate Register] – регистр выбора скорости (частоты) f_{SCL}.

TWCR [TWI Control Register] – регистр управления.

TWSR [TWI Status Register] – регистр состояния.

TWDR [TWI Data Register] – регистр данных, по записи на передачу, по чтению на прием.

TWAR [TWI (Slave) Address Register] – 7-битный регистр адреса (только для ведомого).

Частота тактирования шины f_{SCL} (задается только для ведущего) обычно выбирается из значений 100 кГц (нормальная), 400 кГц (ускоренная), абсолютная точность не важна. $f_{SCL} = f_{CLK} / (16 + 2TWBR \cdot 4^{TWPS})$, где TWBR – численное однобайтное беззнаковое значение регистра, TWPS – двухбитное значение из регистра TWSR.

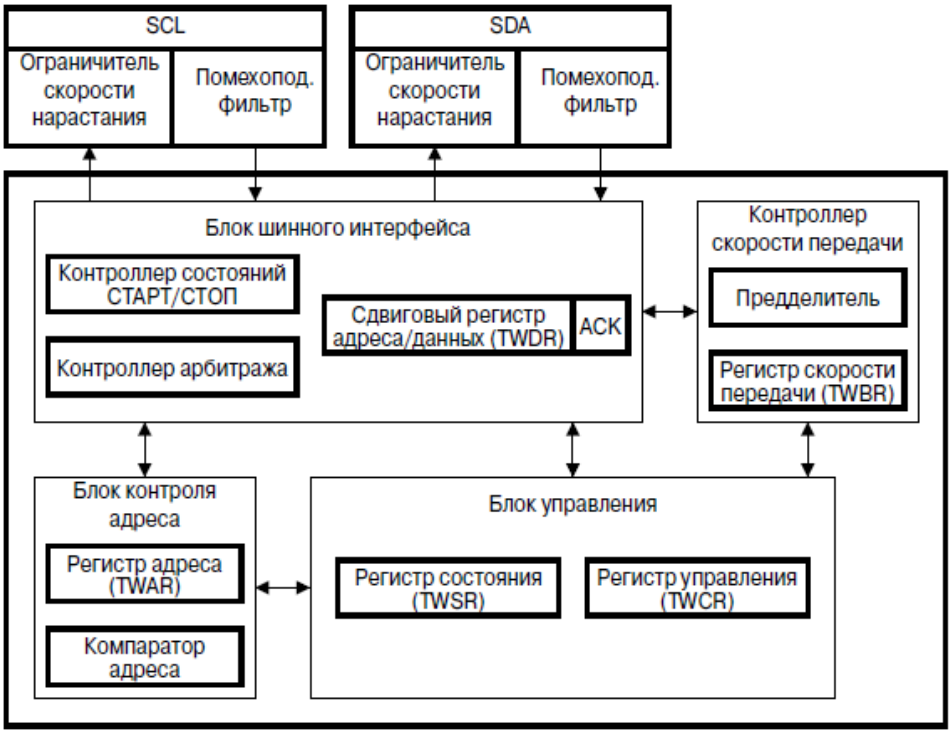


Рис. 7.10

Регистр управления TWCR.

Бит 7 TWINT [TWI INTerrupt flag] – флаг прерывания, устанавливается аппаратно после выполнения очередной операции, когда модуль ожидает отклика со стороны программы; сброс выполняется только программно записью «единицы» в TWINT; это «спусковой крючок» конечного автомата, программа анализирует код состояния в битах TWS7…TWS3 регистра TWSR и подготавливает выполнение конкретного набора действий битами TWCR, после чего вновь установится флаг и т. д.

Бит 6 TWEA [TWI Enable Acknowledge bit] – разрешение ответа ACK (при 1), сигнализирует адресату согласие на выполнение операции, например, подтверждение приема адреса или байта данных; при 0 формируется NACK, то есть отказ от выполнения следующей операции.

Бит 5 TWSTA [TWI START Condition bit] – запрос на формирование СТАРТа (последовательности изменения SDA, SCL) и получения статуса Мастера; выполняется при отсутсвии другого Мастера на линии или ждет STOP активного Мастера.

Бит 4 TWSTO [TWI STOP Condition bit] – запрос на формирование СТОПа (последовательности изменения SCL, SDA) и отказа от статуса Мастера; в режиме Ведомого помогает выйти из состояния ошибки без генерации последовательности СТОП.

Бит 3 TWWC [TWI Write Collision flag] – конфликт записи (коллизия); возникает при выполнении записи в TWDR при TWINT = 0, сбрасывается записью в TWDR при TWINT = 1.

Бит 2 TWEN [TWI Enable bit] – активирует блок TWI, настраивает выводы SCL и SDA (ограничение скорости изменения выхода, фильтрация входа).

Бит 1 – не используется.

Бит 0 TWIE [TWI Interrupt Enable] – разрешение прерывания по флагу TWINT (при общем разрешении SREG.I = 1).

Регистр состояния TWSR. Старшие 5 бит (именуются TWS7…TWS3) доступны только по чтению и показывают код состояния конечного автомата после установки флага TWCR.TWINT (запрос прерывания). Младшие два бита TWPS1, TWPS0 доступны по записи/чтению, задают частоту SCL, бит 2 не используется.

Число состояний автомата определяется числом бит и равно 32, для выделения кода состояния в регистре TWSR используется код *0xf8*.

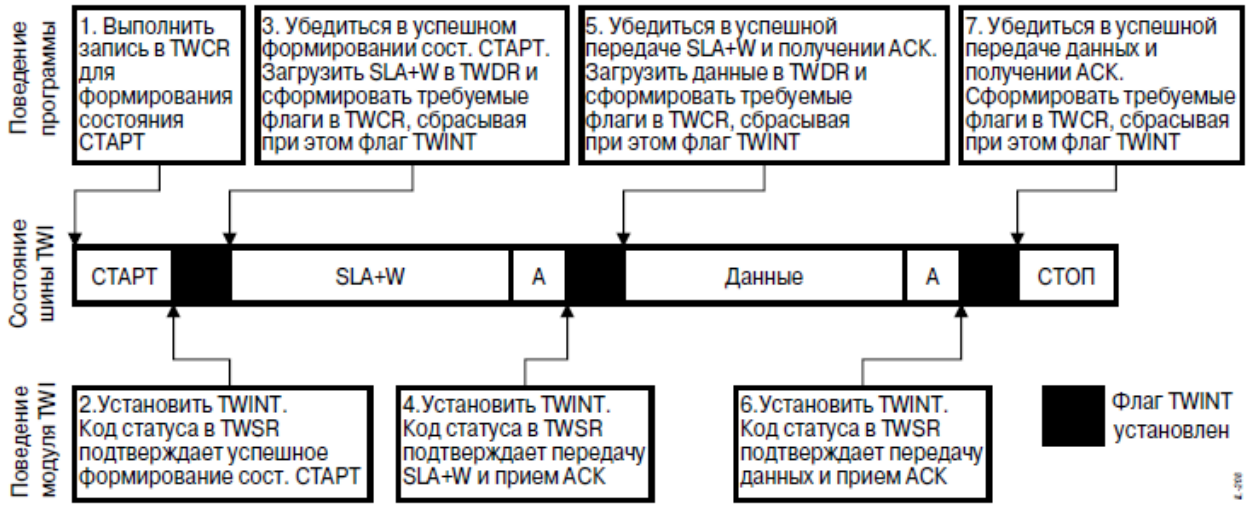


Рис. 7.11 Пример взаимодействия программы с модулем TWI

На рис. 7. 11 подробно показано взаимодействие программы ведущего МК с модулем TWI при передаче одного байта от ведущего к ведомому.

Формирование очередного действия выполняется записью нужного кода в регистр управления. Выбор момента и типа действия синхронизируется установкой флага TWINT и последующим анализом кода состояния в старших 5 битах TWSR. 5 бит описывают 32 состояний конечного автомата модуля, поэтому логика поведения достаточно сложная, хотя большую часть занимают нештатные ситуации.

Подробное перечисление кодов состояния и вариантов возможных действие занимает в [3, 4] порядка 15 страниц, поэтому ниже я привожу краткое и читабельное описание от опытного программиста, обладающего чувством юмора [http://easyelectronics.ru/avr-uchebnyj-kurs-ispolzovanie-avr-twi-dlya-raboty-s-shinoj-iic-i2c.html].

Коды состояний для режима Master

- **0x00 Bus Fail** Автобус сломался… эээ в смысле аппаратная ошибка шины. Например, внезапный старт посреди передачи бита.
- **0x08 Start** Был сделан старт. Теперь мы решаем, что делать дальше, например, послать адрес ведомого.
- **0x10 ReStart** Был обнаружен повторный старт. Можно переключиться с записи на чтение или наоборот. От логики зависит.
- **0x18 SLA+W+ACK** Мы отправили адрес с битом записи, а в ответ получили ACK от ведомого. Значит можно продолжать.
- **0x20 SLA+W+NACK** Мы отправили адрес с битом записи, а нас послали NACK. Обидно, сгенерим ошибку или повторим еще раз.
- **0x28 Byte+ACK** Мы послали байт и получили подтверждение, что ведомый его принял. Продолжаем.
- **0x30 Byte+NACK** Мы послали байт, но подтверждение не получили. Видимо ведомый уже сыт по горло нашими подачками или он захлебнулся в данных. Либо его ВНЕЗАПНО посреди передачи данных украли инопланетяне.
- **0x38 Collision** А у нас тут клановые разборки — пришел другой мастер, по хамски нас перебил, да так, что мы от возмущения аж заткнулись. Ничего I’l be back! До встречи через n тактов!
- **0x40 SLA+R+ACK** Послали адрес с битом на чтение, а ведомый отозвался. Хорошо! Будем читать.
- **0x48 SLA+R+NACK** Крикнули в шину «Эй ты, с адресом XXX, почитай нам сказки» А в ответ «Иди NACK!» В смысле на запрос адреса с битом чтения никто не откликнулся. Видимо не хотят или заняты. Также может быть, никого нет дома.
- **0x50 Receive Byte** Мы приняли байт. И думаем что бы ответить ведомому. ACK или NACK.
- **0x58 Receive Byte+NACK** Мы приняли байт от ведомого и сказали ему «иди NACK!» И он обиженый ушел, освободив шину.

Коды состояний для режима Slave:

- **0x68 Receive SLA+W LP** Мы были мастером, трепались с подчиненными по шине. И тут появляется на шине другой, более равный, мастер, перебивает нас и молвит «Уважаемый XX, а не возьмете ли вы вот эти байтики…» Что ж, он круче. Придется бросать передачу и брать его байты себе.
- **0x78 Receive SLA+W LP Broadcast** Были мы, как нам казалось, самыми крутыми мастерами на шине. Пока не появился другой мастер и перебив нас прогундосил на всю шину: «Эй, слышь тыыы. Слушай сюда» Девайсы-лохи, с неотключенными

широковещательными запросами подчиняются. Остальные отмораживаются и всякое Broadcast-быдло игнорируют.

- **0x60 Receive SLA+W** Сидим на шине, никого не трогаем, ни с кем не общаемся. А тут нас по имени… Конечно отзовемся :)
- **0x70 Receive SLA+W Broadcast** Ситуация повторяется, но на этот раз слышим уже знакомое нам: «Слышь, тыыыы». Кто? К кому? Игнорируем Broadcast запросы? Или нет? Зависит от моральных качеств программы.
- **0x80 Receive Byte & 0×90 Receive Byte Broadcast** Принимаем байты. От кого и в каком виде не важно. Решаем что сказать «Давай еще (ACK)» или «Иди NACK». Тут уже по обстоятельствам.
- **0x88 Receive Last Byte & 0×98 Receive Last Byte Broadcast** Приняли последний байт и распихиваем по карманам.
- **0xA0 Receive ReStart** Ой, у нас «Повторный старт». Видимо то что пришло в первый раз, был таки адрес страницы. А сейчас пойдут данные…
- **0xB0 Receive SLA+R LP** Слали мы что то, слали, а тут нас перебивает другой мастер, обращается по имени и говорит «А ну XX, зачитай нам что-нибудь из Пушкина» Что делать, приходится читать.
- **0xA8 Receive SLA+R** Либо просто к нам какой-то другой мастер по имени обращается и просить ему передать байтиков.
- **0xB8 Send Byte Receive ACK** Ну дали мы ему байт. Он нам ACK. А мы тем временем думаем слать ему еще один (последний) и говорить «иди NACK». Или же у нас дофига их и можно еще пообщаться.
- **0xC0 Send Last Byte Receive NACK** Дали мастеру последний имеющийся байт, а он нам «иди NACK». Хамло. Ну и хрен с ним. Уходим с шины.
- **0xC8 Send Last Byte Receive ACK** Дали мастеру последний имеющийся байт, а он требует еще. Но у нас нету, так что разворачиваемся и уходим с шины. А он пусть карманы воздухом наполняет (в этот момент мастер начнет получать якобы от slave 0xFF байты, на самом деле это просто чтение висящей шины).

В простых случаях ожидание установки флага TWINT (while (!(TWCR & (1<<TWINT))) ;), анализ состояния и программные действия выполняются в основном цикле. Для ведущего логика последовательности таких действий достаточно детерминирована, с ведомым дело обстоит сложнее.

Таблица 7.2. Константы и функции для обмена байтами между Ведущим и Ведомым

`#define Fclk 8000000L`
`#define FSCL 400000L`
`#define TWBR_SET (Fclk/FSCL - 16)/2`
	Задание частоты тактирования I2C (400 кГц) через частоту процессора

`Ведущий`	Ведомый
Коды состояния ведущего (часть) #define START 0x08 /* Сформирован СТАРТ */ #define RESTART 0x10 /* Сформирован СТОП */ #define SLA_W_ACK 0x18 /* Передан SLA_W и получен Ack */ #define SLA_W_NACK 0x20 /* Перед… и не получен Ack */ #define BYTE_W_ACK 0x28 /* Переданы данные и получен Ack */ … #define SLA_R_ACK 0x40 /* Передан SLA_R и получен Ack */ #define SLA_R_NACK 0x48 #define BYTE_R_ACK 0x50 /* Получены данные и передан Ack */ …	Коды состояния ведомого (часть) #define Receive_SLA_W 0x60 /* Принят свой SLA_W и передан Ack */ … #define Receive_Byte 0x80 /* Принят байт и передан Ack */ #define Receive_Last_Byte 0x88 /* Принят байт и передан NAck */ … #define Send_Byte_Receive_ACK 0xb8 /* Передан байт и получен Ack */ … Общее именование бит #define R_W_bit (1<<0) /* */ #define Ack 1 /* Подтверждение приемника */ #define NoAck 0 /* Нет подтверждения */

```
void init_i2c_Master(void) {//Настройка
  TWCR=(1<<TWEN);//Вкл. TWI
  TWBR=TWBR_SET;//Задание частоты SCL
}//-------------
void i2c_start(void) {
  TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
  while (!(TWCR & (1<<TWINT))) ;
  if((TWSR & 0xF8) != START) error_i2c();
}//-------------
void i2c_stop(void) {
  TWCR=(1<<TWINT)|(1<<TWSTO)|(1<<TWEN);
}//-------------
void i2c_SLA_W(char byte) { //MT
  TWDR = (byte & ~R_W_bit);
  TWCR=(1<<TWEN)|(1<<TWINT);//Send SLA+W
  while (!(TWCR & (1<<TWINT))); //Wait TWINT
  if((TWSR & 0xF8) != SLA_W_ACK) error_i2c();
}//-------------
void i2c_SLA_R(char byte) { //
  TWDR = (byte | R_W_bit);
  TWCR=(1<<TWEN)|(1<<TWINT);//Send SLA+R
  while (!(TWCR & (1<<TWINT))); //Wait TWINT
  if((TWSR & 0xF8) != SLA_R_ACK) error_i2c();
}//-------------
void i2c_write(char byte) {
  TWDR = (byte);
  TWCR=(1<<TWEN)|(1<<TWINT);//Send byte
  while (!(TWCR & (1<<TWINT)));//Wait TWINT
  if((TWSR & 0xF8) != BYTE_W_ACK) error_i2c();
}//-------------
unsigned char i2c_read(char ack)  {
  TWCR =(1<<TWEN)|((ack)<<TWEA)|(1<<TWINT);
  while (!(TWCR & (1<<TWINT)));//Wait TWINT
  return TWDR;
}//-------------
```

```
void init_i2c_ Slave(char addr) {//Настройка
  TWCR=(1<<TWEN) |(1<<TWEA);//Разр. Подтвержд.
  TWAR=addr;//Адрес ведомого
}//-------------
char i2c_slave(char inbyte) {
  volatile char static last_code = 0;
  volatile static char code = 0, outbyte = 0;
  while (!(TWCR & (1<<TWINT))); //Wait for TWINT
flag set.
  code = TWSR & 0xF8;
  switch(code) {
    case Receive_SLA_R:
         TWDR = inbyte;
    case Receive_SLA_W:
         last_code = code;
         TWCR |= (1<<TWEA)|(1<<TWINT);
         break;
    case Receive_Byte:
    case Receive_Last_Byte:
         if(last_code == Receive_SLA_W)
         { outbyte = TWDR; last_code = 0; }
         TWCR |= (1<<TWEA)|(1<<TWINT);
         break;
    case Send_Byte_Receive_ACK:
    case Send_Byte_Receive_NACK:
    case Send_Last_Byte_Receive_ACK:
         last_code = 0;
         break;
    case Receive_Restart:
         break;
  }
  return outbyte;
}
```

Ожидание установки флага TWINT может выполняться по прерываниям, если при настройке модуля TWI установить бит TWCR.TWIE. Тогда структура программы обслуживания прерывания будет сложной (сплошные switch { case…}), причем наибольший объем уйдет на редко выполняемые действия, но быстрой по выполнению, так как действий в каждой ветке мало.

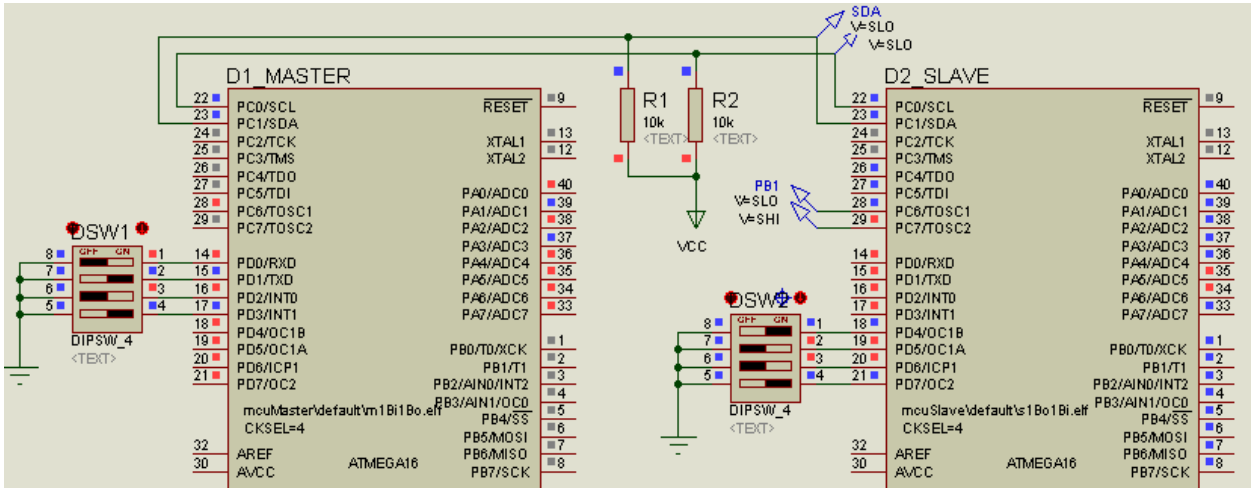
В табл. 7.2 приведены выдержки программных действий ведущего и ведомого для работы в основном цикле. Полный состав находится в файлах <i2c.c>, <i2c.h> и испольщется в примере, рассмотренном ниже.

Симулятор AVR-Studio не поддерживает TWI, поэтому ничего кроме, правильной адресации регистров и бит проверить нельзя.

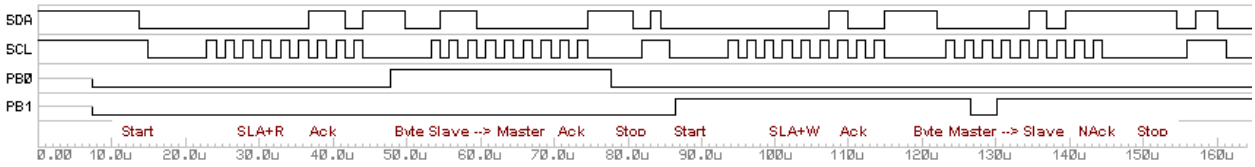
Proteus VSM моделирует все все функции TWI в библиотеке AVR2 (семейства Mega и Tiny).

Пример обмена байтами между двумя МК по I2C (TWI) интерфейсу

На рис. 7.12 приведена схема, временная диаграмма и программы периодического обмена байтами между двумя МК (аналогично примеру для SPI-интерфейса). Переключатель DSW1 подключен к 4 младшим входам порта B ведущего МК, переключатель DSW2 – к 4 старшим входам порта B ведомого МК. Выходы порта A обоих МК дублируют состояние двух переключателей. Подробно смотрите этот пример в папке <I2C 2mcu>, Fclk = 8 МГц, частота f_{SCL} = 400 кГц, передача и прием одного байта занимают около 146 мкс.



а



б

```
#include "i2c.h"
#define SLA_MCU_2 2

char in, out;
int main(void) {
  PORTD = 0xff;//Вкл притяжку PD0...PD7
  DDRA = 0xff;//Выходы PA0...PA7
  init_i2c_Master();
  while (1) { // Основной цикл
    in = PIND & 0x0f; //PD0...PD3 в in
// Прием байта out:
    i2c_start();
    i2c_SLA_R(SLA_MCU_2);
    out = i2c_read(Ack);
    i2c_stop();
// Передача байта in:
    i2c_start();
    i2c_SLA_W(SLA_MCU_2);
    i2c_write(in);
    i2c_stop();
  PORTA = (in & 0x0f)|(out & 0xf0 );// 4 мл in, 4 ст out
  } // Конец основного цикла
} // Конец функции main() ============================
//Вариант с функц SPI_Transmit (пока не опроб)
char in, out;
int main(void) {
  PORTD = 0xff;//Вкл притяжку PD0...PD7
  DDRA = 0xff;//Выходы PA0...PA7
  SPI_MasterInit();
  while (1) { //Основной цикл
    in = PIND & 0x0f; //PD0...PD3 в in
    out = SPI_Transmit(in);//Передача in, прием out
    PORTA = (in & 0x0f)|(out & 0xf0 );//4 мл in, 4 ст
out
} } // Конец основного цикла и функции main()
```

```
#include "i2c.h"
#define SLA_MCU_2 2

char in, out;
int main(void) {
  PORTD = 0xff;//Вкл притяжку PD0...PD7
  DDRA = 0xff;//Выходы PA0...PA7
  SPI_SlaveInit();
  while (1) { //Основной цикл
    in = PIND & 0xf0; // PD4...PD7 в in
    out = i2c_slave(in);
    PORTA = (out & 0x0f)|(in & 0xf0);//4
мл out, 4 ст in
  } // Конец основного цикла
} // Конец функции main()
```

в

Рис. 7. 12. Пример обмена байтами двух МК по интерфейсу I2C

Проект находится в папке Primer/I2C mcu/ и состоит из схемы с моделями двух МК в Proteus VSM <I2C-2mcu.dsn> и проектов программ для МК MCUmaster и MCUslave.

7.5 ЦАП с последовательным интерфейсом

1. ЦАП с последовательным интерфейсом в библиотеке Proreus VSM

ЦАП расположены в подразделе Data Converters / D/A Converters, модели с последовательным интерфейсом можно найти по наличию слова Serial в графе Description (см. Табл. 7.5). Технические описания перечисленных в таблице микросхем находятся в папке DataSheets \ Serial-DAC-Proteus.

Большинство моделей ЦАП с ПИ в среде Proteus имеют трехпроводный интерфейс SPI (графа «ПИ, МГц»), что объясняется его высокой производительностью, следовательно, возможностью получать быстрые изменения выходного аналогового сигнала.

Число моделей внешнего ЦАП с двухпроводным интерфейсом I2C существенно меньше, так как он менее производительный. Тактовая частота в стандартном режиме 100 кГц, в «ускоренном» 400 кГц, требуется время для передачи адреса и ряда служебных битовых комбинаций.

Таблица 7.5. ЦАП с ПИ из библиотеки Proteus VSM.

Наимен.	ПИ, МГц	Разр.	Опора	Кан.	Выв.	Формат данных, биты от старш. к младш.
ad5310	SPI, 33	10	VCC	1	6	xxxx$d_9d_8d_7d_6$ $d_5d_4d_3d_2d_1d_0$xx
ad5320	SPI, 33	12	VCC	1	6	xxxx$d_{11}d_{10}d_9d_8$ $d_7d_6d_5d_4d_3d_2d_1d_0$
ad5601	SPI, 30	8	VCC	1	6	xx$d_7d_6d_5d_4d_3d_2$ $d_1d_0$xxxxxx
ad5611	SPI, 30	10	VCC	1	6	xx$d_9d_8d_7d_6d_5d_4$ $d_3d_2d_1d_0$xxxx
ad5621	SPI, 30	12	VCC	1	6	xx$d_{11}d_{10}d_9d_8d_7d_6$ $d_5d_4d_3d_2d_1d_0$xx
Max504	SPI, 10	10	VCC	1	14	xxxx$d_9d_8d_7d_6$ $d_5d_4d_3d_2d_1d_0$xx
Max515	SPI, 10	10	2.048, ±5	1	8	xxxx$d_9d_8d_7d_6$ $d_5d_4d_3d_2d_1d_0$xx
Max5258/9	SPI, 10	8	Внеш.<5	8	16	xx$c_2c_1c_0$110 $d_7d_6d_5d_4d_3d_2d_1d_0$
Mcp4821/2	SPI, 20	12	2.048*2	½	8	$c_0$0g1$d_{11}d_{10}d_9d_8$ $d_7d_6d_5d_4d_3d_2d_1d_0$**
Mcp4725	I2C, 3.4	12	VCC	1	6	110000$a_0$0 0000$d_{11}d_{10}d_9d_8$ d_7…d_0
Max5820 Max5821 Max5822	I2C, 0.4	8 10 12	Внеш.<5	2	8	011100$a_0$0 000$c_0d_7d_6d_5d_4$ d_3…$d_0$xxxx* 011100$a_0$0 000$c_0d_9d_8d_7d_6$ d_5…$d_0$xx* 011100$a_0$0 000$c_0d_{11}d_{10}d_9d_8$ d_7…d_0* 01110000 11110000 00001100
Max517 Max518 Max519	I2C, 3.4	8	Внеш.<5 VCC Внеш.<5	1 2 2	8 8 16	01011$a_1a_0$0 000000xxx d_7…d_0 01011$a_1a_0$0 000000xxc_0 d_7…d_0 010$a_3a_2a_1a_0$0 000000xxc_0 d_7…d_0

*Требуется инициализация для выхода из режима PowerDown, см. строку ниже

** Бит g управляет усилением. При VCC = 5 В выбирайте g = 0, Vout = D * 2.048*2 / 4096.

Разрядность (графа «Разр.») варьируется от 8 до 12 бит, диапазон выходного напряжения у всех однополярный, верхний предел (графа «Опора») у большинства задан напряжением питания VCC = 5 В, у других не превышает 5 В. Ряд микросхем объединяют несколько ЦАПов в одном корпусе (графа «Кан.»), в этом случае используется аппаратная или программная адресация канала. В графе «Выв.» указано число ног микросхемы, наименование выводов видно на графическам изображении. В графе «Формат данных, биты от старшего к младшему» указан формат кадра в виде последовательности двух или трех байт (в двоичном формате) для записи нового значения выбранного канала ЦАП, обозначения: x – не значащие биты (0), d_x – данные, a_x – аппаратный адрес микросхемы ЦАП, c_x – программный адрес канала в многоканальных ЦАП.

2. ЦАП с SPI (3-Wire) интерфейсом и их использование

Контроллер SPI в МК AVR использует выводы SCK (выход тактов) и MOSI (выход данных),

вывод MISO (вход данных) при работе с внешним ЦАП обычно не используется. Для формирования сигнала выборки микросхемы ЦАП (активный уровень – 0) используется один или несколько выводов портов в режиме выхода.

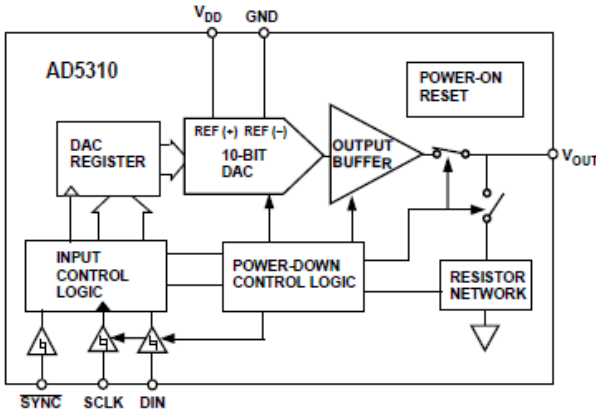
Наименование выводов микросхем ЦАП с интерфейсом SPI отличается разнообразием. Как правило, вход тактов называется CLK или SCK или SCLK, вход данных – DI или DIN или SDIN, выход данных (если имеется) – DO или SDO, вход выборки кристалла – CS или SYNC (со знаком инверсии) или LATCH. Аналоговый выход носит название OUT или VOUT (если выходов несколько, они нумеруются), вход опорного напряжения – REF или REFIN, при его отсутствии в качестве опорного напряжения используется напряжение питания (обычно +5 В).

Вход выборки кристалла служит не только для адресации конкретного ЦАП, но и позволяет ведомому вести счет принимаемых байтов. При знакомстве с конкретной микросхемой ЦАП следует обращать внимание на требования к формату кадра для записи данных в регистр ЦАПа.

Для подключения двух и более микросхем ЦАП к одному МК вход выборки кристалла каждой надо подключить к отдельному выходу МК и программно формировать срез в начале посылки и фронт в конце для адресации. Если микросхема содержит более одного канала ЦАП, сама посылка (кадр) содержит биты для адресации.

Для всех прведенных в табл. 7.5 микросхем ЦАП кадр для SPI интерфейса состоит из двух однобайтных посылок.

На рис. 7.13 приведены выдержки из DataSheets на AD5310, ниже дан подстрочный перевод и пояснения.



Single 10-Bit DAC
6-Lead SOT-23 and 8-Lead microSOIC Packages
Micropower Operation: 140 μA @ 5 V
Power-Down to 200 nA @ 5 V, 50 nA @ 3 V
+2.7 V to +5.5 V Power Supply
Guaranteed Monotonic by Design
Reference Derived from Power Supply
Power-On-Reset to Zero Volts
Three Power-Down Functions
Low Power Serial Interface with Schmitt-Triggered Inputs
On-Chip Output Buffer Amplifier, Rail-to-Rail Operation
SYNC Interrupt Facility

PIN FUNCTION DESCRIPTIONS

SOT-23 Pin Numbers

Pin No.	Mnemonic	Function
1	V_{OUT}	Analog output voltage from DAC. The output amplifier has rail-to-rail operation.
2	GND	Ground reference point for all circuitry on the part.
3	V_{DD}	Power Supply Input. These parts can be operated from +2.5 V to +5.5 V and should be decoupled to GND.
4	DIN	Serial Data Input. This device has a 16-bit shift register. Data is clocked into the register on the falling edge of the serial clock input.
5	SCLK	Serial Clock Input. Data is clocked into the input shift register on the falling edge of the serial clock input. Data can be transferred at rates up to 30 MHz.
6	$\overline{SYNC}$	Level triggered control input (active low). This is the frame synchronization signal for the input data. When $\overline{SYNC}$ goes low, it enables the input shift register and data is transferred in on the falling edges of the following clocks. The DAC is updated following the 16th clock cycle unless $\overline{SYNC}$ is taken high before this edge in which case the rising edge of $\overline{SYNC}$ acts as an interrupt and the write signal is ignored by the DAC.

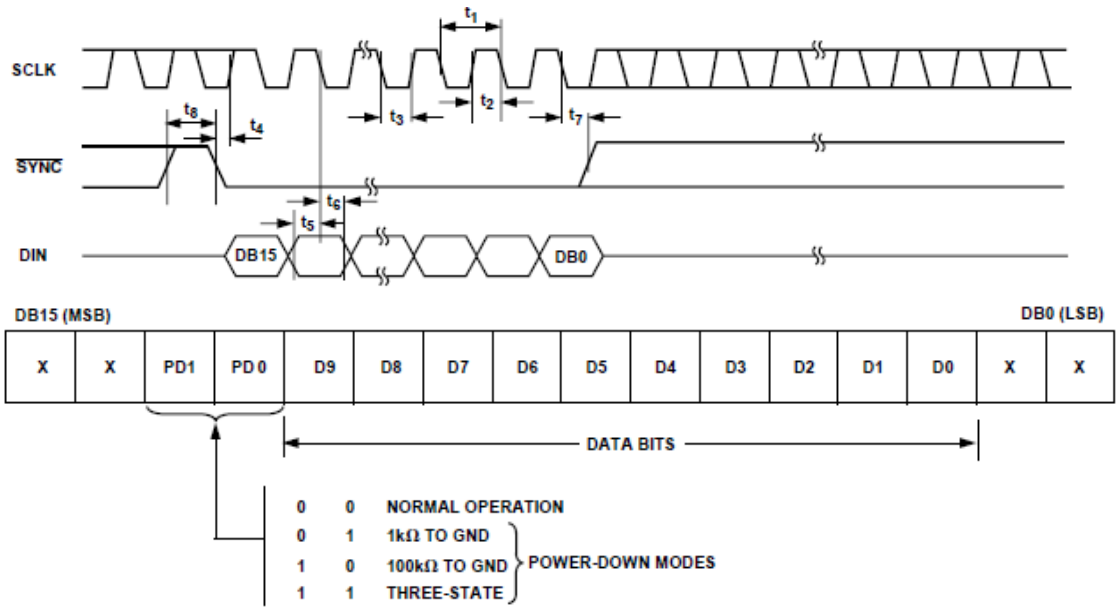


Рис. 7.13. ЦАП AD5310

Структурная схема показывает связи между внутренними блоками и выводами микросхемы. Блок Input control logic через входы SYNC, SCLK, DIN получает данные по интерфейсу SPI и передает их в регистр данных ЦАП [DAC register] и в блок управления режимом пониженного энергопотребления [Power down control logic]. Собственно ЦАП [10 bit DAC] преобразует полученный код в напряжение, используя как опору питание микросхемы [Ref+ = Vdd, Ref- = GND], далее это напряжение проходит через буферный усилитель [Output buffer] и замкнутый ключ на выход Vout. Особенности работы в режиме пониженного энергопотребления рассматривать не будем.

Из раздела кратких технических характеристик важно отметить:

- «Single 10-bit DAC» - одноканальный 10-битный ЦАП,
- «+2.7 V to +5.5 V» - диапазон напряжения питания (и опоры),
- «Reference derived from power supply» - опорное напряжение из напряжения питания,
- «On-chip output buffer amplifier, rail-to-rail operation» -- встроенный буферный усилитель с выходом, ограниченным только питанием.

Назначение выводов:

1 Vout Аналоговый выход ЦАП ограниченный только питанием

2 GND Общий вывод всех цепей

3 Vdd Вход питания с диапазоном от +2.5 до +5.5 В по отношению к выводу GND (аналогично VCC)

4 DIN Последовательный вход данных в 16-битный сдвиговый регистр, тактируемый срезом тактов

5 SCLK Последовательный вход тактов с частотой до 30 МГц

6 #SYNC Вход управляемый низким уровнем, синхронизирует цикл приема длительностью 16 тактов

Операция последовательной записи (определяет логику сигналов и обозначения временных параметров).

Назначение бит 16-битного входного регистра:

- 15, 14 – не имеют значения,
- 13, 12 (PD1, PD0) – управление режимами пониженного энергопотребления (для нашего случая требуются нулевые значения),
- 11…2 – разряды 9…0 ЦАП,
- 1, 0 – не имеют значения.

Для записи 10-разрядного числа unsigned int val в ЦАП AD5310 потребуется разбить его на два байта – старший (передается первым) (val >> 6) & 0x03, младший байт (передается вторым) (val << 2) & 0xfc.

В модели Proteus VSM в качестве питания используется напряжение +5 В, поэтому Vout = val * 5 / 1024 [В].

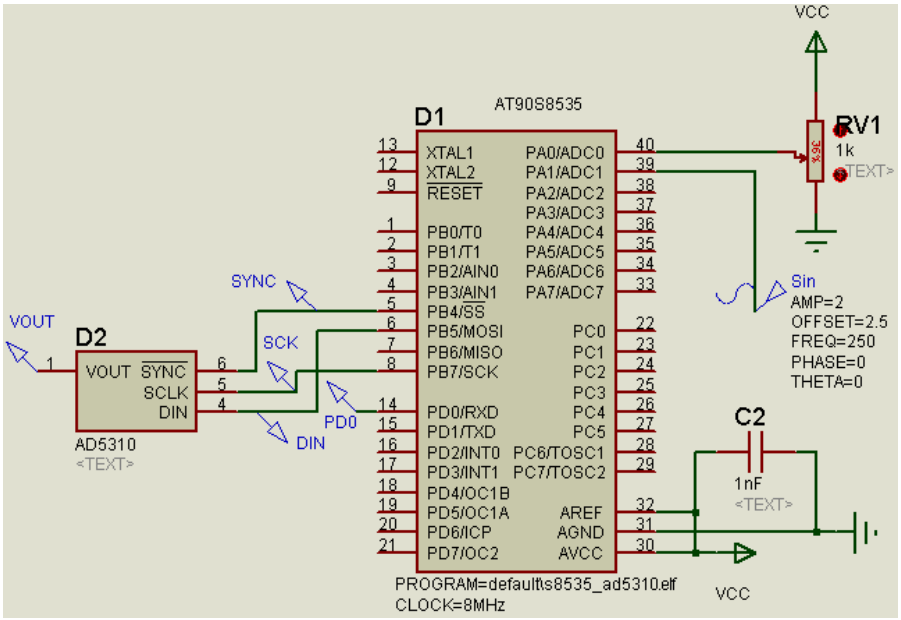
Для подключения к МК единственного внешнего ЦАП AD5310 выходы SPI порта SCK, MOSI подключим ко входам SCLK, DIN, вход #SYNC – к любому выходу МК (совсем не обязательно к #SS, так

как это название входа). Для подключения более чем одного ЦАП AD5310 цепи SCLK, DIN добавляются в параллель одноименным, цепи выборки #SYNC – каждая к своему выходу МК, разделение времени по выборкам позволяет разделить вывод в тот или другой ЦАП.

Пример оцифровки и восстановления синусоиды с ЦАП со встроенным SPI интерфейсом

В папке p4 Serial Interface \ ADC_DAC-SPI рассматривается пример с МК at90s8535 и 10-разрядным ЦАП AD5310. Идея примера состоит в том, что на вход встроенного в МК АЦП подается аналоговый сигнал в виде синусоиды, внешний ЦАП «восстанавливает» аналоговый сигнал, при этом видна роль задержки двойного преобразования. Модель ЦАП нечувствительна к изменению полярности (CPOL) и фазы (CPHA) тактирующих импульсов.

На рис.7.14 представлены схема (из файла <s8535_ad5310.dsn>), элементы текста программы (из <s8535_ad5310.c>), временные диаграммы передачи данных по SPI и сравнения измеряемого и восстановленного сигналов (из окна Transient Analysis (Mixed) файла <s8535_ad5310.dsn>).

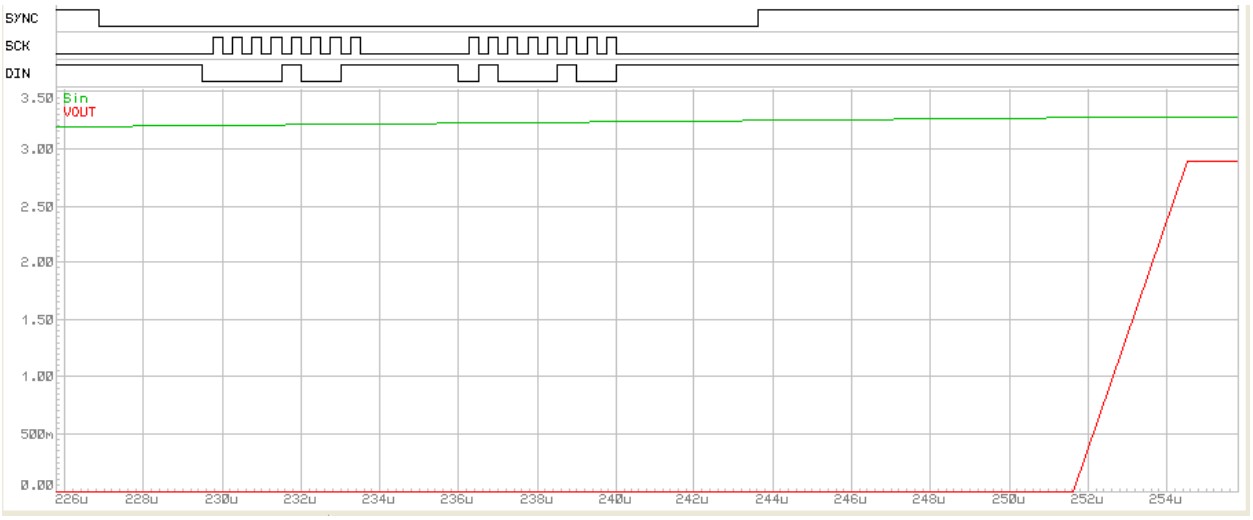


а. Схема

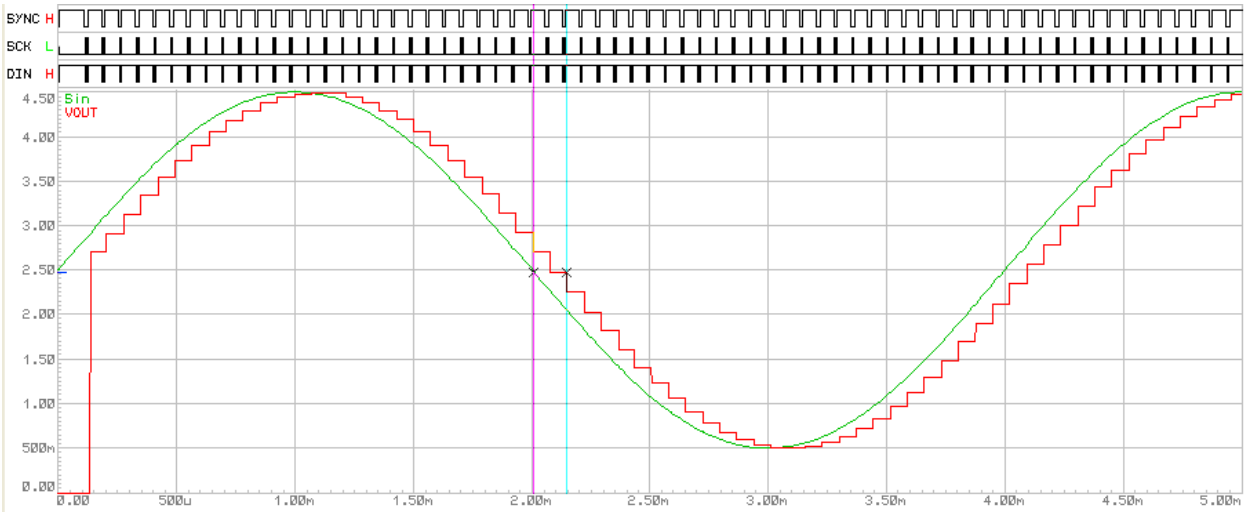
```
#define CS1 PB4
#define CS2 PB3
void put_DAC_ad5310(unsigned int value1, char ch) { //ts=0.75us
  unsigned char bbyte;
  if(ch==0) PORTB &= ~(1<<CS1); else PORTB &= ~(1<<CS2);//Начало  кадра
  //ts=1.5us
  SPDR = value1>>6; //В первом байте передаются 4 чтарших бита, ts=5.63 us
  while(!(SPSR &(1<<SPIF)));//0.5мкс*8=4мкс, ts=10.0us
  bbyte = SPDR;//Этот байт не используется, но требуется чтение! ts=10.13us

  SPDR = value1<<2;//Во втором байте передаются 6 младших бит, ts=10.63us
  while(!(SPSR &(1<<SPIF)));//ts=15.0us
  bbyte = SPDR;// Этот байт не используется, но требуется чтение! ts=15.13us

  if(ch==0) PORTB |= (1<<CS1); else PORTB |= (1<<CS2);//Завершение кадра
} //************************

int main() {
  static unsigned int val;
  DDRB = 0xff;   DDRC = 0xff; DDRD = 0xff;
  PORTB = (1<<CS2)|(1<<CS1);//Выходы выборки ЦАП
  ADMUX=0;// channel=0
```

```
  ADCSR=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS0);//ВклАЦП, fadc=fclk/32=250кГц
  SPCR=(1<<SPE)|(1<<MSTR);//|(1<<CPOL)|(1<<CPHA);//Вкл.SPI, Ведущий
  SPSR=0;// fclk/4=2МГц

  while (1) { //70.88 us
    val = read_adc(1); //Чтение аналогового входа, 54.5 us
    put_DAC_ad5310(val, 0); //Запись в аналоговый выход, 16.38 us
  };
} //**************************
```

б. Текст программы (выдержки)



в. Пересылка данных (0b1001010001 = 593) и изменение выхода (U = 5*593/1024 = 2.895 В)



г. Входной и восстановленный сигналы

Рис. 7.14. Пример ADC_DAC-SPI

В первом варианте примера одновременно работали два канала оцифровки и восстановления. Сейчас от второго канала на схеме остался потенциометр RV1 на входе ADC0, в программе – функция put_DAC_ad5310 имеет второй параметр для выборки кристалла CS1 или CS2.

При тактовой частоте МК fclk = 8МГц максимальная частота тактирования SPI для данного МК в 4 раза ниже fspi = fclk/4 = 8/4 = 2 МГц, поэтому для восьми тактов передачи одного байта требуется 8 / fspi = 4 мкс. По временной диаграмме (рис.7.14, в) время выборки ad5310 tsync = 16.7 мкс, задержка обновления выхода ЦАП 8.2 мкс, время преобразования АЦП tadc = 56 мкс, период основного цикла Тоц = 72 мкс примерно равен сумме tsync + tadc = 16.7 + 56 = 72.7. В данной связке более «ленив» АЦП.

Задержка восстановленного сигнала не зависит от формы входного сигнала и составляет

(рис. 7.14, г) от 72 до 144 мкс, то есть 1-2 периода Тоц. При частоте входного сигнала 250 Гц (период 4 мс) форма восстановленного сигнала весьма грубо повторяет входной, относительная задержка 2.5 %, при снижении частоты на порядок все значительно улучшится.

3. ЦАП с I2C (2-Wire) интерфейсом и их использование

Именование выводов: выводы для I2C интерфейса всегда используют стандартные имена (SCL, SDA), имена выводов питания и аналоговых выходов не отличаются от других ЦАП, еще есть вход/ы задания адреса A0 или ADD или AD0, AD1, уровнеь на этих входах формирует младшие биты 7-битного адреса, это позволяет использовать на одной плате несколько однотипных ЦАП.

Для всех прведенных в табл. 7.5 микросхем ЦАП кадр для I2C интерфейса состоит из трех однобайтных посылок.

MCP4725 (рис. 7.15) имеет один канал ЦАП разрядностью 12 бит, встроенная энергонезависимая ячейка EEPROM гарантирует требуемый уровень напряжения сразу после подачи питания, вход A0 позволяет адресовать от одного МК две миросхемы ЦАП. Для нашего применения удобно использовать временную диаграмму загрузки числа в регистр ЦАП (Fast Mode на рис. 7.11). Биты PD1, PD0 обнулить (другие значения нужны в режиме энергосбережения), биты A2, A1 тоже обнулить, бит A0 должен совпадать с уровнем на входе A0. Модель в Proteus имеет параметры Data Write Time (время записи в EEPROM) и Address Option (выбор одного изх трех бит A2-A0 для аппаратной адресации), эти параметры не нуждаются в подстройке.

Три подобных двухканальных модели ЦАП max5820, max5821, max5822 различаются разрядностью (8, 10, 12 бит), имеют одниковое назначение и расположение выводов, похожую логику работы. Напряжение на входе REF определяет диапазон на выходах OUTA, OUTB и не должно превышать напряжение питания VDD (=VCC). Каждая из микросхем выпускается в двух модификациях, отличающихся значением старших разрядов адреса (L: 0b011100X, M: 0b101100X, здесь X – значение на входе аппаратной адресации ADD), есть настройка в модели Proteus. Для начала работы этой микросхемы необходимо вывести ее из режима пониженного энергопотребления (write to power-down register sequence).

В некоторых приложениях важно получить одновременное изменение напряжения на выходах двух каналов, для этого есть два дополнительных регистра (Input Register A и B) и команды (посылки) записи сначала в A, затем в B, затем одновременная загрузки в регистры ЦАПов. Для наших задач такой алгоритм не требуется, поэтому используется команда (посылка) для непосредственной записи данных в регистр выбранного ЦАПа (см. Табл. 7.), бит a выбирает канал.

8-разрядные ЦАП max517/8/9 имеют один/два выхода OUT0, OUT1, от 2 до 4 входов аппаратной адресации микросхемы AD0…AD3, от 0 до 2 входов опорного напряжения REF:

Max517 – 8 ног, 1 канал, 1 вход опоры, 2 входа адресации,

Max518 – 8 ног, 2 канала, нет входа опоры (Vref = VDD), 2 входа адресации,

Max517 – 16 ног, 2 канала, 2 входа опоры, 4 входа адресации.

Модели в Proteus параметров настройки не имеют.

Пример оцифровки и восстановления синусоиды с ЦАП со встроенным I2C интерфейсом

В папке ADC_DAC- I2C находится пример с МК atmega16 и ЦАП с интерфейсом I2C max5821. Проект m16_max5128.aps состоит из двух файлов – <m16_max5128.c> (функции main(), put_DAC_max5821() и знакомая read_adc()) и < i2c.c> (функции, составляющие put_DAC_max5821()), последний подключает файл описания констант < i2c.h>, включая выбор частоты тактирования. На рис. 7.16 приведены выдержки из этих файлов.

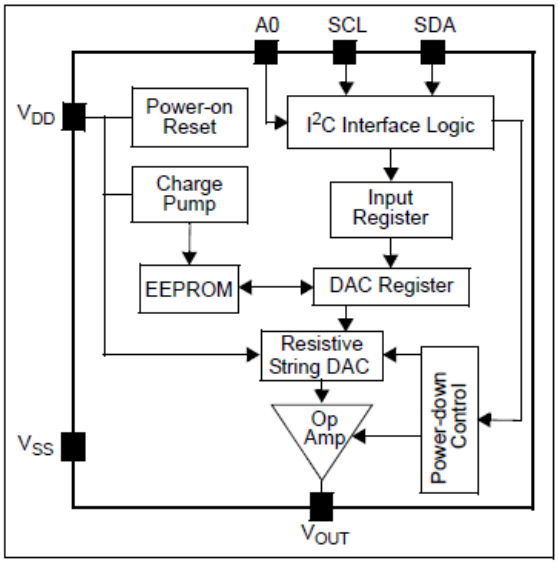
При частоте тактирования fscl = 400 кГц (максимальная рекомендуемая для данного МК)

цикл передачи команды настройки или данных для одного канала ЦАП по временной диаграмме составляет 86 мкс, период основного цикла – примерно 195 мкс, задержка восстановленного сигнала по отношению к измеряемому составляет 86 / 283 мкс.

0b11 0001 0101 = 810, 810*5/1024 = 3.955 В, но OUTA = 3.85 В.

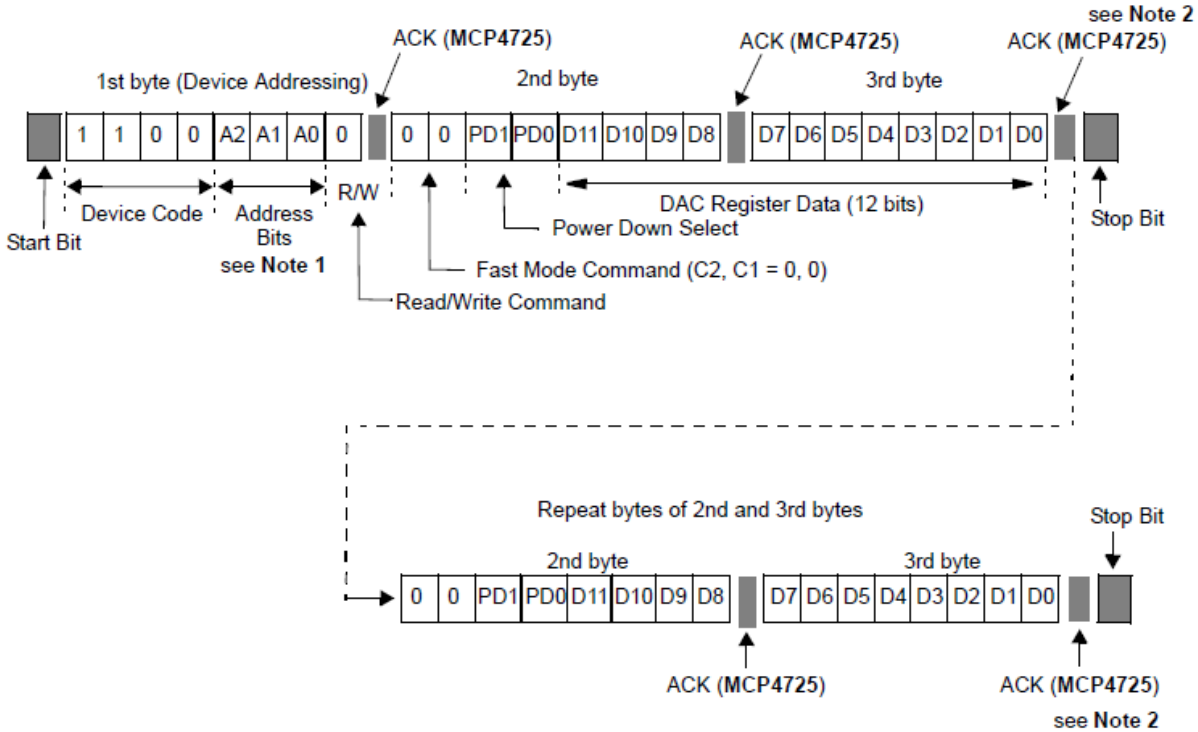
ti2c = 88.4us, tadc = 108us

Смущает минимальная задержка 86 мкс, ведь она должна состоять из tadc + ti2c = 108 + 88 = 196 мкс, вот максимальная похожа на правду tadc + 2*ti2c = 108 + 177 = 285.



- 12-Bit Resolution
- On-Board Non-Volatile Memory (EEPROM)
- ±0.2 LSB DNL (typical)
- External A0 Address Pin
- Normal or Power-Down Mode
- Fast Settling Time of 6 µs (typical)
- External Voltage Reference (V_{DD})
- Rail-to-Rail Output
- Low Power Consumption
- Single-Supply Operation: 2.7V to 5.5V
- I^2C^{TM} Interface:
  - Eight Available Addresses
  - Standard (100 kbps), Fast (400 kbps), and High-Speed (3.4 Mbps) Modes
- Small 6-lead SOT-23 Package
- Extended Temperature Range: -40°C to +125°C

Change DAC Code in Fast Mode: (C2,C1) = (0,0)

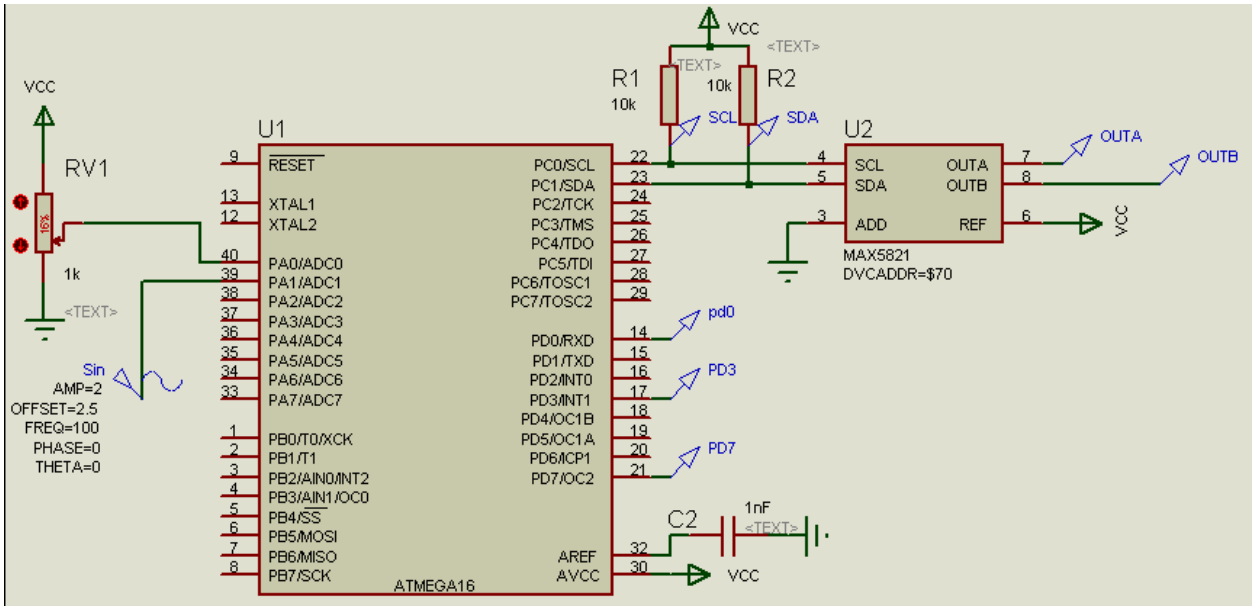


Note 1: A2 and A1 bits are programmed at the factory by hard-wired, and A0 bit is determined by the logic state of A0 pin.

2: The device updates V_{OUT} at the falling edge of the ACK pulse of the 3rd byte.

FIGURE 6-1: *Write Command for Fast Mode.*

Рис. 7.15. ЦАП с двухпроводным интерфейсом mcp4725



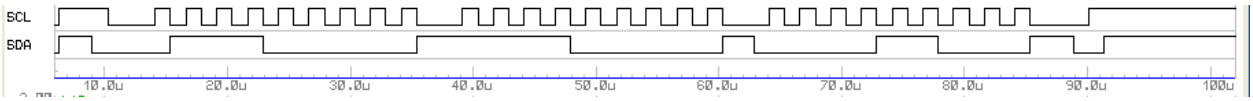
а. Схема

```
#define SLA_MAX5821L 0b01110000 /* 10 бит, 2 канала */
void put_DAC_max5821(unsigned int value, char ch) {
    i2c_start();
    i2c_SLA_W(SLA_MAX5821L);
    i2c_write((value>>6)|(ch<<4));//4ст.бита и выбор канала
    i2c_write(value<<2);//6мл.бит
    i2c_stop();
} //************************

int main() {
  static unsigned int val;
  DDRB = 0xff; DDRC = 0xff; DDRD = 0xff;
  ADMUX=(1<<REFS0);//AVCC+c(AREF), channel=0
  ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1);//fadc=fclk/64=125кГц, 8*14=112
  init_i2c_Master();
    i2c_start();
    i2c_SLA_W(SLA_MAX5821L);
    i2c_write(0xf0);//Команда выхода из
    i2c_write(0x0c);//Power-Down Mode
    i2c_stop();
  while (1) {
    val = read_adc(1); //>=112us
    put_DAC_max5821(val, 0);//=87us
  };
} //***************************
<i2c.c>
#include <avr/io.h>
#include "i2c.h"

void error_i2c(void) { PORTC |= (1<<6); }
//-------------------------------------------------------
void init_i2c_Master(void) {// 2 Wire Bus initialization
  //PORTC |= ((1<<0)|(1<<1));//Pull-Up
  TWCR=(1<<TWEN);//TWEA- Enable ACKnowledge
  TWBR=TWBR_SET;
```

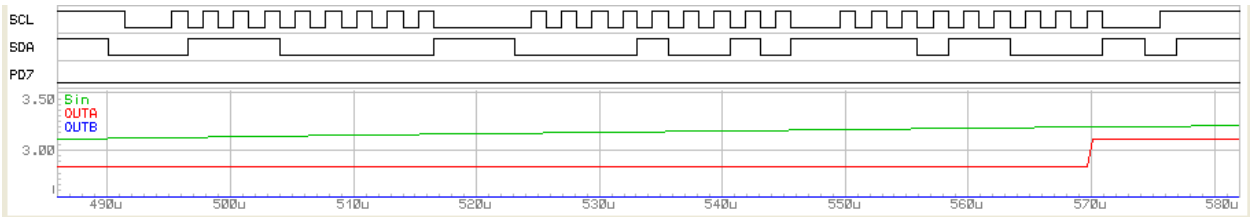
```
}//-----------------------------------------------------
void i2c_start(void) {
  TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN);  //Send START condition
  while (!(TWCR & (1<<TWINT))) ;//Wait for TWINT flag set
  if((TWSR & 0xF8) != START) error_i2c();
}//-----------------------------------------------------
void i2c_stop(void) {
  TWCR=(1<<TWINT)|(1<<TWSTO)|(1<<TWEN);  //Send STOP condition
}//-----------------------------------------------------
void i2c_SLA_W(char byte) { //MT
  TWDR = (byte & ~R_W_bit);
  TWCR=(1<<TWEN)|(1<<TWINT);//Send SLA+W
  while (!(TWCR & (1<<TWINT))); //Wait for TWINT flag set.
  if((TWSR & 0xF8) != SLA_W_ACK) error_i2c();
}//-----------------------------------------------------
void i2c_SLA_R(char byte) { //
  TWDR = (byte | R_W_bit);
  TWCR=(1<<TWEN)|(1<<TWINT);//Send SLA+R
  while (!(TWCR & (1<<TWINT))); //Wait for TWINT flag set.
  if((TWSR & 0xF8) != SLA_R_ACK) error_i2c();
}//-----------------------------------------------------
void i2c_write(char byte) {
  TWDR = (byte);
  TWCR=(1<<TWEN)|(1<<TWINT);//Send byte
  while (!(TWCR & (1<<TWINT))) ;//Wait for TWINT flag set
  if((TWSR & 0xF8) != BYTE_W_ACK) error_i2c();
}//-----------------------------------------------------
unsigned char  i2c_read(char ack)  {
  TWCR =(1<<TWEN)|((ack)<<TWEA)|(1<<TWINT); //Read byte
  while (!(TWCR & (1<<TWINT))) ;//Wait for TWINT flag set
  return TWDR;
}//-----------------------------------------------------
<i2c.h>
#define Fclk  8000000L
#define FSCL  400000L
#define TWBR_SET    (Fclk/FSCL - 16)/2

// TWI Control Register – TWCR
// Bit 7 – TWINT: TWI Interrupt Flag
// Bit 6 – TWEA: TWI Enable Acknowledge Bit – разреш генерации ответа
// Bit 5 – TWSTA: TWI START Condition Bit
// Bit 4 – TWSTO: TWI STOP Condition Bit
// Bit 3 – TWWC: TWI Write Collision Flag
// Bit 2 – TWEN: TWI Enable Bit
// Bit 0 – TWIE: TWI Interrupt Enable

// Status Codes for Master Transmitter Mode
#define  START          0x08
#define  RESTART        0x10
#define  SLA_W_ACK      0x18
#define  SLA_W_NACK     0x20
#define  BYTE_W_ACK     0x28
#define  BYTE_W_NACK    0x30
```

```
#define  ARBITR_LOST    0x38
#define  SLA_R_ACK      0x40
#define  SLA_R_NACK     0x48
#define  BYTE_R_ACK     0x50
#define  BYTE_R_NACK    0x58

#define Ack 1
#define NoAck 0

#define  READ  0x01
#define  R_W_bit (1<<0)
```

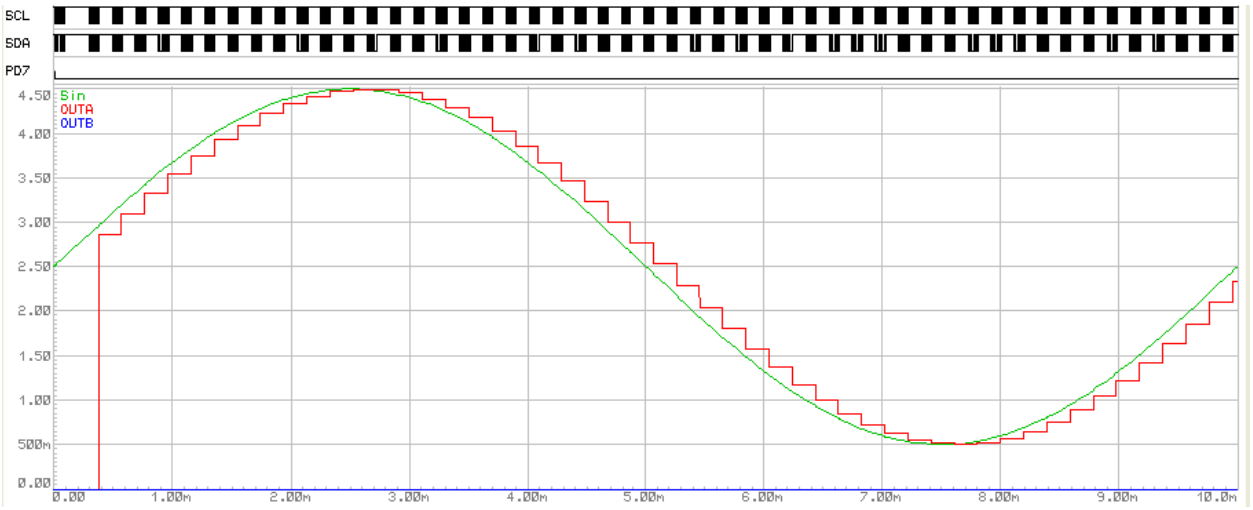
д. Тексты программ (выдержки)



б. Процедура настройки max5821



в. Передача данных первого канала (0b100111101100 = 2540, 2540*5/4096 = 3.10 В)



г. Аналоговый и восстановленный сигналы

Рис. 7.16. Пример ADC_DAC-I2C

7.6. Устройство контроллера U(S)ART и его применение

Асинхронные протоколы применяются в МК давно и широко: связь МК с персональным компьютером (COM-порт), с программатром/отладчиком, построение распределенных систем управления. В большинстве случаев требуемая дальность связи начинается от единиц до сотен метров, что заставляет учитывать ослабление полезного сигнала и влияние помех.

Большинство современных МК имеют в своем составе один или несколько модулей *UART [Universal Asynchronous serial Receiver and Transmitter]*. Асинхронные протоколы не передают тактирующий сигнал, синхронизация передатчика и приемника выполняется за

счет выбора одинаковой частоты тактирования и передачи в потоке данных дополнительных синхроимпульсов. Наличие двух линий передачи данных: *RX*(*D*) – вход приемника [*receiver*], *TX*(*D*) – выход передатчика [*transmitter*], позволяет при использовании контроллера *UART* вести передачу и прием одновременно, это называется дуплексным режимом обмена.

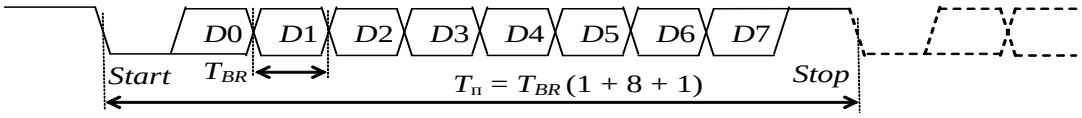


Рис. 7.17

Формат посылки протокола и контроллера *UART* допускает вариации, основные элементы: нулевой старт-бит для синхронизации генератора тактов приемника, от 5 до 9 битов данных (младший всегда первый), необязательный бит паритета (четности или нечетности), 1–2 стоп-бита «единичного» уровня для контроля синхронизации в конце посылки. На рис. 7.17 приведен наиболее распространенный вариант формата 8*N*1 с 8 битами данных, без бита паритета (*N*) и одним стоп-битом. Необходимым условием является выбор одинаковой частоты работы тактовых генераторов передатчика и приемника из стандартного фиксированного ряда (50, 75, …, 9600, 19200, 38400, 57600, 115200 бод для COM-порта), отклонение частоты приемника и передатчика не должно превышать 0.5…2 %.

Контроллеры UART / USART в AVR-8

Контроллер USART поддерживает асинхронный и синхронный режимы. Синхронный режим мы рассматривать не будем, ограничимся функциями UART. В структуре контроллера (рис. 7.18) блоки и вывод, требуемые только для синхронного режима выделены серым цветом.

Основным яляется регистр данных UDR (UDRn, n = 0, 1 …, если модулей несколько) форматом один байт, запись в регистр запускает процесс передачи, чтение позволяет получить данные приемника.

После записи байта в регистр данных он копируется в сдвиговый регистр и начинается процесс передачи. Если требуется послать следующий байт, следует дождаться флага «передатчик пуст». Если в режиме полудуплекса после передачи надо перейти к приему, следует дождаться флага «завершение передачи».

Процесс приема состоит в ожидании установка флага «завершение приема» и последующем чтении регистра данных.

Регистры UCSR(n)A, UCSR(n)B , UCSR(n)C содержат биты управления и состояния (флаги). Прием и передача являются независимыми процессами, общим является только тактирование, число в регистре UBRR(n) задает частоту одинаковую для передатчика и приемника.

При нулевом значении UBRR выбирается максимальная частота тактирования, она составит fclk / 8 или fclk / 16, например, при fclk = 16 МГц получим 1 МГц. В этом случае минимальная по длительности посылка из 8 бит данных, одного старт-бита и одного стоп-бита займет 10 тактов, то есть 10 мкс. Реальные значения частоты тактирования обычно существенно ниже, особенно в промышленных условиях, длительность посылки возрастает. Поэтому для экономии времени процессора актуально использование прерываний – одно от приемника «завершение приема» и два от передатчика «буфер передатчика пуст» и «завершение передачи».

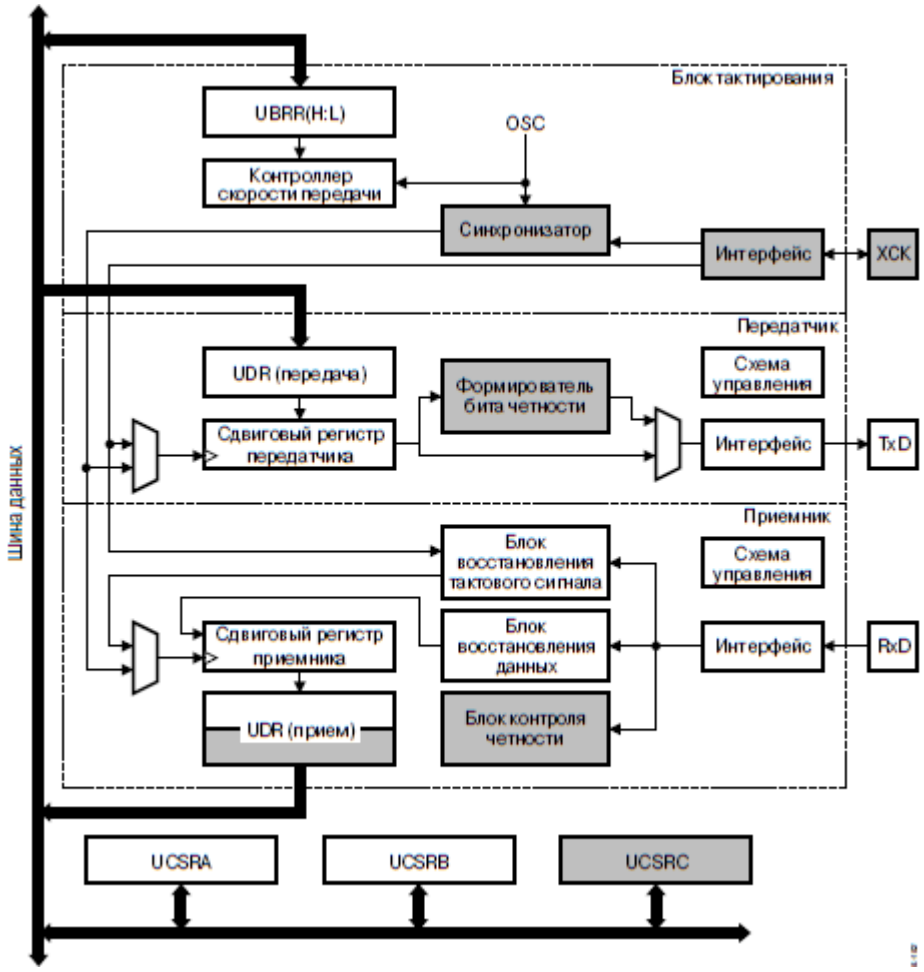


Рис. 7.18

Контроль ошибок в процессе предачи/преима на уровне контроллера U(S)ART:

- контроль четности: передатчик добавляет в посылку бит равный или противоположный сумме по модулю два (Исключающее ИЛИ) бит данных, приемник сравнивает полученный бит с суммой принятых бит данных, рассчитанной аналогично;

- переполнение приемника – программа не успела прочитать предыдущий байт данных,

- ошибка кадрирования приемника – нулевой уровень стоп-бита.

USART Control and Status Register A – UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

RXC [Receive Complete] – флаг звершения приема (запрос прерывания), устанавливается в 1 приемником при наличии непрочитанных данных в буфере приемника UDR, сбрасывается при чтении.

TXC [Transmit Complete] – флаг звершения передачи (запрос прерывания), устанавливается в 1 после освобождения сдвигового регистра передатчика при отсутствии очередного байта в буфере передатчика UDR; сбрасывается при входе в прерывание или программно записью 1.

UDRE [Date Register Empty] – флаг опустошения регистра данных (запрос прерывания), устанавливается в 1 после пересылки байта из регистра данных передатчика

UDR в сдвиговый регистр; сбрасывается аппаратно при записи в UDR.

FE [Frame Error] – флаг ошибки кадра (посылки), устанавливается приемником при нулевом уровне стоп-бита, сбрасывается при приеме стоп-бита с уровнем единица.

DOR [Data OverRun] – флаг переполнения приемника, устанавливается в 1, если при приеме новой посылки предыдущая не прочитана из UDR (потеря байта), сбрасывается при чтении UDR; при записи в UCSRA всегда обнулйте этот бит.

PE [Parity Error] – флаг ошибки контроля четности, устанавливается в 1 при обнаружении ошибки и разрешении контроля четности (UCSRC.UPM1 = 1).

U2X [Double Speed] – удвоение скорости обмена, в формуле выбора частоты тактирования: 1 – делитель равен 8, 0 – равен 16.

MPCM [Multy-Processor Communication Mode] – режим мультипроцессорного обмена, при 1 ведомый МК ожидает кадр (посылку) со своим адресом, остальные игнорируются.

USART Control and Status Register B – UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

RXCIE [RX Complete Interrupt Enable] – разрешение прерывания по завершению приема.

TXCIE [TX Complete Interrupt Enable] – разрешение прерывания по завершению передачи.

UDRIE [TX Data Register Empty Interrupt Enable] – разрешение прерывания по очистке регистра данных передатчика.

RXEN [Receiver Enable] – разрешение работы приемника, активирует вход RXD.

TXEN [Transmit Enable] – разрешение работы передатчика, активирует выход TXD.

UCSZ2 [Character Size] – размер посылки, в паре с битами UCSRC.UCSZ1 и UCSZ0 определяет число бит в посылке приема и передачи (значение по умолчанию 011 – 8 бит, значение 111 – 9 бит).

RXB8 [Receive Data Bit 8] – восьмой разряд принимаемых данных при 9-битном размере посылки, читать до чтения UDR.

TXB8 [Transmit Data Bit 8] – восьмой разряд передаваемых данных при 9-битном размере посылки, запись до записи UDR.

USART Control and Status Register C – UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

URSEL [Register SELect] – выбор регистра, 0 – обращение к UBRRH, 1 - к UCSRC.

UMSEL [Mode SELect] – выбор режима, 0 – асинхронный, 1 – синхронный.

UPM1:0 [Parity Mode] – выбор режима формирования и проверки четности (паритета), 00 – запрещено, 01 – резерв, 10 – разрешена четность, 11 – разрешена не четность.

USBS [Stop Bit Select] – количество стоп-бит при передаче, 0 – 1 бит, 1 – 2 бита.

USCZ1:0 [Character Size] – в объединении с UCSRB.USCZ2 определяет число бит в

посылке (см. выше).

UCPOL [Clock POLarity] – важен только в синхронном режиме.

USART Baud Rate Registers – UBRRL and UBRRH

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

URSEL – см. описание одноименного бита регистра UCSRC.

UBRR11:…:0 – 12-битное число, в паре с битом UCSRA.U2X определяющее значение коэффициента деления fclk для получения тактирования преемника и передатчика:

U2X = 0: BAUD = fclk / [16*(UBRR + 1)], UBRR = fclk / (16*BAUD) – 1,

U2X = 1: BAUD = fclk / [8*(UBRR + 1)], UBRR = fclk / (8*BAUD) – 1,

Здесь BAUD – значение частоты тактировваия [Бод] = [бит/с] = [Гц].

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

Максимальная ошибка, то есть отклонение значения частоты от стандартного значения, должна быть менее 1.5 – 2 %, лучше менее 0.5 % (для устойчивости к помехам). Пример выбора частоты интерфейса, частоты МК и значений для кодирования:

Ниже приведены примеры функций настройки, передачи и приема байта через UART, взятые из технического описания МК:

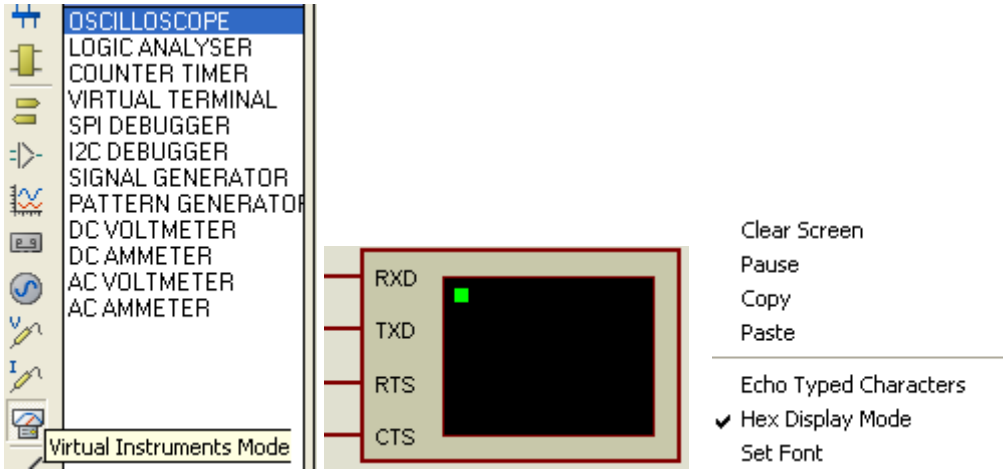
```
#define fclk 8000000L /* Частота тактирования процессора */
#define BAUD 9600 /* Скорость связи в бодах */
#define UBRR_VAL fclk/16/BAUD – 1 /* Вычисление кода задания скорости */

void init_USART( unsigned int ubrr) {//Настройка с заданием скорости
  UBRRH = (unsigned char)(ubrr>>8); //Загрузка кода скорости
```

```
  UBRRL = (unsigned char)ubrr;       // побайтно
  UCSRB = (1<<RXEN)|(1<<TXEN);/* Enable receiver and transmitter */
  UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);/* Set format: 8 data, 2stop bit */
} // ***************************
void transmit(char byte) { //Передача байта
  while(!(UCSRA & (1<<UDRE)));//Ожидание готовности буфера передатчика
  UDR = byte; //данные в буфер передатчика
} // ***************************
unsigned char receive(void) { //Прием байта
  while (!(UCSRA & (1<<RXC)));//Ожидание завершения приема
  return UDR; //чтение приемника
} // ***************************
```

Примеры использования UART в Proteus VSM

Примеры работы UART в системе моделирования Proteus VSM используют инструмент Virtual Terminal. Он позволяет в интерактивном режиме использовать экран и клавиатуру ПК для посылки и приема байт по асинхронному последовательному интерфейсу RS-232 из моделируемого МК.

Для выбора этого инструмента выберите в левом столбце пиктограмму Virtual Instrument Mode, окно буфера выбора теперь содержит список доступных инструментов (рис. 7.19, а), выберите и разместите в окне схемного редактора Virtual Terminal (рис.7.19 , б). Как и в МК вывод RXD является входом данных, его надо соединить с выходом TXD МК, вывод TXD – выходом данных, соединяется со входом RXD МК. Выводы RTS [ready-to-send] и CTS [clear-to-send] требуются для аппаратного квитирования [hardware handshake], в наших примерах соединены между собой. Основные настройки [properties] состоят из параметров частоты связи Baud Rate, длины посылки Data Bits, выбора проверки четности Parity, длины стопа Stop Bits, разрешения программного квитирования Send Xon/Xoff. Дополнительные настройки [advanced properties] позволяют выбрать полярность сигналов и задать настройки отладочных сообщений при трассировке.



а б в

Рис. 7.19

После запуска сеанса интерактивного моделирования появляется всплывающее окно Virtual Terminal (см. рис. 7.20, г, д), в это окно выводятся полученные по входу RXD данные в виде Hex-кодов или ASCII-символов, при вводе данных надо предварительно кликнуть в окне, затем нажимать клавиши клавиатуры. В окне действует контекстное

всплывающее меню (рис 7.19, в), оно позволяет выполнять стандартные функции с содержимым окна (очистка, пауза, копирование и вставка) и настраивать отображение (печать ввода с клавиатуры, Hex или ASCII, выбор шрифта).

Пример 7.3.1 «Терминал в режиме ASCII» находится в папке p4_Serial_Interface \ UART_VirtualTerminal-ASCII, общая схема примера (файл test_UART.dsn и рис. 7.20, а) состоит из МК и Virtual Terminal, потенциометра RV1 и генератора синусоидальных колебаний Sin на входах встроенного АЦП, фильтров нижних частот R1C1 и R2C2 на выходах сравнения.

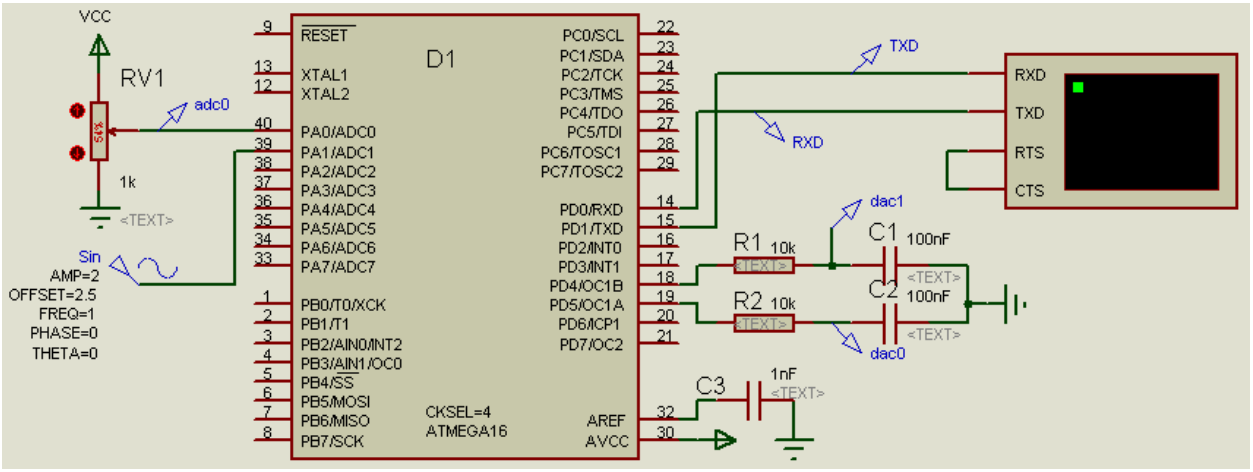
Настройки Virtual Terminal:

- настройки в режиме редактора схемы (Edit Properties): “Baud Rate” = 9600, “Data Bits” = 8, “Parity” = NONE, ”Stop Bits” = 1 должны соответствовать программным настройкам UART’a, Xon/Xoff в этой задаче не влияет, Advanced Properties – без изменения,

- настройки в режиме интерактивного моделирования (правой кнопкой мыши в окне Virtual Terminal): “Hex Display Modes”, “Echo Typed Characters” – без галочки.

Рассмотрим несколько примеров программ, по принципу «от простого к сложному».

А) «Нулевой цикл» для знакомства с выводом данных через UART на терминал (режимы Hex и ASCII). В схеме используются только МК и Virtual Terminal, проект программы <VTo_m16 \ VTo_m16.aps> (выдержки на рис. 7.20, б) выполняет периодический вывод на терминал нарастающего числа (i = 0, 1, … 255, 0, 1, …) с периодом 1 секунда.

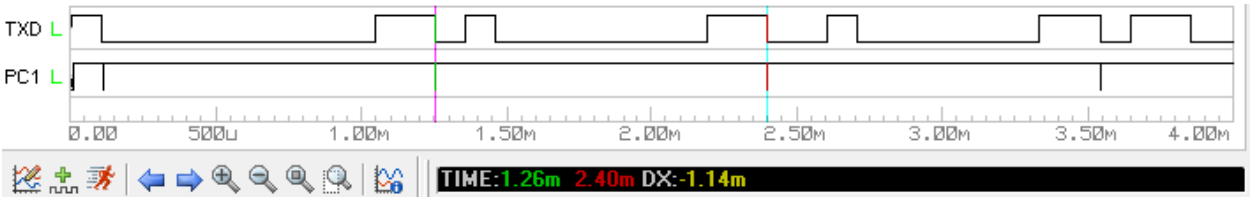


а

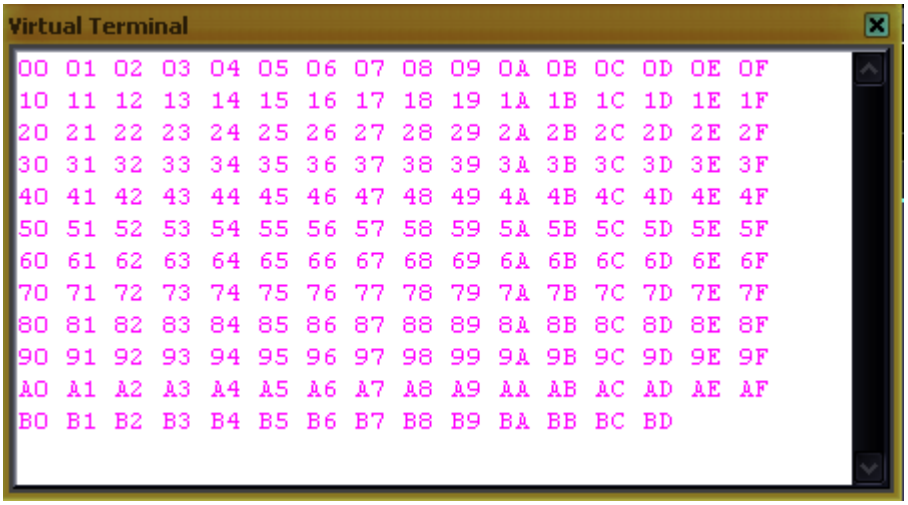
```
…
#define BAUD 9600
#define UBRR_VAL fclk/16/BAUD - 1
…
int main() {
  static unsigned char i=0;
  DDRC = 0xff;
  init_USART(UBRR_VAL);

  while (1) {
    PORTC ^= (1<<1);//Инверсия PC1
    transmit(i++);// Посылка байта на TXD (индикация на терминале)
    PORTC ^= (1<<1);//Инверсия PC1
    _delay_ms(1000);//Комментировать в режиме временной диаграммы
```

```
  }
} // ******************************
```

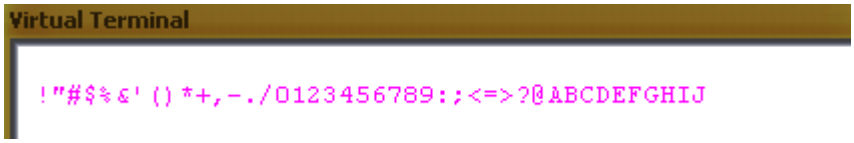
б



в



г



д

Рис. 7.20

Сначала рассчитаем и проверим временную диаграмму сигнала на выходе TXD. При тактировании BAUD = 9600, дительности посылки 1 + 8 + 2 = 11 тактов время одной посылки должно составить T = 11/9600 = 1.14 мс. «Закомментируем» в программе строку с задержкой 1000 мс, в этом случае период основного цикла будет практически равен длительности формирования посылки с одним байтом данных – функция transmit(i++).

После перекомпиляции в проекте <VTo_m16.aps> перейдите в <test_UART.dsn>, откройте свойства МК D1, выберите в качестве загрузочного модуля < VTo_m16.elf> и запустите расчет в окне Digital Analysis графиков на выходах TXD и PC1 на интервале 4 мс. На рис.7.20, в приведена временная диаграмма первых трех периодов основного цикла – это хорошо видно по сигналу на выводе PC1. С помощью двух курсоров измерена длительность периода – «1.14m» - что совпадает с расчетом. Легко разглядеть и передаваемые данные – числа 0, 1, 2.

Теперь, собственно, познакомимся с Virtual Terminal. Для этого «раскомментируем» в программе строку с задержкой 1000 мс, выполним компиляцию, в <test_UART.dsn> запустим интерактивный режим моделирования. Появится всплывающее окно Virtual Terminal, в которое выводятся данные (числа 0, 1, … 255, 0, 1, …) в шестнадцатиричном формате (рис. 7.20, г) или в формате ASCII (рис. 7.20, д), переключение медлу ними «галкой» возле строки Hex Display Modes во всплывающем меню. Формат ASCII (см.

Приложение X) предназначен для вывода символов, прежде всего, латинского алфавита A-Z, a-z, цифр 0-9 и других печатных симолов на дисплей или печатающее устройство. Обратите внимание, что не всем числам в коде ASCII соотвтетствуют видимые символы …

Б) Небуферированный ввод/вывод UART - Virtual Terminal, программа в папке и однименном проекте <VTio_m16>. Основной цикл

```
  while (1) {
    i = receive();//Прием байта с RXD (введенного с клавиатуры терминала)
    transmit(i);//Посылка байта на TXD (индикация на терминале)
  }
```

состоит из чтения байта с приемника и посылки его же на передатчик, то есть выполнение эхопечати в интерактивном режиме.

Для тестирования программы в свойствах МК <test_UART.dsn> замените файл Programm file на <VTio_m16.elf> и запустите интерактивный режим. Вновь появится окно Virtual Terminal, при нажатии клавиш на клавиатуре ПК их коды отображаются в окне в Hex (числом с пробелом) или ASCII (видимые символы или выполнение некоторых действий редактирования).

Функция receive() ждет поступления байта в приемник UART, это увеличивает период основного цикла на трудно предсказуемое время. Для управляющих программ, в основном цикле которых выполняются и другие задачи это резко увеличивает максимальное время реакции. (Хотя вполне подходит для программ «общения» с оператором.)

Использование прерывания приемника решает проблему зависания основного цикла:

```
ISR(USART_RXC_vect) {//Обработчик прерывания приемника
  in_byte = UDR;//Чтение премника
  transmit(in_byte);//и его запись в передатчик
}
```

Недостатком данного варианта можно считать значительное время выполнения прерывания приемника за счет функции transmit(). Вариант с использованием двух прерываний (приемника и передатчика) рассмотрен в папке и проекте VT_ISRio_m16.

```
ISR(USART_UDRE_vect) {//Обработчик прерывания передатчика
  UDR = in_byte;//Запись в передатчик и сброс флага UDRE
  UCSRB &= ~(1<<UDRIE);//Запрет прерывания передатчика
} // ****************************

ISR(USART_RXC_vect) {//Обработчик прерывания приемника
  in_byte = UDR;//Чтение премника
  UCSRB |= (1<<UDRIE);//Разрешение прерывания передатчика
} // ****************************
```

В) Иллюстрация простого протокола, работающего по принципу «Запрос» - «Ответ» использует все элементы схемы <test_UART.dsn>. Проект и программа в папке <VT_protocol_m16>.

Входы АЦП ADC0, 1 настроены на опору AVCC с программным запуском и чтением результата, выходы сравнения OC1A, B настроены на генерацию импульсов быстрой 8-разрядной ШИМ (f = 8 МГц / 256 = 31.25 кГц = 1 / (32мкс)). Последним действиеа выполняется вывод символа `'>'` , означающее приглашение ко вводу.

В основном цикле выполняется обработка протокола, предполагается использование Virtual Terminal в режиме отображения ASCII.

Протокол выполняет чтение байта с приемника, вывод на передатчик (для индикации), запись в буфер и продвижение указателя буфера на глубину до трех байт.

При получении кода клавиши <Enter> обнуляется указатель буфера и выполняется разбор содержимого буфера. Если получена последовательность ‘R’ и ‘0’ или ‘1’, то печатается значение со входа АЦП “Un=XXXX ” (n – 0 или 1, XXXX – десятичное число из четырех цифр), если получена последовательность ‘T’ и ‘0’ или ‘1’, то значение АЦП копируется на ЦАП (цифра задает номер канала), в остальных случаях на печать выводится сообщение «Не правильная команда».

```
while (1) {
  byte = receive();//Прием
  transmit(byte);  //Печать приема
  buf[i] = byte; //Поместить в очередь
  if(i<3) { i++; } //Сдвиг указателя очереди
  else {  buf[0] = buf[1]; buf[1] = buf[2]; buf[2] = byte; } //сдвиг очереди
  if(byte==ENTER) { // Если конец запроса, то начать обработку очереди,
    i=0; // обнулить указатель очереди
    if(buf[0]=='R' && (buf[1]=='0'||buf[1]=='1')) { //Если 'R' и '0' или '1',
      transmit('U'); transmit(buf[1]); transmit('=');// то печать "U0/1="
      buf[1] = buf[1]-'0'; val = read_adc(buf[1]); // чтение АЦП(0 или 1)
      u16to4dec(val);// преобразование числа в ряд цифр и вывод в окно
      transmit(b[3]+'0');transmit(b[2]+'0');transmit(b[1]+'0');
transmit(b[0]+'0');
      transmit(ENTER);  transmit('>');//Приглашение ко вводу
    }
    else if(buf[0]=='T'&&(buf[1]=='0'||buf[1]=='1')) {//Если 'T', '0' или '1'
      transmit('T');   //, то печать "T",
      if(buf[1]=='0') { val = read_adc(0); OCR1A = val>>2;//результат на ЦАП1
      } else {          val = read_adc(1); OCR1B= val>>2; //или ЦАП2  }
      transmit(ENTER);  transmit('>');//Приглашение ко вводу
    }  else  {
      putstring("Не правильная команда"); transmit(ENTER); transmit('>'); }
  } // Конец обработчика очереди
} // End of main cycle
```

7.7. Стандарты электрические RS-232, RS-485

Помехозащищенность линий связи с синфазными однополярными сигналами стандартных уровней КМОП/ТТЛ не высока – примерно одна треть напряжения питания VCC. Линия нуля питания GND является общей для всех сигналов, при длинных линиях связи растет ее сопротивление и падение напряжения от всех сигналов. Поэтому для связи между удаленными устройствами и/или в условиях повышенного уровня помех используют внешние формирователи уровней (приемопередатчики), преобразующие стандартные уровни и схемотехнику КМОП в уровни и схемотехнику одного из рекомендованных электрических стандартов RS-232, RS-422, RS-485.

1. *UART* + *RS*-232 = *COM-port* используют в персональных компьютерах (ПК) для связь ПК с модемом. Это связь только двух устройств (*PnP*) на небольшие расстояния с невысокими скоростями (15 м при 19 200 бод или 3 м при 115 200 бод) по синфазной

(несимметричной) линии связи, аппаратно поддерживается режим дуплекса, но в приложениях чаще используется полудуплекс. Стандарт *RS*-232 (первоначальное название, сейчас описывается целым семейством стандартов) предусматривает передачу логической. 1 сигналом напряжения –15…–3 В, логического 0 – сигналом +3…+15 В, гальваническая развязка не предусмотрена.

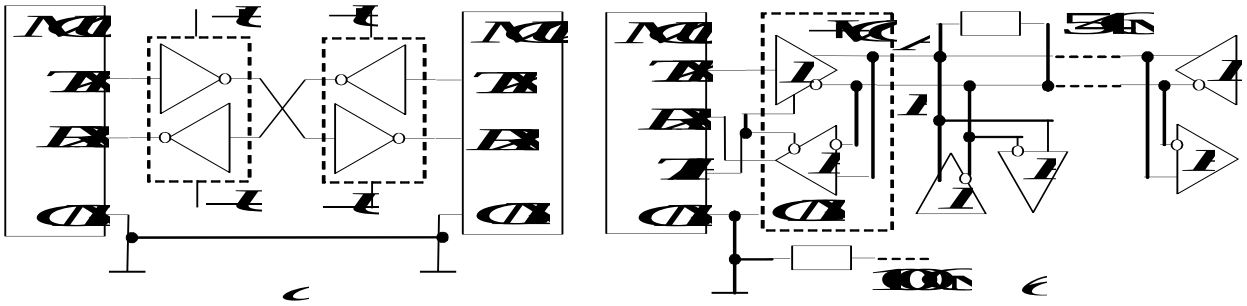


Рис. 7.21

На рис. 7.21, *а* показана схема связи двух МК через приемопередатчики, выполняющие инверсию сигнала и преобразование уровней в двуполярный, приемопередатчики требуют двуполярного питания.

На рис. 7.22 приведена схема подключения к МК (*D*1) микросхемы усилителя-формирователя *ADM*202 (*D*4), имеющей в своем составе два усилителя передатчика, два усилителя приемника, преобразователь напряжения питания из 5 В в двуполярное на переключаемых конденсаторах, защиту входов от статического электричества. Для организации гальванической развязки *RXD, TXD* используются оптопары 6*N*137 (*D*3, *D*4) [*www.firchildsemi.com*], для развязки питания – микросхема импульсного преобразователя постоянного напряжения в постоянное *V*1-0505*SS* (*D*2) [*www.motien.com*]. Нагрузочная способность выхода *R*1*out* ИМС *ADM*202 не достаточна для непосредственного управления светодиодом оптопары, поэтому используется ключевой усилитель на транзисторе *VT*1.

2. *UART* + *RS*-422/485 = шина локального управления, на базе которой построены протоколы в промышленных распределенных системах управления нижнего и среднего уровней. Стандарты *RS*-422/485 используют дифференциальную, т. е. симметричную линию связи (витая пара, можно дополнить экранированием) длиной до 1200 м при скоростях передачи до 500 Кбод.

На рис. 7.21, *б* представлен МК и «его» приемопередатчик, далее дифферециальная линии связи A и B, приемопередатчики других МК и нагрузочный резистор 54 Ом, называемый «терминальным». (Его роль – формирование тока в линии связи и согласование импедансов. Резистор 100 Ом иногда используется для объединения нулей питания, если требуется выравнивание потенциалов.) Каждый приемопередатчик может находиться в активном или в пассивном состоянии, выбор состояния выполняет МК через отдельный выход (T/R на рис. 7.21, *б*). В активном состоянии (режим передачи) при передаче логической 1 на прямом выходе (*A*) 3.3…4.7 В, на инверсном (*B*) 1…1.8 В, при передаче логического 0 уровни меняются на противоположные. В пассивном состоянии (режим приема) приемопередатчик представляет собой высокоимпедансную (*Z*) нагрузку и не влияет на линию связи. Гальваническая развязка стандартом не предусмотрена.

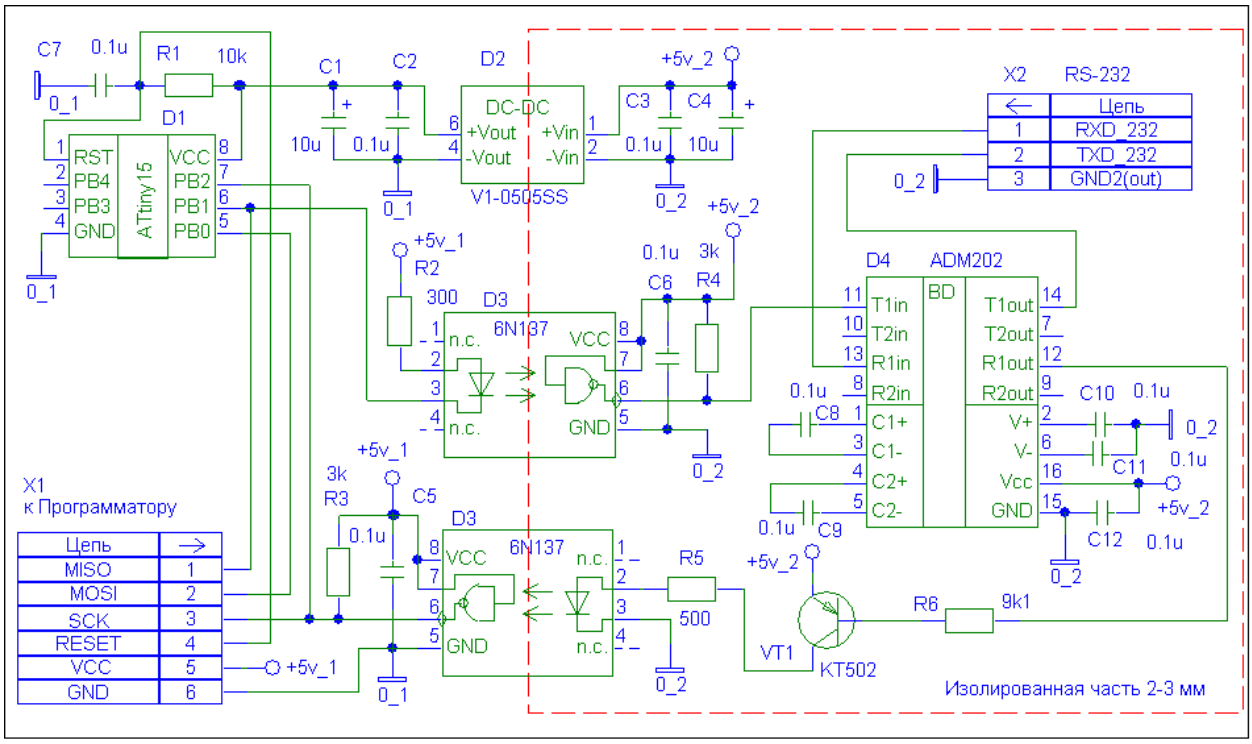


Рис. 7.22

В данной схеме возможен только режим полудуплекс, когда одно устройство в локальной сети является «ведущим» и активизирует свой передатчик для передачи сообщения одному из адресуемых «ведомых», после чего переходит в режим приема (выключает передатчик) для получения ответа от выбранного ведомого. После получения ответа или по окончании заданного времени (тайм-аут приема) ведущий вновь активизирует свой передатчик. Конфликт при попытке включения сразу нескольких передатчиков называется коллизией и решается программным способом в протоколах связи.

Микросхема *ADM*485 содержит передатчик и приемник с питанием +5 В и двумя входами разрешения – передатчика *DE* и приемника #*RE*.

7.8. Программная реализация последовательного интерфейса

В принципе, возможна полностью программная реализация протоколов связи, то есть без использования аппаратных контроллеров с регистрами данных, управления, сдвиговыми регистрами и пр. В этом случае все операции выполняются программно. Это обеспечивает максимальную гибкость в выборе типа и разновидности протокола связи, полную свободу в выборе выводов МК, но существенно снижает частоту тактирования последовательного интерфейса (особенно в режиме приема) и загружает процессор рутинными действиями. Программная реализация протоколов связи актуальна прежде всего в бюджетных МК с малым числом выводов (например, подсемейство Tiny семейства AVR), в которых отсутствуют аппаратные контроллеры последовательных интерфейсов.

Для начала рассмотрим программную реализацию синхронного трехпроводного протокола (типа SPI) в роли ведомого, так как ведущим обычно выступает МК с использованием аппаратного контроллера. Основная задача – выявление фронтов и срезов на входе тактов SCK, которые синхронизируют как процесс приема (вход SI), так и передачи (выход SO). Удобно использовать в качестве входа SCK вход внешнего

прерывания INTx, настроенный на аппаратное выявление среза и фронта импульса.

На рис. 7.23 представлены два варианта временных диаграмм: режим Mode 0 с передачей начиная с младшего бита, режим Mode 3 с передачей начиная со старшего бита. На рис. 7.24 представлены варианты блок-схем подпрограммы прерываний ISR INTx для вариантов диаграмм.

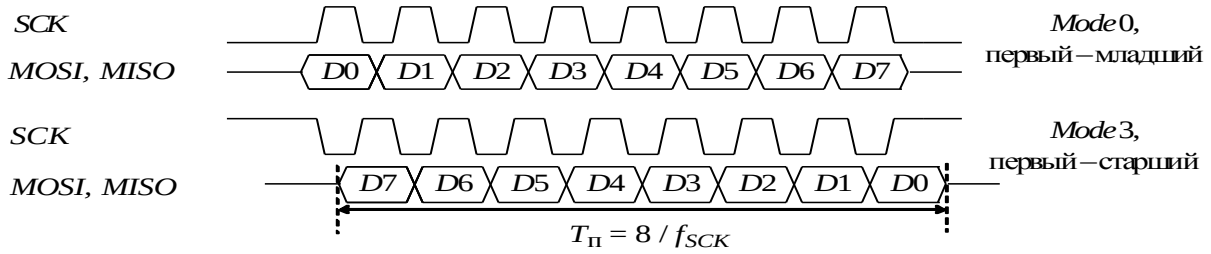


Рис. 7.23

tr, *rr* – регистры данных передатчика и приемника, *pbit* – указатель-счетчик бит

Ведущий, *SCK*, *MO* – выходы, *MI* – вход Ведомый, *SCK*, *SI* – входы, *SO* – выход

Mode 0, первый – младший

$SCK_{init} = 0$

```
for (rr = 0, pbit = 0x80; pbit >>= 1;) {
      if (tr & pbit)   SETBIT (MO);
      else CLRBIT   (MO);
      WAIT1;
      SETBIT (SCK);
      WAIT2;
      if (MI) rr |= pbit;
     CLRBIT (SCK);                    }
```

```
for (rr = 0, pbit = 0x80; pbit >>= 1;) {
      if (tr & pbit)   SETBIT (SO);
      else CLRBIT   (SO);
      while ( !SCK);
      if (SI) rr |= pbit;
      while ( SCK);
 }
```

Mode 3, первый – старший

$SCK_{init} = 1$

```
for (rr = 0, pbit = 0x01; pbit <<= 1;) {
   CLRBIT (SCK);
   WAIT1;
   if (tr & pbit)   SETBIT (MO);
   else CLRBIT   (MO);
   WAIT2;
   SETBIT (SCK);
   if (MI) rr |= pbit;
}
```

```
for (rr = 0, pbit = 0x01; pbit <<= 1;) {
   while ( !SCK);
   if (tr & pbit) SETBIT (SO);
   else CLRBIT   (SO);
   while ( SCK);
   if (SI) rr |= pbit;
}
```

Рис. 7.24

В режим Mode 0 уровень младшего бита регистра передатчика OUT_REG копируется на выход данных SO до прихода первого тактового импульса. Например, сразу после записи нового значения в регистр передатчика, либо по фронту синхронизации выборки ведомого. По фронту SCK выполняется чтение уровня на входе SI и его копирование в младший бит регистра приемника IN_REG, затем сдвиг содержимого регистра приемника влево. По срезу выполняется копирование уровня старшего бита регистра передатчика OUT_REG на выход SO, затем сдвиг содержимого регистра передатчика вправо. Одновременно ведется счет тактов, после отсчета 8 тактов устанавливается флаг завершения процедуры приема/передачи flag_byte. Период тактов должен быть достаточным для поочередного выполнения каждой из ветвей подпрограммы.

В режиме Mode 3 процесс вывода начинается после первого среза тактов с вывода старшего бита на выход SO и заканчивается после восьмого фронта тактов чтением

младшего бита со входа SI, в остальном процессы аналогичные.

При реализации роли ведущего требуется тактирование. При работе на высоких частотах тактов естественные задержки программы определят скорость тактов, нестабильность частоты для синхронных протоколов не опасна.

При сильном снижении частоты тактов удобно использовать прерывания от таймера. Этот способ характерен при реализации асинхронных протоколов. Рассмотрим пример программных алгоритмов приема и передачи байта по асинхронному протоколу.

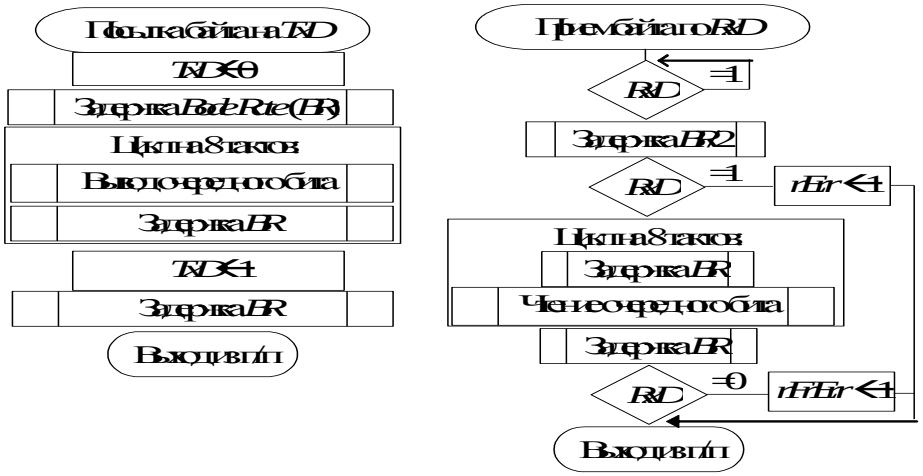


Рис. 7.25

Программная реализация приема/передачи байта по протоколу *UART*:

1. В основном цикле – монопольные задачи (рис. 7.25), только полудуплекс. В процессе приема возможно выявление ошибки приема (флаг *rErr* – байт не принят) и ошибки формата приема (флаг *rFrErr* – стоп-бит равен нулю).

2. С использованием прерывания таймера по переполнению для тактирования передачи и приема (дуплекс и полудуплекс). Передача тактируется на частоте связи. Синхронизация со старт-битом при приеме выполняется либо на утроенной частоте (1 – 0 – 0, на следующих 9 тактах делить частоту на 3), либо с использованием входа прерывания *INTx* (срез), затем по таймеру половина такта и проверка на ноль, и еще 9 тактов.

7.9. Сетевые протоколы и их стандартизация

Различают связь типа «точка-точка» (*Point to Point Interface*), когда информацией обмениваются только два устройства и «многоточечный» интерфейс (*Multy Point Interface*), когда информацией обменивается несколько устройств (сеть). Во втором случае возникают дополнительные задачи: адресации устройств в сети, топология сети, то есть геометрия и способы подключения линий связи между устройствами. При этом различают физические и логические каналы связи, зачастую они не совпадают.

Разнообразие прикладных задач и конкуренция производителей породили множество опробованных решений (протоколов), но постоянно появляются и новые. Большинство промышленных протоколов связи являются закрытыми, то есть рассчитанными на применение устройств одной фирмы разработчика. В момент появления на рынке обычно рекламируются только пользовательские характерстики новых протоколов, технические описания не афишируются. При успешном продвижении новой технологии у конечных пользователей появляется желание объединить

преимущества разных фирм, разработчики готовых средств автоматизации идут на встречу и «осваивают» эти закрытые протоколы.

Некоторые успешные и проверенные временем решения становятся «стандартом де факто», их протоколы и рекомендации по реализации становятся полностью окрытыми.

Стремление к унификации и стандартизации оборудования, программ и методов передачи информации в области автоматизации привели к разработке эталонной модели взаимодействия открытых систем (ВОС) *[Open System Interconnections – OSI]*, которая описывает рекомендуемые подходы при разработке стандартов, разбивая протоколы связи на семь возможных уровней описания.

Уровни протоколов OSI (7 уровней):

1. Физический (Physical Layer)
   - Уровни сигналов и представление бит (Signal Level and Bit Representation)
   - Среда передачи (Transmission Medium)
2. Транспортный (Transfer Layer)
   - Выявление неисправностей (Fault Confinement)
   - Выявление ошибок и передача сигналов (Error Detection and Signaling)
   - Проверка достоверности сообщения (Message Validation)
   - Уведомление (об успешном приеме данных) (Acknowledgment)
   - Разрешение конфликтов (Arbitration)
   - Формирование и синхронизация кадра сообщения (Message Framing)
   - Скорость передачи и временная диаграмма (Transfer Rate and Timing)
3. Объектный (Object Layer)
   - Фильтрация сообщений (Message Filtering)
   - Управление сообщениями и состоянием (Message and Status Handling)
4. Уровень приложения (Application Layer)
5. …

Более высокие уровни здесь не рассматриваются, так как они не определены в протоколах ПИ, применяемых в МК для встраиваемых приложений.

Промышленная сеть — сеть передачи данных, связывающая различные датчики, исполнительные механизмы, промышленные контроллеры и используемая в промышленной автоматизации. Термин употребляется преимущественно в автоматизированной системе управления технологическими процессами (АСУ ТП). Описывается стандартом IEC 61158.

Устройства используют сеть для:

- передачи данных, между датчиками, контроллерами и исполнительными механизмами,
- диагностики и удалённого конфигурирования датчиков и исполнительных механизмов,
- калибрования датчиков,
- питания датчиков и исполнительных механизмов,
- передачи данных между датчиками и исполнительными механизмами минуя центральный контроллер,
- связи между датчиками, исполнительными механизмами, ПЛК и АСУ ТП верхнего уровня
- связи между контроллерами и системами человеко-машинного интерфейса (SCADA)

В промышленных сетях для передачи данных применяют:

- кабели
- волоконно-оптические линии
- беспроводную связь (радиомодемы и Wi-Fi).

Промышленные сети могут взаимодействовать с обычными компьютерными сетями, в частности использовать глобальную сеть Internet.

Термин **полевая шина** является дословным переводом английского термина *fieldbus*. Термин **промышленная сеть** является более точным переводом и в настоящее время именно он используется в профессиональной технической литературе.

****

Распределённая система управления *[Distributed Control System, DCS]* — система управления технологическим процессом, характеризующаяся построением распределённой системы ввода вывода и децентрализацией обработки данных.

РСУ применяются для управления *непрерывными* и гибридными технологическими процессами (хотя, строго говоря, сфера применения РСУ только этим не ограничена). К непрерывным процессам можно отнести те, которые должны проходить днями и ночами, месяцами и даже годами, при этом остановка процесса, даже на кратковременный период, может привести к порче изготавливаемой продукции, поломке технологического оборудования и даже несчастным случаям. Классическим примером непрерывного процесса является изготовление стекла в стекловаренной печи.

Сферы применения РСУ многочисленны:

1.Химия и нефтехимия.

2.Нефтепереработка и нефтедобыча.

3.Стекольная промышленность.

4.Пищевая промышленность: молочная, сахарная, пивная.

5.Газодобыча и газопереработка.

6.Металлургия.

7. Энергоснабжение и т. д.

Требования к современной РСУ:

1.Отказоустойчивость и безопасность.

2.Простота разработки и конфигурирования.

3.Поддержка территориально распределенной архитектуры.

4.Единая конфигурационная база данных.

5.Развитый человеко-машинный интерфейс.

****

АСУ ТП: Историческое обоснование перехода от централизованного построения к децентрализованному (распределенному) - дешевизна и миниатюризация, развитие ПИ (дальность и помехоустойчивость).

Принципы построения децентрализованных (распределенных) СУ (посм. В Инете):

1) модульность в разбиении алгоритмов, то есть выделение стандартных действий: измерения, фильтрация, математическая обработка, генерация сигналов, протоколирование нештатных ситуаций и пр. и выбор оптимальных аппаратно-программных модулей для их реализации;

2) быстрые связи (время цикла менее 1...100 мс) одного локального объекта управления предпочтительно замыкать на одно процессорное устройство.

Уровни иерархии при построении АСУ ТП (снизу вверх – от уровня технолоческого узла/установки до уровня планирования и управления производственными показателями):

0) обычно процесс автоматизации "растет" снизу вверх, т.к. вместе с разумной механизацией это сразу дает резкое увеличение производительности труда при сравнительно небольших затратах.

1) (интеллектуальные) датчики и исполнительные органы;

2) устройства управления уровня технологического узла/установки - либо встроенные на базе МК для встриваемых приложений

7.10. Открытый протокол Modbus]

Modbus – открытый коммуникационный протокол, основанный на архитектуре <клиент-сервер>. Широко применяется в промышленности для организации связи между электронными устройствами. Может использовать для передачи данных последовательные линии связи RS-485, RS-422, RS-232, а также сети TCP/IP.

История

Modbus был разработан фирмой Modicon (в настоящее время принадлежит Schneider Electric) для использования в её контроллерах с программируемой логикой. Впервые спецификация протокола была опубликована в 1979 году. Это был открытый стандарт, описывающий формат сообщений и способы их передачи в сети состоящей из различных электронных устройств.

Первоначально контроллеры MODICON использовали последовательный интерфейс RS-232. Позднее стал применяться интерфейс RS-485, так как он обеспечивает более высокую надёжность, позволяет использовать более длинные линии связи и подключать к одной линии несколько устройств.

Многие производители электронного оборудования поддержали стандарт, на рынке появились сотни использующих его изделий. В настоящее время развитием Modbus занимается некоммерческая организация Modbus-IDA, созданная производителями и пользователями электронных приборов.

Несмотря на солидный возраст, благодаря простоте аппаратной и программной реализации, Modbus остается одним из самых распространенных протоколов. В России он конкурирует с гораздо более дорогим современным протоколом Profibus.

Краткое описание и технические характеристики

Modbus относится к протоколам прикладного уровня сетевой модели OSI. Контроллеры на шине Modbus взаимодействуют, используя клиент-серверную модель, основанную на транзакциях, состоящих из кадра запроса и кадра ответа.

Обычно в сети есть только один клиент (ведущий, англ. *master*) и несколько серверов (ведомых, англ. *slaves*). Ведущий инициирует транзакции (передаёт кадр запроса), ведомый формирует и посылает кадр ответа, при получении правильного в заданное время ведущий завершает транзакцию. Далее все може повторяться.

Запрос содержит адрес ведомого (1 байт), команду (1 байт) и данные для передачи ведомому, либо информацию о данных, требуемых от ведомого (варьируемое число байт). Ответ аресуемого ведомого содержит его адрес, команду, далее либо информация об успешном принятии данных, либо запрошенные данные, либо сообщение об ошибке, если запрос не может быть выполнен.

При необходимости ведущий може обратиться сразу ко всем устройствам на линии связи (по широковещательному адресу), в этом случае ответ ведомого не требуется (иначе бы они «смешались»).

Данные для обмена представлены битами, группами бит, двухбайтными словами и группами двухбайтных слов, группы непрерывны. Стандартных команд не много, они состоят из группы команд записи (то есть передачи данных ведомому) и группы команд чтения (получение данных от ведомого). Конкретные команды различают тип данных (бит или два байта) и один элемент данных или группа. В структуру команд запроса данных входит адрес данных, при запросе группы – ее размер.

Длина запроса и ответа варьируется от не скольких байт, до нескольких сотен байт.

Для контроля достверности передаваемой информации на уровне байта исползуется механизмы контроллера UART (контроль по стоп-биту и (при выборе) по биту четности/нечетности), на уровне структуры запроса/ответа – передача контрольной суммы байт вычисляемая по определенному алгоритму (зависит от модификации протокола).

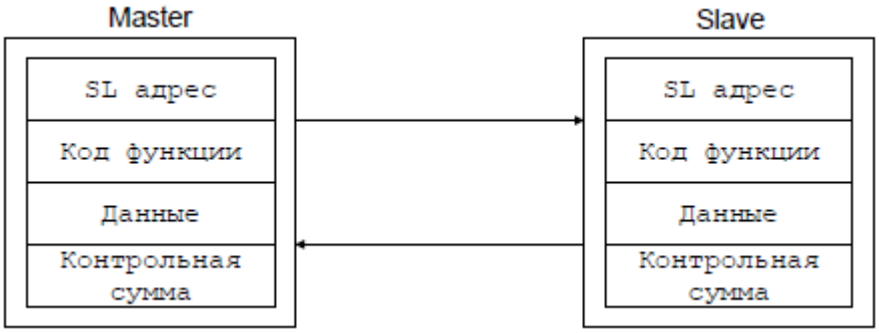


Рис. 7.26

Ведомый, обнаружив свой адрес, ведет прием до появления символа окончания запроса. Для модификации протокола Modbus ASCII это символы <CR>, <LF>, для Modbus RTU это 3.5 байта «тишины» (отсутствие посылок).

Ведущий, выдав запрос, обычно переходит в режим прослушивания линии связи. Он находится в этом состоянии либо до завершения приема ответа, начинающегося с переданного в запросе адреса и заканчивающегося соответствующими символами, либо ждет до окончания времени ожидания (таймаута), после чего может начать повторный запрос, число повторных запросов [*retrive*] тоже настраивается заранее.

Практика прменения

Здесь надо про настройки параметров Ведущего/Ведомого

Подробное описание

Спецификация Modbus описывает структуру запросов и ответов. Их основа – элементарный пакет протокола, так называемый PDU (Protocol Data Unit). Структура PDU не зависит от типа линии связи и включает в себя код функции и поле данных. Код функции кодируется однобайтовым полем и может принимать значения в диапазоне 1...127. Диапазон значений 128...255 зарезервирован для кодов ошибок. Поле данных может быть переменной длины. Размер пакета PDU ограничен 253 байтами.

Modbus PDU

номер функции	Данные
1 байт	N < 253 (байт)

Для передачи пакета по физическим линиям связи PDU помещается в другой пакет, содержащий дополнительные поля. Этот пакет носит название ADU (Application Data Unit). Формат ADU зависит от типа линии связи.

Существуют три основных реализации протокола Modbus.

Modbus RTU и *Modbus ASCII* применяют для передачи данных по последовательным линиям связи, как медным EIA/TIA-232-E (RS-232), EIA-422, EIA/TIA-485-A (RS-485), так и оптическим и радио. В Modbus RTU (Remote Terminal Unit) байт в посылке передается без изменений, в Modbus ASCII байт передается двумя символами (байтами) кода ASCII, например, байт 0x12 будет разбит на байты ‘1’ = 0x31 и ‘2’ = 0x32.

Табл. 7.? Характерстики Modbus ASCII и Modbus RTU

Характеристика	ASCII (7-бит)	RTU(8-бит)
Система кодирования	Используются ASCII символы 0-9,A-F	8-битовая двоичная система
Число бит на символ		
Стартовые биты	1	1
Биты данных (LSB вперед)	7	8
Четность	Вкл./Выкл.	Вкл./Выкл.
Стоповые биты	1 или 2	1 или 2
Контрольная сумма	LRC (Longitudinal Redundancy Check). LRC	CRC (Cyclical Redundancy Check). CRC 16

Modbus TCP применяют для передачи данных по сетям Ethernet поверх TCP/IP.

Общая структура ADU следующая (в зависимости от реализации, некоторые из полей могут отсутствовать):

адрес ведомого устройства *код функции* *Данные* *блок обнаружения ошибок*

- *адрес ведомого устройства* - адрес подчинённого устройства, к которому адресован запрос. Ведомые устройства отвечают только на запросы, поступившие в их адрес. Ответ также начинается с адреса отвечающего ведомого устройства, который может изменяться от 1 до 247. Адрес 0 используется для широковещательной передачи, его распознаёт каждое устройство, адреса в диапазоне 248...255 - зарезервированы;
- *номер функции* - это следующее однобайтное поле кадра. Оно говорит ведомому устройству, какие данные или выполнение какого действия требует от него ведущее устройство;
- *данные* - поле содержит информацию, необходимую ведомому устройству для выполнения заданной мастером функции или содержит данные, передаваемые ведомым устройством в ответ на запрос ведущего. Длина и формат поля зависит от номера функции;
- *блок обнаружения ошибок* - контрольная сумма для проверки отсутствия ошибок в кадре.

Максимальный размер ADU для последовательных сетей RS232/RS485 - 256 байт, для сетей TCP - 260 байт.

Категории кодов функций

В действующей в настоящее время спецификации протокола определяются три категории кодов функций:

Стандартные команды

Их описание должно быть опубликовано и утверждено Modbus-IDA. Эта категория включает в себя как уже определенные, так и свободные в настоящее время коды.

Пользовательские команды

Два диапазона кодов (от 65 до 72 и от 100 до 110), для которых пользователь может реализовать произвольную функцию. При этом не гарантируется, что какое-то другое устройство не будет использовать тот же самый код для выполнения другой функции.

Зарезервированные

В эту категорию входят коды функций, не являющиеся стандартными, но уже используемые в устройствах, производимых различными компаниями. Это коды 9, 10, 13, 14, 41, 42, 90, 91, 125, 126 и 127.

Модель данных

Одно из типичных применений протокола – чтение и запись данных в регистры контроллеров. Спецификация протокола определяет четыре таблицы данных:

Таблица	**Тип элемента**	**Тип доступа**	**Адрес в Modicon**
Дискретные входы (*Discrete Inputs*)	один бит	только чтение	
Регистры флагов (*Coils*)	один бит	чтение и запись	00000…
Регистры ввода (*Input Registers*)	16-битное слово	только чтение	30000…
Регистры хранения (*Holding Registers*)	16-битное слово	чтение и запись	40000…

Доступ к элементам в каждой таблице осуществляется с помощью 16-битного адреса, первой ячейке соответствует адрес 0. Таким образом, каждая таблица может содержать до 65536 элементов. Спецификация не определяет, что физически должны представлять собой элементы таблиц и по каким внутренним адресам устройства они должны быть доступны. Например, допустимо организовать перекрывающиеся таблицы, В этом случае команды, работающие с дискретными данными (битами) и с 16-битными регистрами будут фактически обращаться к одним и тем же данным.

Следует отметить, что со способом адресации данных связана определённая путаница. Modbus был первоначально разработан для контроллеров Modicon. В этих контроллерах для каждой из таблиц использовалась специальная нумерация. Например, первому регистру ввода соответствовал номер ячейки 30001, а первому регистру хранения - 40001. Таким образом, регистру хранения с адресом 107 в команде Modbus соответствовал регистр № 40108 контроллера. Хотя такое соответствие адресов больше не является частью стандарта, некоторые программные пакеты могут автоматически <корректировать> вводимые пользователем адреса, например, вычитая 40001 из адреса регистра хранения.

Стандартные функции протокола Modbus

PDU запроса и ответа для стандартных функций

Запрос/ответ							Command	Команда
1 (0x01)	A1	A0	Q1	Q0			*Read Coil Status*	Чтение неск. рег. Флагов
N	D (N байт)							
2 (0x02)	A1	A0	Q1	Q0			*Read Discrete Inputs*	Чтение неск. дискр. рег.
N	D (N байт)							
3 (0x03)	A1	A0	Q1	Q0			*Read Holding Registers*	Чтение неск. рег. Хранения
N	D (N байт)							
4 (0x04)	A1	A0	Q1	Q0			*Read Input Registers*	Чтение неск. рег. Ввода
N	D (N байт)							
5 (0x05)	A1	A0	D1	D0			*Force Single Coil*	Запись одного флага
A1	A0	D1	D0					
6 (0x06)	A1	A0	D1	D0			*Preset Single Register*	Запись одного рег. Хранения
A1	A0	D1	D0					
15 (0x0F)	A1	A0	Q1	Q0	N	D (N байт)	*Force Multiple Coils*	Запись неск. рег. Флагов
A1	A0	Q1	Q0					
16 (0x10)	A1	A0	Q1	Q0	N	D (N байт)	*Preset Multiple Registers*	Запись неск. рег. Хранения
A1	A0	Q1	Q0					

- **A1** и **A0** - адрес элемента,
- **Q1** и **Q0** - количество элементов,
- **N** - количество байт данных
- **D** – данные

Чтение данных

Для чтения значений из перечисленных выше таблиц данных используются функции с кодами 1-4 (шестнадцатеричные значения 0x01-0x04):

- **1 (0x01)** - чтение значений из нескольких регистров флагов *(Read Coil Status)*
- **2 (0x02)** - чтение значений из нескольких дискретных регистров *(Read Discrete Inputs)*
- **3 (0x03)** - чтение значений из нескольких регистров хранения *(Read Holding Registers)*
- **4 (0x04)** - чтение значений из нескольких регистров ввода *(Read Input Registers)*

Запрос состоит из адреса первого элемента таблицы, значение которого требуется прочитать, и количества считываемых элементов. Адрес и количество данных задаются 16-битными числами, старший байт каждого из них передается первым.

В ответе передаются запрошенные данные. Количество байт данных зависит от количества запрошенных элементов. Перед данными передается один байт, значение которого равно количеству байт данных.

Значения регистров хранения и регистров ввода передаются начиная с указанного адреса, по два байта на регистр, старший байт каждого регистра передаётся первым:

Байт 1	Байт 2	Байт 3	Байт 4	...	Байт N-1	Байт N
RA,1	RA,0	RA+1,1	RA+1,0	...	RA+Q-1,1	RA+Q-1,0

Значения флагов и дискретных входов передаются в упакованном виде: по одному биту на флаг. Единица означает включённое состояние, ноль – выключенное. Значения запрошенных флагов заполняют сначала первый байт, начиная с младшего бита, затем следующие байты, также от младшего бита к старшим. Младший бит первого байта данных содержит значение флага, указанного в поле <адрес>. Если запрошено количество флагов, не кратное восьми, то значения лишних битов заполняются нулями:

Запись одного значения

- **5 (0x05)** - запись значения одного флага *(Force Single Coil)*
- **6 (0x06)** - запись значения в один регистр хранения *(Preset Single Register)*

Команда состоит из адреса элемента (2 байта) и устанавливаемого значения (2 байта).

Для регистра хранения значение является просто 16-битным словом.

Для флагов значение 0xFF00 означает включённое состояние, 0x0000 - выключенное, другие значения недопустимы.

Если команда выполнена успешно, ведомое устройство возвращает копию запроса.

Запись нескольких значений

- **15 (0x0F)** - запись значений в несколько регистров флагов *(Force Multiple Coils)*
- **16 (0x10)** - запись значений в несколько регистров хранения *(Preset Multiple Registers)*

Команда состоит из адреса элемента, количества изменяемых элементов, количества передаваемых байт устанавливаемых значений и самих устанавливаемых значений. Данные упаковываются так же, как в командах чтения данных.

Ответ состоит из начального адреса и количества изменённых элементов.

Ниже приведён пример запроса ведущего устройства и ответа ведомого (для Modbus RTU).

Master → Slave

00 Адрес подч.	01 Номер функ-ции	02 Адрес ст. байт	03 Адрес мл. байт	04 Кол. флагов ст. байт	05 Кол. флагов мл. байт	06 Кол. байт данных	07 Данные (флаги биты 0-7)	08 Данные (флаги биты 8-15)	09 CRC мл. байт	0A CRC ст. байт
0x01	0x0F	0x00	0x13	0x00	0x0A	0x02	0xCD	0x01	0x72	0xCB

Slave → Master

00 адрес подчиненно	01 номер функции	02 Адрес ст. байт	03 Адрес мл. байт	04 Кол. флагов ст.	05 Кол. флагов мл.	05 CRC мл. байт	06 CRC ст. байт

го				байт	байт		
0x01	0x0F	0x00	0x13	0x00	0x0A	0x24	0x09

Контроль ошибок в протоколе Modbus RTU

Во время обмена данными могут возникать ошибки двух типов:

- ошибки, связанные с искажениями при передаче данных;
- логические ошибки.

Ошибки первого типа обнаруживаются при помощи фреймов символов, контроля чётности и циклической контрольной суммы CRC-16-IBM (используется число-полином = 0xA001).

RTU фрейм

В RTU режиме сообщение должно начинаться и заканчиваться интервалом тишины - временем передачи не менее 3.5 символов при данной скорости в сети. Первым полем затем передаётся адрес устройства.

Вслед за последним передаваемым символом также следует интервал тишины продолжительностью не менее 3.5 символов. Новое сообщение может начинаться после этого интервала.

Фрейм сообщения передаётся непрерывно. Если интервал тишины продолжительностью 1.5 возник во время передачи фрейма, принимающее устройство должно игнорировать этот фрейм как неполный.

Таким образом, новое сообщение должно начинаться не раньше 3.5 интервала, т.к. в этом случае устанавливается ошибка.

Немного об интервалах (речь идёт о Serial Modbus RTU): при скорости 9600 и 11 битах в кадре (стартовый бит + 8 бит данных + бит контроля чётности + стоп-бит): 3.5 * 11 / 9600 = 0,00401041(6), т.е. более 4 мс; 1.5 * 11 / 9600 = 0,00171875, т.е. не более 1 мс. Для скоростей более 19200 бод допускается использовать интервалы 1,75 и 0,75 мс соответственно.

Логические ошибки

Для сообщений об ошибках второго типа протокол Modbus RTU предусматривает, что устройства могут отсылать ответы, свидетельствующие об ошибочной ситуации. Признаком того, что ответ содержит сообщение об ошибке, является установленный старший бит кода команды. Пример кадра при выявлении ошибки ведомым устройством, в ответ на запрос приведён в (Таблица 2-1).

1. Если Slave принимает корректный запрос и может его нормально обработать, то возвращает нормальный ответ.
2. Если Slave не принимает какого-либо значения, никакого ответа не отправляется. Master диагностирует ошибку по таймауту.
3. Если Slave принимает запрос, но обнаруживает ошибку (parity, LRC, or CRC), никакого ответа не отправляется. Master диагностирует ошибку по таймауту.
4. Если Slave принимает запрос, но не может его обработать (обращение к несуществующему регистру и т.д.), отправляется ответ, содержащий в себе данные об ошибке.

Таблица 2-1. Кадр ответа (SlaveMaster) при возникновении ошибки Modbus RTU

Направление	**адрес подчинённого**	**номер**	**данные (или код**	**CRC**

передачи	устройства	функции	ошибки)	
Запрос (MasterSlave)	0x01	0x77	0xDD	0xC7 0xA9
Ответ (SlaveMaster)	0x01	0xF7	0xEE	0xE6 0x7C

Стандартные коды ошибок

- **01** - Принятый код функции не может быть обработан на подчиненном.
- **02** - Адрес данных, указанный в запросе, не доступен данному подчиненному.
- **03** - Величина, содержащаяся в поле данных запроса, является не допустимой величиной для подчиненного.
- **04** - Невосстанавливаемая ошибка имела место, пока подчиненный пытался выполнить затребованное действие.
- **05** - Подчиненный принял запрос и обрабатывает его, но это требует много времени. Этот ответ предохраняет главного от генерации ошибки таймаута.
- **06** - Подчиненный занят обработкой команды. Главный должен повторить сообщение позже, когда подчиненный освободится.
- **07** - Подчиненный не может выполнить программную функцию, принятую в запросе. Этот код возвращается для неудачного программного запроса, использующего функции с номерами 13 или 14. Главный должен запросить диагностическую информацию или информацию об ошибках с подчиненного.
- **08** - Подчиненный пытается читать расширенную память, но обнаружил ошибку паритета. Главный может повторить запрос, но обычно в таких случаях требуется ремонт.

Пример связи МК и панели оператора (СУ ТГИ)

МК ATmega2560 выполняет информационно-измерительные функции и функции управления транзисторным инвертором для индукционного нагрева. Панель оператора MT6070iH (Weintek) имеет цветной ЖК дисплей (800x480, 7’) с сенсорным вводом, является одним из основных элементов пульта управления транзисторного генератора.

На аппаратном уровне схема соединения панели с МК (рис.7.27) состоит из микросхем драйвера RS-485 с гальванической развязкой DD15 (ADM2483rwz), DC-DC с гальванической развязкой DD14 (AM1L-0505S-NZ), разъемов, фильтрующих конденсаторов, резисторов. Дифференциальные линии связи RS-485 (линии A и B) «нагружены» на терминальный резистор R202 (120 Ом) и имеют резисторы притяжки R201, R203; монтаж связей выполнен витой парой.

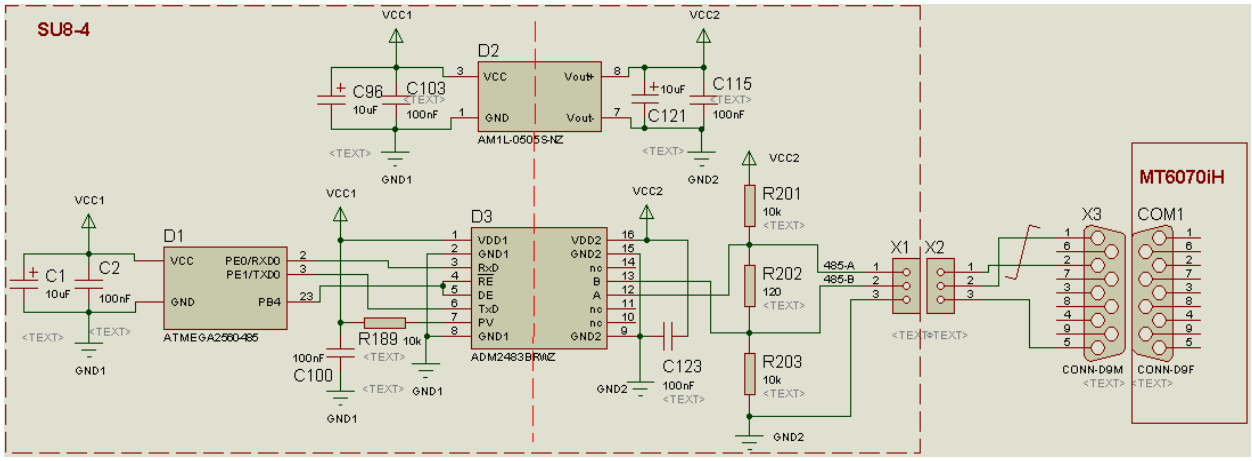


Рис. 7.27

Панель выступает главным устройством, используется порт COM1 RS485-2W (то есть полудуплекс) с настройками байтового асинхронного интерфейса 19200 бод, 8 бит данных, 1 стоп бит, без бита четности, протокол Modbus RTU, адрес единственного ведомого (МК) 1.

Надо минимизировать описание того, что реализуется в панели (не в тему).

Обработка сообщений ModBus

Панель оператора на шине является мастером, а плата системы управления слейвом (ведомый). Это означает, что инициатором начала обмена всегда служит панель, плата системы управления лишь отвечает на запросы.

Пакет данных содержит в себе несколько полей. В зависимости от номера функции длина и содержимое пакета данных могут изменяться, но существуют поля, присутствующие в каждом пакете (таблица *7.8*ца 7.8). Первым байтом передается адрес ведомого устройства, если адрес совпадает, то ведомое устройство дожидается конца сообщения и приступает к обработке данных, иначе просто его игнорирует. Окончанием пакета данных служит пауза - время, необходимое для передачи более 3.5 символов, в течение которого сервер не посылает никаких данных. После обнаружения окончания пакета ведомое устройство начинает проверку правильности передачи, для этого оно рассчитывает контрольную сумму содержимого пакета и сравнивает ее с принятой контрольной суммой. Если они совпадают, ведомое устройство выполняет требуемую функцию и возвращает ответный пакет, в котором могут содержаться необходимые данные или извещение об успешном выполнении. Если контрольная сумма не совпадает, ведомое устройство игнорирует пакет. Блок схема функции обработки сообщения показана на рис. 7.28.

Таблица 7.8

Структура запроса сервера

1 байт	1 байт	2 байта	2 байта	2 байта
Адрес подчиненного устройства	Номер функции	Адрес	Данные	Контрольная сумма

Передача одного бита необходима для манипуляции с параметрами, которые могут принимать два состояния, например, бит аварии по определенному каналу или состояние генератора (генерация включена/выключена). Запись/чтение регистров, каждый из которых представляет собой 2 байта, необходима для передачи чисел, диапазон которых лежит в интервале от 0 до 65535 ($65536 = 2^{16}$). Такими параметрами служат значения измеренной мощности, тока, температуры, расхода воды системы охлаждения и прочие. При входе в функцию обработки сообщений проверяется состояние флага наличия данных (DataAvalable). Этот флаг устанавливается по таймеру (Timer3 файл ModBus.c) при обнаружении признака конца пакета. Если данные доступны, производится расчет контрольной суммы принятых данных и сравнение их с контрольной суммой, находящейся в конце пакета данных. В случае их совпадения проверяется номер команды, и при командах 0x01 или 0x04 (таблица ица) формируется ответный пакет данных, содержащий в себе запрашиваемые данные. Если номер команды равен 0x05 или 0x10

формируется ответный пакет данных, уведомляющий сервер об успешном выполнении записи.

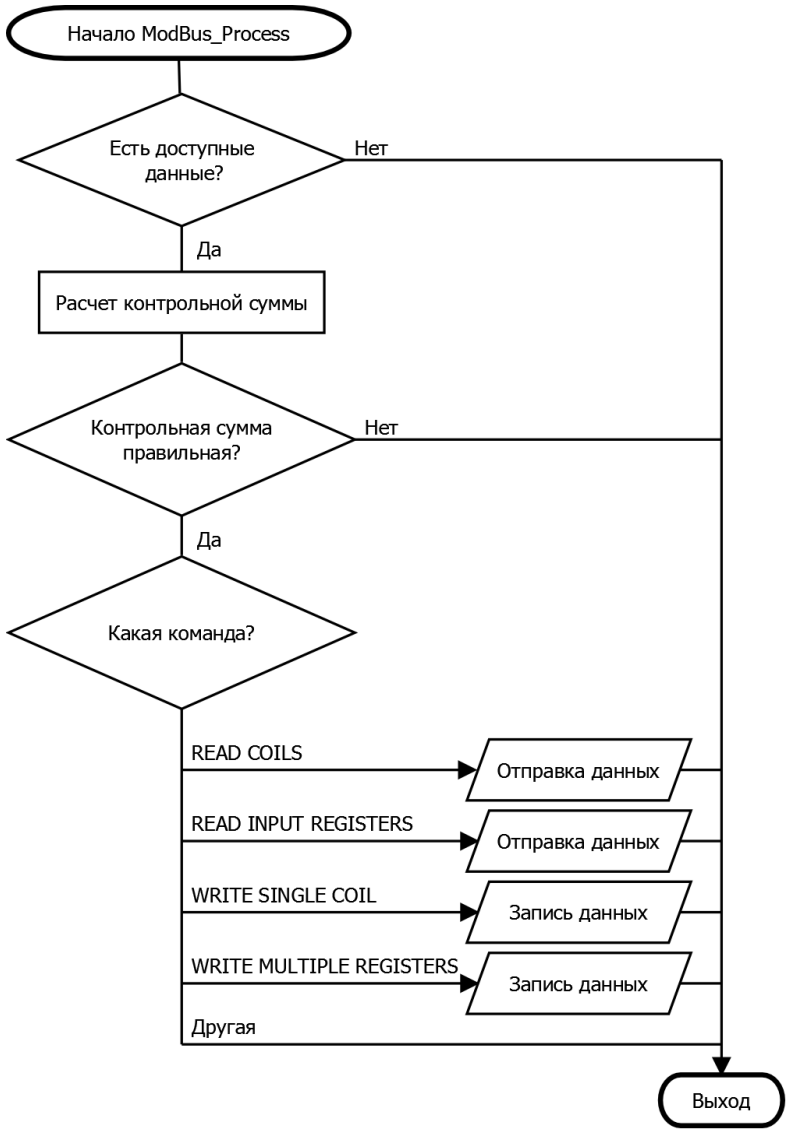


Рисунок.7.28

Таблица 7.9 Список поддерживаемых ModBus функций

Наименование функции	Код функции (шестнадцатеричное)	Назначение
Read Coils	0x01 0b0000 0001	Чтение одного бита
Write Single Coil	0x05 0b0000 0101	Запись одного бита
Read Input Registers	0x04 0b0000 0100	Чтение нескольких регистров
Write Multiple (Holding) Registers	0x10 0b0001 0000	Запись нескольких регистров

Перечень переменных, передаваемых по сети ModBus, состоит из 72 бит (массив unsigned char Coils[CoilSize], CoilSize = 72/8) и 387 регистров – ячеек по 16 бит (unsigned int InputRegisters[inRegSize]), часть регистров оргнизована в массивы. Формально биты и регистры доступны по чтению и записи, в МК часть ячеек имеет временную защиты от записи, например, в процессе нагрева нельзя менять имя и режима; при этом команда записи «выполняется» успешно.

Для удобства разработчика каждая переменная имеет символическое имя, например, бит Coils[AVAR_DR1] или регистр InputRegisters[Pd]. Эти же имена в панели Weintek копируются в таблицу «тэгов» в Address Tag Library без разделения на массивы, различаемые по формату (Modbus RTU bit 0x-1 или Word 3x-1). Выдержка из перечня представлена.в Табл. 7.10.

Таблица 7.10 Перечень переменных платы системы управления (выдержка)

Название переменной	Адрес	Описание	Название Переменной	Адрес	Описание
Переменные типа бит (запись/чтение командами 0x01/0x05)					
AVAR_DR1	0*	Авария драйвера 1	ATS5_HI	25*	Температура датчика 5 больше допустимой
AVAR_DR2	1*	Авария драйвера 2	ATS6_HI	26*	Температура датчика 6 больше допустимой
Переменные типа слово (запись/чтение командами 0x04/0x10)					
Pd	0*	Измеренное значение мощности	DS75_TEMP	22*	Измеренное значение температуры датчика на плате
Ud	1*	Измеренное значение постоянного напряжения	Fset	23*	Уставка частоты

Структура программ ведомого (МК) и расчет быстродействия обмена данными.

При скорости обмена 19200 бод период тактирования составляет 52.1 мкс, с учетом старт, стоп битов передача одного байта занимает не мене 521 мкс. Признаком завершения запроса является интервал тишины длительностью не менее 3.5 байт-интервалов, то есть 1563 мкс.

Прием и передача ведутся с использованием встроенного контроллера асинхронного интерфейса USART0. Прием запроса выполняется по прерыванию приемника:

```
ISR(USART0_RX_vect)  {
    TCCR3B = (1 << CS32);//Пуск тактирования ftc = fclk/256 = 64кГц, 15.6мкс
    TCNT3 = 0;//Обнуление таймера для отсчета межбайтового интервала
    RxBuffer1[Rxcounter1++] = UDR0;//Чтение байта в буфер приема
}
```

Таймер TC3 ведет измерение межбайтовых интервалов, переполнение таймера произойдет через 256 тактов или /64000 = 4 мс, что составляет 7.6 байт-интервалов. Обработка прерывания по переполнению TC3 завершает прием запроса:

```
ISR (TIMER3_COMPA_vect) //сработал таймер - конец фрейма запроса
  TCCR3B = 0; TCNT3 = 0;  //останавливаем и обнуляем TC3
  DataAvalable1 = 1;//устанавливаем флаг наличия данных
  PORTJ ^= (1 << PINJ7);//Переключение светодиода индикации обмена
}
```

Все остальные действия выполняются в основном цикле в функции ModBus1k_Process() с частотой 30 Гц (период 33 мс).

Если установлен флаг наличия данных и первый байт в буфере приема совпадает с адресом ведомого, выполняется разбор принятого запроса. По содержимому кода команды и другим параметрам (число передаваемых или принимаемых байт) определяется расположение байтов контрольной суммы, вычисляется контрольная сумма принятых байт, их совпадение разрешает выполнение команды. Любое несоответствие (адрес, код команды, контрольная сумма) завершается отсутствием действий и молчанием ведомого.

Выполнение команд чтения сводится к копированию данных из массивов Coils или InputRegisters в буфер передачи, команды записи копируют данные из буфера приема в соответствующие биты или регистры (если разрешено).

После выполнения команды заполняется буфер передачи (функция Modbus1k_SendResponce()) и выполняется отправка ответа (функция RS485_U0_send()). Для передачи используется передатчик USART0, драйвер RS-485 (микросхема ADM2483) переводится в режим передачи, собственно пересылка байтов синхронизируется программным опросом флага UDRE0, завершение передачи контролируется по флагу TXC0, драйвер RS-485 переводится в режим приема.

Временные характеристики обмена

Длительность запроса варьируется от 8+3 = 11 до 9 + 3 + 240 (максимальная длина блока данных, заданная в настройках панели Weintek) = 252 байт-интервалов, при тактировании 19200 бод это составит от 0.573 до 13.125 мс. Максимальная задержка начала разбора запроса (функция ModBus1k_Process()) может доходить до двух периодов 30 Гц, то есть 66 мс. Длительность ответа варьируется от 6 + 3 = 9 до 5 + 3 + 240 = 248 байт-интервалов, то есть 0.47 до 12.92 мс. Суммароное время «запрос + ответ» варьируется от 1.1 до 80 мс.

Ведущий (панель) ждет ответа в течение тайм-аута (1000 мс), в отсутствии ответа может начать до N повторных попыток запроса (в данной настройке N = 0), по исчерпании попыток на панели выводится предупреждающая надпись «PLC No Response».

Из приведенных расчетов видно, что основной вклад в длительность обмена информацией вносит период вызова разбора запроса (в среднем 33 мс). Более частое выполнение этой задачи должно несколько снизить общие затраты времени. Частота и «длина» запросов определяются ведущим.

Доступные инструменты программирования и настройки панели не позволяют в явном виде управлять процессом выдачи запросов. Состав и частота запросов зависят от состава переменных текущего экрана, периодически выполняемых задач (макросы и триггерный запуск), и от настроек интерфейса: миксимальная длина блока данных (по чтению и записи, = 120 слов); интервал между ячейками, объединемыми в один запрос (= 5); пауза между запросами (= 0) и пр.

В рассматриваемом примере панель периодически сохраняет во встроенном USB-Flash «историю» работы системы управления в виде численных значений измеряемых переменных и переменных процесса управления. Файл «истории нагрева» содержит 10 значений (7 измеренных и 3 уставки), шаг сохранения во времени 1 секунда. Для ряда процессов желательно иметь шаг 0.1 секунда, пока это не выполняется, хотя формально «задается». Файл «истории охлаждения» содержит 15 измеренных значений с шагом 5

секунд. Сделаем грубую прикидку расхода времени на передачу этих данных из МК в панель.

Табл. Структура записи «истории нагрева»

Имя	Pd	Freq	Ia	Id	Tt	Pset	Iset	Tset	Ud	Uc
Адрес	0	25	3	2	8	24	26	27	1	10

8 - 3 = 5 допустимый интервал, 24 - 10 = 14 – недопустимый, формально требуется два запроса. Первый – на ячейки с 0 по 8, средний расход времени (8 + 3) / 1920 + 0.033 + (3 + 9*2 + 5) / 1920 = 0.0193 + 0.033 = 0.0523 с. Второй – на ячейки с 24 по 27, средний расход времени (8 + 3) / 1920 + 0.033 + (3 + 4*2 + 5) / 1920 = 0.0141 + 0.033 = 0.0471 с. Два запроса требуют от двух до четырех периодов 30 Гц, то есть сложно выполнить 10 парных запросов за секунду даже в отсутствии других запросов.

Табл. Структура записи «истории охлаждения»

Имя	RT1	RT2	RT3	RT4	RT5	RT6	RT7	RT8	RT9	RTg1	RTg2	WS1	WS2	WS3	WS4
Адрес	6	4	5	12	11	7	15	14	13	28	29	16	17	18	19

11 – 7 = 4 – допустимый интервал, 28 – 19 = 9 – недопустимый, также два запроса.

Реально кроме этих переменных с разной периодичностью опрашиваются еще от 4 до 12 шестнадцатибитных регистров и не мене 3 бит.

Для экспериментальной оценки состава запросов и временных параметров обмена данными были сняты осциллограммы сигналов на входе приемника Rx (канал 1) и выходе передатчика Tx (канал 2). Генератор ТГИ находился в состоянии «Нагрев» («без силы»), «Основной экран». На всех диаграммах масштаб по напряжению – 2 В/дел, по времени врьируется от 100 мс/дел до 500 мкс/дел.

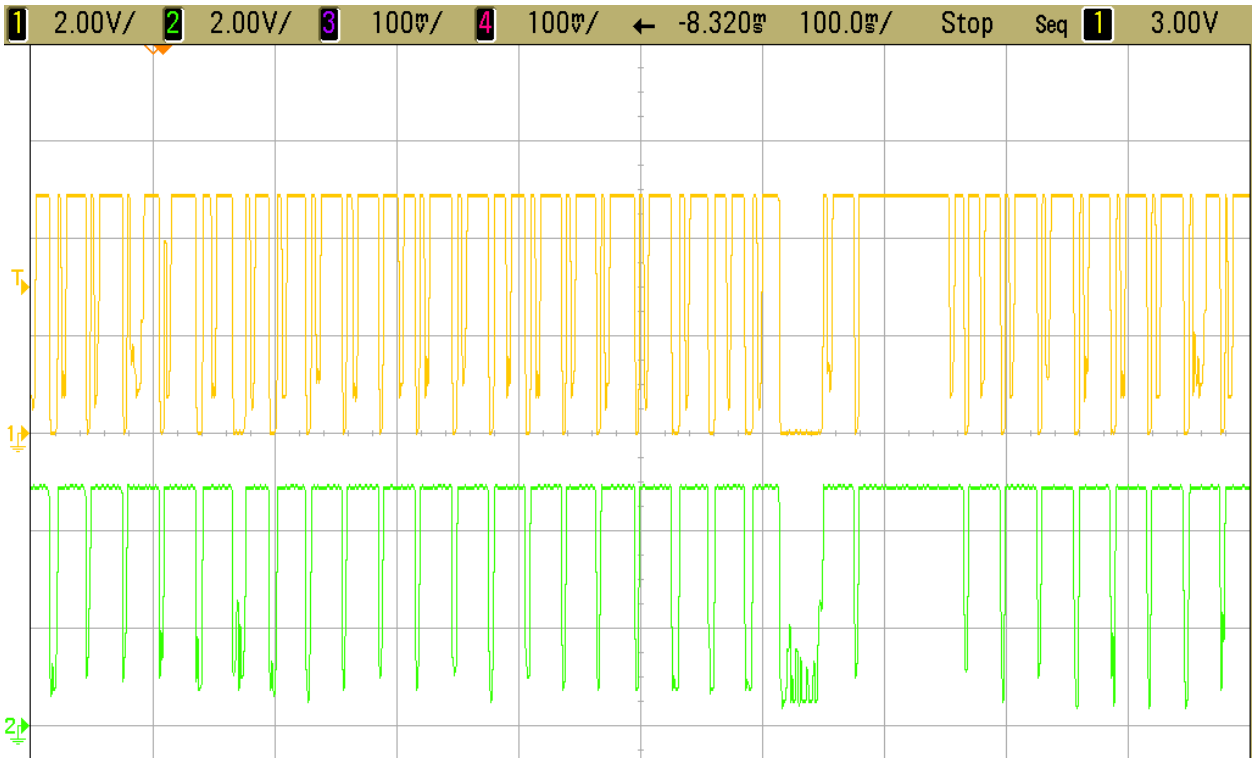


Рис. 7. 29

На рис. 7.29 прведена диграмма обзора одной секунды, число запросов-ответов 27-30 за секунду, наибольшая частота – 7 за 0.2 с (35 Гц), пауза между запросом и ответом (18-20 мс) больше паузы между ответом и очередным запросом (3-5 мс).

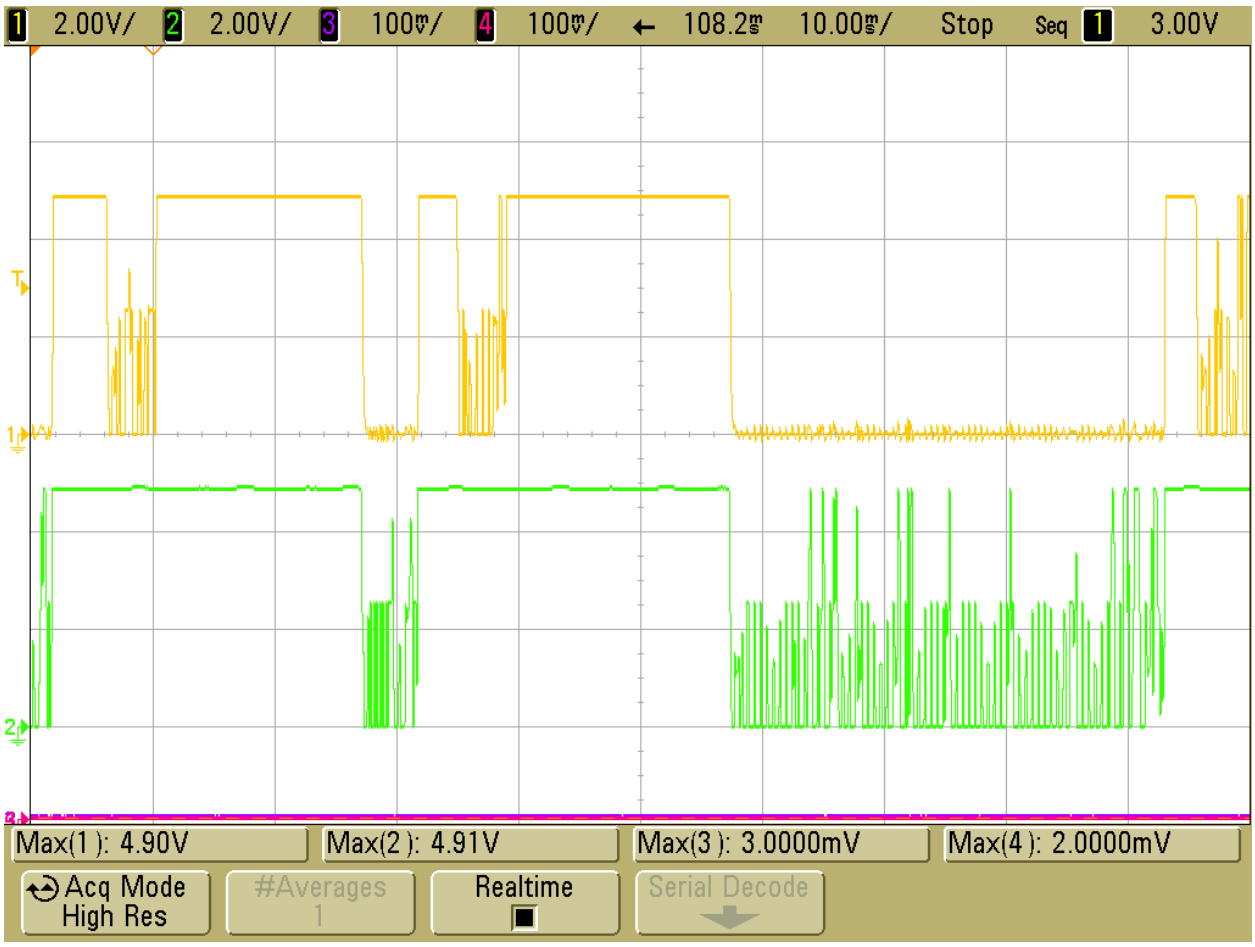


Рис. 7. 30 «Запрос- Ответ» 2 раза

Масщтаб 10 мс позволяет видеть детальнее запросы и ответы (рис. 7.30), в первой клетке виден запрос (примерно 4 мс), затем через 17 мс ответ (5 мс), через 3 мс новый запрос, через 18 мс ответ (35 мс). По длительности второго ответа можно понять, что это ответ на команду чтения нескольких слов.

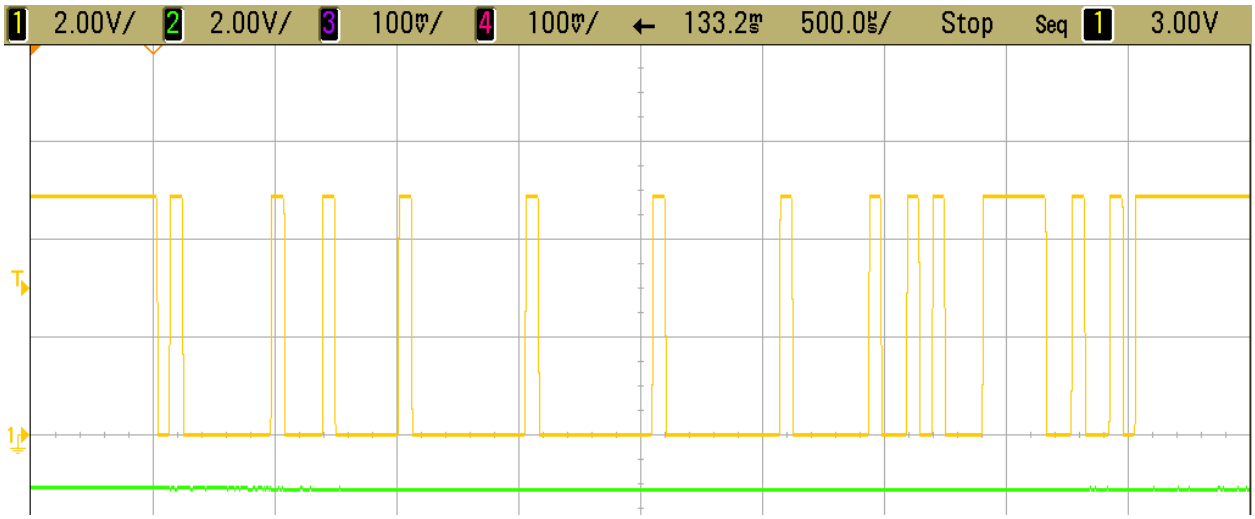


Рис. 7. 31 «Запрос»

Масштаб 5 мс в клетке (рис. 7.31, 7.32) позволяет «расшифровать» число передаваемых байт и их содержимое (по стоповому биту). Ниже представлен запрос с рис. 7.31 в двоичном виде; здесь 1’0 – последовательность Стоп-Старт, далее записаны 8 бит данных для удобства сгруппированние в тетрады:

1’0 1000 0000 1’0 0010 0000 1’0 0000 0000 1”0 0000 0000 1”0 0000 0000 1”0 0000 0100 1”0…

Два последних байта «расшифровывать» не обязательно, это контрольная сумма CRC16. Перепишем запрос в HEX-коде (опуская старт и стоп биты и приставку 0x):

01 04 00 00 00 20 CRC

Это соответсвует команде Read Input Register по адресу Adr = 0, число ячеек N=32.

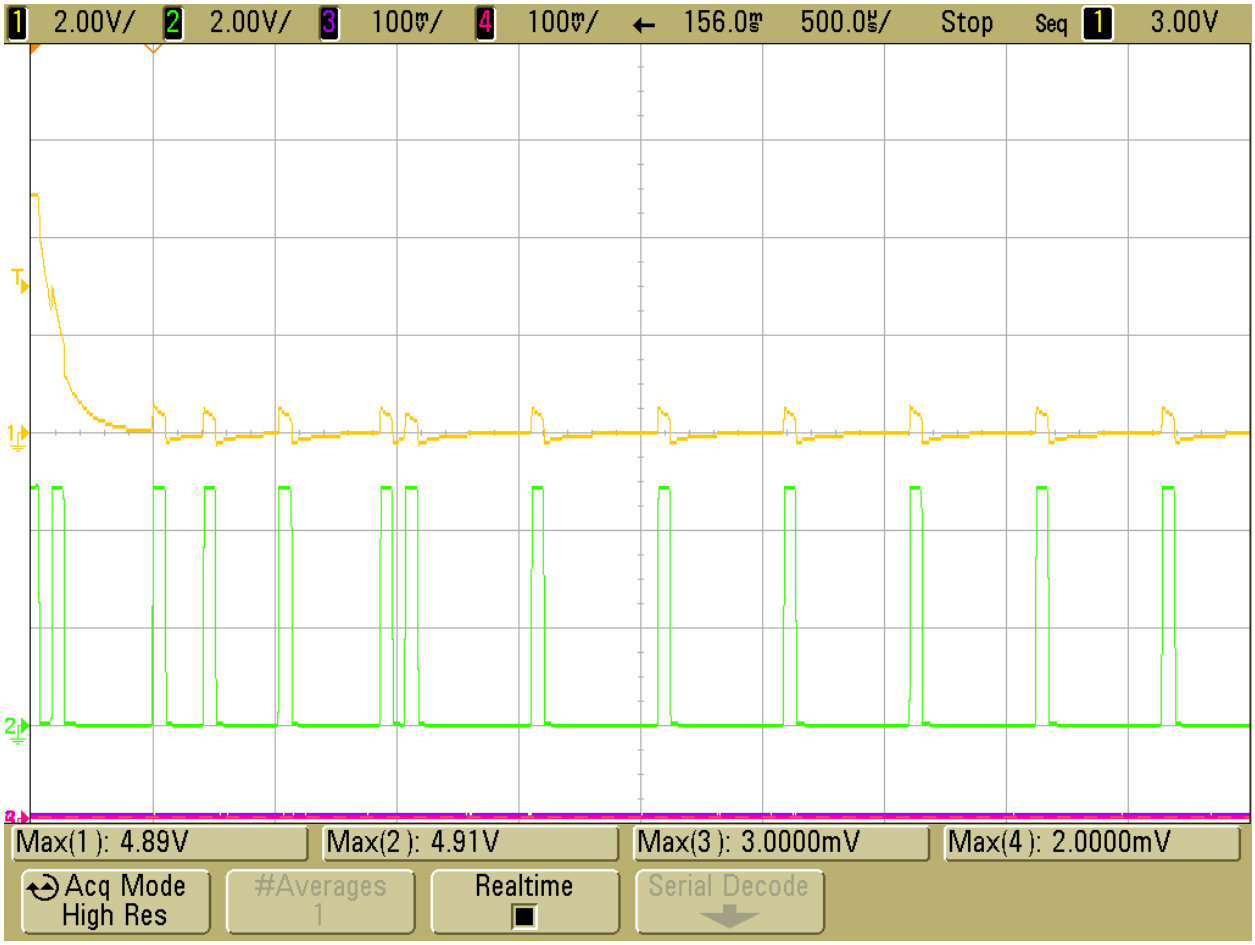


Рис. 7. 32 «Ответ»

Ниже приведена расшифровка ряда последовательных запросов и ответов (предшествующие номера обозначают имена файлов при съеме данных с осциллографа).

023: Rx 1 4 0 0 0 20 CRC (Read In reg Adr=0 N=32), 4 мс, 024: Tx 1 4 64 0 0 0 0 0 0 …, 35.5 мс

026: Rx 1 4 0 35 0 2 CRC (Read In reg Adr=53 N=2), 4 мс, 027: Tx 1 4 4 4 AA=170 0 0 CRC, 4.6 мс

028: Rx 1 4 0 3C 0 1 CRC (Read In reg Adr=60 N=1), 4.1 мс, 029: Tx 1 4 2 4 0 0 CRC, 3.6 мс

030: Rx 1 4 0 20 0 1 CRC (Read In reg Adr=32 N=1), 4.1 мс, 031: Tx 1 4 2 4 0 0 CRC, 3.6 мс

032: Rx 1 1 0 0 0 40 CRC (Read Coils Adr=0 N=64), 4.0 мс, 033: Tx 1 1 8 0 0 … CRC, 6,6 мс

035: Rx 1 4 0 43 0 1 CRC (Read In reg Adr=67 N=1), 4.1 мс, 036: Tx 1 4 2 0x92 0xB4 CRC, 3.6 мс

037: Rx 1 1 0 10 0 10 CRC (Read Coils Adr=16 N=16), 4.1 мс, 038: Tx 1 1 2 7 0 CRC, 3,5 мс

039: Обзор «длинного» запроса и короткого ответа (запись) Rx 11 мс, пауза 12 мс, Tx 6 мс, ниже - подробнее

041: Rx 1 10 0 EF 0 5 0x0C 0 … CRC (Write Regs Adr=0 N=64), 10.8 мс, 044: Tx 1 10 0 EF 0 5 CRC, 6,6 мс

045: Rx 1 1 0 10 0 30 CRC (Read Coils Adr=16 N=48), 4. мс, 046: Tx 1 1 6 7 0 10 0 10 0 CRC, 5,5 мс

048: Rx 1 1 0 0 0 40 CRC (Read Coils Adr=0 N=64), 4 мс, 049: Tx 1 1 8 0 0 7 0 10 0 10 CRC, 6,6 мс

Анализ состава запросов. Рассмотрено 10, из них чтение регистров – 5 (32 + 2 + 1 + 1 + 1 = 37 ячеек), чтение бит – 4 (64 + 16 + 48 + 64 = 192 бит), запись регистров – 1 (64 ячейки).

Контрольные вопросы по р. 6, 7

Теоретические

1. Дискретизация аналогового сигнала при вводе в МК и восстановление аналогового сигнала при выводе
2. Устройства аналогового ввода в МК.
3. Устройства аналогового вывода из МК.
4. Функции и технические характеристики встроенного аналогового компаратора 8-разрядных МК.
5. Структура и функции встроенного аналого-цифрового преобразователя 8-разрядных МК.
6. Технические характеристики встроенного аналого-цифрового преобразователя 8-разрядных МК.
7. Датчики температуры
8. Согласование термоэлетрического преобразователя (термопары) с АЦП
9. Согласование терморезистора с АЦП
10. Фильтрация при вводе аналогового сигнала температуры
11. Устройство и функции терморегулятора
12. Схема ЦАП на базе ШИМ-сигнала и его ограничения.
13. Предпосылки к замене параллельного интерфейса последовательным.
14. Принцип организации последовательного интерфейса синхронного типа и рекомендуемая область применения.
15. Принцип организации последовательного интерфейса асинхронного типа и рекомендуемая область применения.
16. Функции и узлы встроенного контроллера последовательного интерфейса.
17. Принцип организации и область применения однопроводного интерфейса
18. Технические характеристики встроенного контроллера последовательного интерфейса *SPI* 8-разрядных МК.
19. Технические характеристики встроенного контроллера последовательного интерфейса I^2C (*TWI*) 8-разрядных МК.
20. Технические характеристики встроенного контроллера последовательного интерфейса *U(S)ART* 8-разрядных МК.
21. Электрический стандарт RS-232 и драйверы лини связи.
22. Электрический стандарт RS-485 и драйверы лини связи.
23. Гальваническая развязка в последовательных каналах связи.
24. Открытый протокол Modbus – структура запросов и ответов.

Практические

1. Схема и алгоритм контроля разряда батареи питания с порогом срабатывания 3.0 В (компаратор и короткие вспышки светодиода).
2. Схема и алгоритм измерения и индикации температуры в диапазоне 500…1500 °С с дискретностью до 5 °С.
3. Схема и алгоритм ключевого терморегулятора паяльной станции мощностью 75 Вт с уставкой температуры 200…400 °С (шаг 2 °С).

4. Схема и алгоритм измерения частоты 48…52 Гц с точностью до 0.05 Гц в однофазной сети 220 В.
5. Схема и алгоритм измерения средней температуры объекта по 4 датчикам в диапазоне -40 … +40 ºС.
6. Схема и алгоритм периодического (T = 1 с) измерения (встроенный АЦП) и индикации температуры (0…999 °С, 3*7-сегментных светодиодных индикатора).
7. Схема и алгоритм приема данных (5 байт с примерным периодом 0,02 секунды) с программной реализацией протокола *SPI* в режиме ведомого.
8. Схема и алгоритм передачи данных (20 байт с примерным периодом 1 секунда) с программной реализацией протокола I^2C в режиме ведомого.
9. Схема и алгоритм передачи данных от 8 канального терморегулятора в персональный компьютер через *COM*-порт.
10. Схема и алгоритм измерения температуры 0…90 °С интегральным датчиком с двухпроводным интерфейсом (DS1621 или подобный).
11. Составить алгоритм для периодической (T = 1 мс) передачи 5 байт, ожидания не более 0.1 с и приема 2 байт с использованием контроллера последовательного интерфейса *U(S)ART*, выбрать скорость связи, МК и частоту тактирования f_{CLK}.
12. Разработать протокол сбора данных (5 двухбайтных параметров) в пульт управления с локальных регуляторов (до 16) в формате *ASCII* в полудуплексе. Формат: запрос – ответ, команда установления связи, отдельный запрос на каждый параметр.
13. Используя протокол Modbus-RTU разработать структуру запросов и ответов для конфигурирования (10 двухбайтных параметров) и опроса (два двухбайтных параметра) 16 терморегуляторов
14. Схема обмена данными пульта управления с 5 локальными регуляторами с использованием контроллера *U(S)ART*, в стандарте *RS*-485 (полудуплекс).
15. Составить алгоритм передачи в режиме ведущего трех байт с использованием контроллера последовательного интерфейса I^2C на скорости 400 Кбод, адрес 0x52, оценить время выполнения, выбрать МК и частоту тактирования f_{CLK}.
16. Составить схему подключения и алгоритм периодического (T = 1 с) измерения температуры тремя цифровыми датчиками DS1621 с использованием контроллера последовательного интерфейса I^2C, выбрать МК и частоту тактирования f_{CLK}.
17. Схема подключения внешнего 12 разрядного ЦАП по последовательному интерфейсу SPI и алгоритм формирования периодического синусоидального сигнала с периодом 1000 Гц амплитудой 2.5 В, выбрать МК и частоту тактирования f_{CLK}.
18. Схема подключения внешнего 10 разрядного ЦАП по последовательному интерфейсу SPI и алгоритм формирования периодического треугольного сигнала с периодом 1000 Гц амплитудой 5 В, выбрать МК и частоту тактирования f_{CLK}.
19. Схема подключения внешнего 10 разрядного ЦАП по последовательному интерфейсу I2C и алгоритм формирования периодического линейно спадающего сигнала с периодом 100 Гц, выбрать МК и частоту тактирования f_{CLK}.
20. Временная диаграмма обмена данными МК AVR-8 с fclk = 16 МГц с внешним ЦАП AD5310 на максимальной скорости обмена. Оценить максимальную скорость изменения напряжения на выходе ЦАП.

21. Временная диаграмма обмена данными МК AVR-8 с fclk = 8 МГц с внешним ЦАП MCP4725 на максимальной скорости обмена. Оценить максимальную скорость изменения напряжения на выходе ЦАП.
22. Схема подключения внешнего двухканального ЦАП с SPI интерфейсом MCP4822 для управления двухкоординатным приводом и алгоритм посылки пары новых значений.

Список литературы

1. Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. – М.: Издательский дом «Додэка-XXI», 2004. – 288 с.
2. Евстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы “ATMEL”. – М.: Издательский дом «Додэка-XXI», 2002. – 288 с.
3. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы “ATMEL”. – М.: Издательский дом «Додэка-XXI», 2008. – 560 с.
4. Евстифеев А.В. Микроконтроллеры AVR семейства MEGA”. – М.: Издательский дом «Додэка-XXI», 2007. — 592 с
5. Болски М.И. Язык программирования Си. Справочник: пер. с англ. – М., «Радио и связь», 1988. – 96 с.
6. Голик С. Е. Микроконтроллеры: архитектура и программирование. Учеб. Пособие. СПб.: Изд-во СПбГЭТУ, 2006.
7. Трамперт В Измерение, управление и регулирование с помощью AVR микроконтроллеров (+CD- ROM) МК-пресс, 2007. -208 с. (RC ШИМ?)
8. Микропроцессорные устройства: Методические указания к практическим занятиям / Сост.: Д. Н. Бондаренко. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2007. 32 с
9. Гребнев В.В. Микроконтроллеры семейства AVR фирмы Atmel. – М.: ИП РадиоСофт, 2002. – 176 с.
10. PIC-микроконтроллеры. Полное руководство / С. Катцен; пер. с англ. Евстифеева А.В. - М. : ДОДЭКА-XXI, 2010. - 650 с.
11. Решение интерфейсных задач с использованием однокристальных микроконтроллеров: метод. указания к лаб. работам; сост.: А.В. Кекконен, А.А.Тимофеев. - СПб. : Изд-во СПбГЭТУ "ЛЭТИ", 2009. - 39 с.
12. Микросхемы АЦП и ЦАП. – М.: Издательский дом «Додэка-XXI», 2005. – 432 с. (Серия «Инегральные микросхемы»).

Приложения

Приложение 1. Системы счисления и форматы представления чисел

В информатике кроме привычной нам десятичной системы счисления широко применяются двоичная и шестнадцатеричная. Это связано с удобством представления чисел – данных и адресов, хранящихся в регистрах, ячейках памяти и передаваемых по проводникам.

Двоичная *[binary]* система счисления использует всего два значения – 0 (Ложь - *[False]*) и 1 (Истина - *[True]*). В информатике один двоичный разряд называется «бит» [bit] и является величиной неделимой. Эта система счисления удобна для представления сигналов в цифровой технике, где по одному проводу передаются сигналы всего двух уровней. Обычно сигнал низкого уровня напряжения (близкий к нулю питания) кодируется «0», сигнал высокого уровня (близкий к плюсу питания VCC) кодируется «1», это так называемая прямая логика. Иногда используется инверсная или обратная логика.

Шестнадцатеричная система счисления *[hexadecimal]* использует цифры от 0 до 9 и латинские символы A, B, C, D, E, F (либо a…f – регистр не имеет значения) для кодирования $2^4 = 16$ состояний. Эта система счисления гораздо более компактна, чем двоичная, одна шестнадцатеричная цифра преобразуется в четыре двоичных.

Восемь бит образуют один байт и позволяют различать $2^8 = 256$ состояний. Для их записи требуется ровно 8 двоичных цифр или ровно две шестнадцатеричных. Преобразование десятичного числа в формат двоичного или шестнадцатеричного требует нескольких операций деления, обратное преобразование несколько проще.

Как правило, наращивание разрядности данных и адресов идет кратно восьми битам или байтам, в этих случаях удобно использовать шестнадцатеричное представление.

Редакторы трансляторов, символьные отладчики и системы моделирования цифровых узлов придерживаются следующей нотации.

Числа в десятичном формате *[decimal]* записываются без дополнительных знаков или с завершающим символом (суффиксом) *d*, например, 357 = *357d*.

Числа в двоичном формате *[binary]* записываются с предшествующим суффиксом *0b*, например, $0b010011 = 1*2^5 + 1*2^2 + 1*2^0 = 16+2+1 = 19d = 19$.

Шестнадцатеричные числа записываются с предшествующим суффиксом *0x* или *$*, либо с последующим суффиксом *H(h)*, например, $0x13 = \$13 = 13h = 1*16^1 + 3*16^0 = 16 + 3 = 19d = 19$.

Для преобразования числа 341 в формат шестнадцатеричного:

1) 341/16 = 21 и остаток 341 – 16*21 = 341 – 336 = 5 – это самая младшая цифра ($*16^0$).

2) Так как, 21 > 16, повторяем предыдущие действия: 21/16 = 1 и остаток 5 – это значение следующего разряда.

3) Так как 1 ≤ 16, то 1 – значение старшего разряда, получили 341 = *0x155*. Это число занимает два байта, так как потребовалось 3 цифры для его представления.

Если требуется только определить число требуемых двоичных или шестнадцатеричных разрядов, достаточно выполнить операцию взятия логарифма с соответствующим основанием: $\log_2(4095) = 12$, $\log_{16}(4095) = 3$.

В 8-разрядных МК наиболее распространенными форматами адресов и данных

являются байт и слово, состоящее из двух байт. При работе с адресами чаще всего используют числа без знака. При измерении физических величин нередко требуются целые числа со знаком. Для представления отрицательных целых используют двоично-дополнительный формат. Единица в старшем разряде числа обозначает числа меньше нуля, оставшиеся разряды представляют модуль числа в дополнительном коде (инверсия плюс 1). Таблицы ниже показывают представление целых чисел без знака и со знаком в двоично-дополнительном коде в формате байта и слова.

Табл. П1

Bin	*Hex*	*dec*		*Bin*	*hex*	*dec*
Байт без знака				**Слово без знака**		
0b1111'1111	*0xFF*	255		*0b1111'1111'1111'1111*	*0xFFFF*	65535
0b1111'1110	*0xFE*	254		*0b1111'1111'1111'1110*	*0xFFFE*	65534
...	...	…		...	...	…
0b0000'0001	*0x01*	1		*0b0000'0000'0000'0001*	*0x0001*	1
0b0000'0000	*0x00*	0		*0b0000'0000'0000'0000*	*0x0000*	0
Байт со знаком				**Слово со знаком**		
0b0111'1111	*0x7F*	+127		*0b0111'1111'1111'1111*	*0x7FFF*	+32767
0b0111'1110	*0x7E*	+126		*0b0111'1111'1111'1110*	*0x7FFE*	+32766
...	...	…		...	...	…
0b0000'0001	*0x01*	+1		*0b0000'0000'0000'0001*	*0x0001*	+1
0b0000'0000	*0x00*	0		*0b0000'0000'0000'0000*	*0x0000*	0
0b1111'1111	*0xFF*	-1		*0b1111'1111'1111'1111*	*0xFFFF*	-1
...	...	…		...	...	…
0b1000'0001	*0x81*	-127		*0b1000'0000'0000'0001*	*0x8001*	-32767
0b1000'0000	*0x80*	-128		*0b1000'0000'0000'0000*	*0x8000*	-32768

Запись констант с выделением заданных двоичных разрядов. При работе с регистрами ввода/вывода можно применять различные тождественные приемы, например: (1<<7)|(1<<3)+(1<<1)|(1<<0) = *0x93 = 0b10001011* = $2^7 + 2^3 + 2^1 + 2^0$ = 128+8+2+1 = 139.

П2. Таблица ASCII кодов

ASCII — American Standard Code for Information Interchange — американский стандартный код для обмена информацией. ASCII представляет собой 8-битную кодировку для представления десятичных цифр, латинского и национального алфавитов, знаков препинания и управляющих символов. Нижнюю половину кодовой таблицы (0 — 127) занимают символы US-ASCII, а верхнюю (128 — 255) — разные другие нужные символы.

Основная таблица ASCII

	00	10	20	30	40	50	60	70
0		►		0	@	P	`	p
1	☺	◄	!	1	A	Q	a	q
2	☻	↕	"	2	B	R	b	r
3	♥	‼	#	3	C	S	c	s
4	♦	¶	$	4	D	T	d	t
5	♣	§	%	5	E	U	e	u
6	♠	▬	&	6	F	V	f	v
7	•	↨	'	7	G	W	g	w
8	◘	↑	(	8	H	X	h	x
9	○	↓	)	9	I	Y	i	y
A	◙	→	*	:	J	Z	j	z
B	♂	←	+	;	K	[	k	{
C	♀	∟	,	<	L	\	l	|
D	♪	↔	-	=	M	]	m	}
E	♫	▲	.	>	N	^	n	~
F	☼	▼	/	?	O	_	o	⌂

Расширенная таблица ASCII (cp866)

	80	90	A0	B0	C0	D0	E0	F0
0	А	Р	а	░	└	╨	р	≡
1	Б	С	б	▒	┴	╤	с	±
2	В	Т	в	▓	┬	╥	т	≥
3	Г	У	г	│	├	╙	у	≤
4	Д	Ф	д	┤	─	╘	ф	⌠
5	Е	Х	е	╡	┼	╒	х	⌡
6	Ж	Ц	ж	╢	╞	╓	ц	÷
7	З	Ч	з	╖	╟	╫	ч	≈
8	И	Ш	и	╕	╚	╪	ш	°
9	Й	Щ	й	╣	╔	┘	щ	∙
A	К	Ъ	к	║	╩	┌	ъ	·
B	Л	Ы	л	╗	╦	█	ы	√
C	М	Ь	м	╝	╠	▄	ь	ⁿ
D	Н	Э	н	╜	═	▌	э	²
E	О	Ю	о	╛	╬	▐	ю	■
F	П	Я	п	┐	╧	▀	я	

Код символа однобайтный, в верхней строке (0x00 – 0xF0) – старшая тетрада, в колонках (0x00 – 0x0F) – младшая тетрада.

Приложение 3. Система команд МК *AVR*

Мнем.	Операнд	Описание	Действия	Флаги	цикл	слов	Двоичный код команды
АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ КОМАНДЫ							
ADD	Rd, Rr	Add without Carry two Registers	Rd ← Rd + Rr	Z,C,N,V,H,S	1	1	0000 11rd dddd rrrr
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H,S	1	1	0001 11rd dddd rrrr
ADIW	Rdl, k6	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl+k6	Z,C,N,V,S	2	1	1001 0110 kkdd kkkk
SUB	Rd, Rr	Subtract without Carry two Registers	Rd ← Rd - Rr	Z,C,N,V,H,S	1	1	0001 10rd dddd rrrr
SUBI	Rd*, k8	Subtract Constant from Register	Rd* ← Rd* - k8,	Z,C,N,V,H,S	1	1	0101 kkkk dddd kkkk
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H,S	1	1	0000 10rd dddd rrrr
SBCI	Rd*, k8	Subtract with Carry Constant from Register	Rd ← Rd – k8 - C	Z,C,N,V,H,S	1	1	0100 kkkk dddd kkkk
SBIW	Rdl, k6	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl – k6	Z,C,N,V,S	2	1	1001 0111 kkdd kkkk
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V,S	1	1	0010 00rd dddd rrrr
ANDI	Rd*, k8	Logical AND Register and Constant	Rd* ← Rd* • k8	Z,N,V,S	1	1	0111 kkkk dddd kkkk
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V,S	1	1	0010 10rd dddd rrrr
ORI	Rd*, k8	Logical OR Register and Constant	Rd* ← Rd* v k8	Z,N,V,S	1	1	0110 kkkk dddd kkkk
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V,S	1	1	0010 01rd dddd rrrr
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V,H,S	1	1	1001 010d dddd 0000
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1	1	1001 010d dddd 0001
SBR	Rd*,k8	Set Bit(s) in Register	Rd* ← Rd* v k8	Z,C,N,V,S	1	1	0110 kkkk dddd kkkk
CBR	Rd*,k8	Clear Bit(s) in Register	Rd* ← Rd* • ($FF – k8)	Z,C,N,V,S	1	1	0111 kkkk dddd kkkk
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1	1	1001 010d dddd 0011
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V,S	1	1	1001 010d dddd 1010
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,C,N,V,S	1	1	0010 00dd dddd dddd
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V,S	1	1	0010 01dd dddd dddd
SER	Rd*	Set Register	Rd* ← $FF	None	1	1	1110 1111 dddd 1111
КОМАНДЫ ВЕТВЛЕНИЯ							
RJMP	ra12	Relative Jump	PC ← PC + ra12 + 1	None	2	1	1100 aaaa aaaa aaaa
IJMP		Indirect Jump to (Z)	PC ← Z	None	2	1	1001 0100 0000 1001
JMP	a16	Jump	PC ← a16	None	3	2	1001 010a aaaa 110a aaaa aaaa aaaa aaaa
RCALL	ra12	Relative Subroutine Call	(SP)←PC, SP←SP-2/3, PC ← PC + ra12 + 1	None	3/4	1	1101 aaaa aaaa aaaa
CALL	a16	Call Subroutine	(SP)←PC, SP←SP-2/3, PC ← a16/22	None	4/5	2	1001 010a aaaa 111a aaaa aaaa aaaa aaaa
ICALL		Indirect Call to (Z)	(SP)←PC, SP←SP-2/3, PC ← Z	None	3/4	1	1001 0101 0000 1001
RET		Subroutine Return	SP←SP+2/3, PC ← (SP)	None	4/5	1	1001 0101 0000 1000
RETI		Interrupt Return	SP←SP+2/3, PC ← (SP)	I	4/5	1	1001 0101 0001 1000
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd == Rr) PC ← PC + 2/3	None	1/2/3	1	0001 00rd dddd rrrr
CP	Rd, Rr	Compare	Rd – Rr	Z,N,V,C,H,S	1	1	0001 01rd dddd rrrr
CPC	Rd, Rr	Compare with Carry	Rd – Rr – C	Z,N,V,C,H,S	1	1	0000 01rd dddd rrrr
CPI	Rd*, k8	Compare Register with Immediate	Rd* – k8	Z,N,V,C,H,S	1	1	0011 kkkk dddd kkkk
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)==0) PC ← PC + 2/3	None	1/2/3	1	1111 110r rrrr 0bbb
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)==1) PC ← PC + 2/3	None	1/2/3	1	1111 111r rrrr 0bbb
SBIC	P*, b	Skip if Bit in I/o register Cleared	if (P*(b)==0) PC ← PC + 2/3	None	1/2/3	1	1001 1001 pppp pbbb
SBIS	P*, b	Skip if Bit in I/o register is Set	if (P*(b)==1) PC ← PC + 2/3	None	1/2/3	1	1001 1011 pppp pbbb
BRBS	s, ra7	BRanch if status flag Set	if (SREG(s)==1) PC ← PC + ra7 + 1	None	1/2	1	1111 00aa aaaa asss
BRBC	s, ra7	BRanch if status flag Cleared	if (SREG(s)==0) PC ← PC + ra7 + 1	None	1/2	1	1111 01aa aaaa asss
BREQ	ra7	BRanch if EQual	if (Z==1) PC ← PC + ra7 +1	None	1/2	1	1111 00aa aaaa a001
BRCS	ra7	BRanch if Carry Set	if (C==1) PC ← PC + ra7 +1	None	1/2	1	1111 00aa aaaa a000
BRNE	ra7	BRanch if Not Equal	if (Z==0) PC ← PC + ra7 +1	None	1/2	1	1111 01aa aaaa a001
BRCC	ra7	BRanch if Carry Cleared	if (C==0) PC ← PC + ra7 +1	None	1/2	1	1111 01aa aaaa a000
BRSH	ra7	BRanch if Same or Higher	if (C==0) PC ← PC + ra7 +1	None	1/2	1	1111 01aa aaaa a000
BRLO	ra7	BRanch if LOwer, unsigned	if (C==1) PC ← PC + ra7 +1	None	1/2	1	1111 00aa aaaa a000
BRMI	ra7	BRanch if MInus	if (N==1) PC ← PC + ra7 +1	None	1/2	1	1111 00aa aaaa a010
BRPL	ra7	BRanch if PLus	if (N==0) PC ← PC + ra7 +1	None	1/2	1	1111 01aa aaaa a010
BRGE	ra7	BRanch if Greater or Equal, signed	if ((N ⊕ V)==0) PC ← PC + ra7 + 1	None	1/2	1	1111 01aa aaaa a100
BRLT	ra7	BRanch if Less Than, Signed	if (N ⊕ V==1) PC ← PC + ra7 + 1	None	1/2	1	1111 00aa aaaa a100
BRHS	ra7	BRanch if Half carry flag Set	if (H==1) PC ← PC + ra7 +1	None	1/2	1	1111 00aa aaaa a101
BRHC	ra7	BRanch if Half carry flag Cleared	if (H==0) PC ← PC + ra7 +1	None	1/2	1	1111 01aa aaaa a101
BRTS	ra7	BRanch if T flag Set	if (T==1) PC ← PC + ra7 +1	None	1/2	1	1111 00aa aaaa a110
BRTC	ra7	BRanch if T flag Cleared	if (T==0) PC ← PC + ra7 +1	None	1/2	1	1111 01aa aaaa a110
BRVS	ra7	BRanch if oVerflow flag is Set	if (V==1) PC ← PC + ra7 +1	None	1/2	1	1111 00aa aaaa a011
BRVC	ra7	BRanch if oVerflow flag is Cleared	if (V==0) PC ← PC + ra7 + 1	None	1/2	1	1111 01aa aaaa a011
BRIE	ra7	BRanch if Interrupt Enabled	if (I==1) PC ← PC + ra7 + 1	None	1/2	1	1111 00aa aaaa a111
BRID	ra7	BRanch if Interrupt Disabled	if (I==0) PC ← PC + ra7 + 1	None	1/2	1	1111 01aa aaaa a111
КОМАНДЫ ПЕРЕДАЧИ ДАННЫХ							
MOV	Rd, Rr	MOVe between registers	Rd ← Rr	None	1	1	0010 11rd dddd rrrr

LDI	Rd*, k8	LoaD Immediate	Rd ← k8	None	1	1	1110 kkkk dddd kkkk
LD	Rd, X	Load Indirect	Rd ← (X)	None	2	1	1001 000d dddd 1100
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2	1	1001 000d dddd 1101
LD	Rd, -X	Load Indirect and Pre-Dec.	X ← X – 1, Rd ← (X)	None	2	1	1001 000d dddd 1110
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2	1	1001 000d dddd 1000
LD	Rd, Y+	LDLoad Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2	1	1001 000d dddd 1001
LD	Rd, -Y	Load Indirect and Pre-Dec.	Y ← Y – 1, Rd ← (Y)	None	2	1	1001 000d dddd 1010
LDD	Rd,Y+ q6	Load Indirect with Displacement	Rd ← (Y + q6)	None	2	1	10q0 qq0d dddd 1qqq
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2	1	1001 000d dddd 0000
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2	1	1001 000d dddd 0001
LD	Rd, -Z	Load Indirect and Pre-Dec	Z ← Z – 1, Rd ← (Z)	None	2	1	1001 000d dddd 0010
LDD	Rd, Z+q6	Load Indirect with Displacement	Rd ← (Z + q6)	None	2	1	10q0 qq0d dddd 0qqq
LDS	Rd, a16	Load Direct from SRAM	Rd ← (k16)	None	2	2	1001 000d dddd 0000 aaaa aaaa aaaa aaaa
ST	X, Rr	Store Indirect	(X) ← Rr	None	2	1	1001 001r rrrr 1100
ST	X+, Rr	Store Indirect and Post-Inc.	ST (X) ← Rr, X ← X + 1	None	2	1	1001 001r rrrr 1101
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X – 1, (X) ← Rr	None	2	1	1001 001r rrrr 1110
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2	1	1001 001r rrrr 1000
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2	1	1001 001r rrrr 1001
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y – 1, (Y) ← Rr	None	2	1	1001 001r rrrr 1010
STD	Y+q6,Rr	Store Indirect with Displacement	(Y + q6) ← Rr	None	2	1	10q0 qq1r rrrr 1qqq
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2	1	1001 001r rrrr 0000
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2	1	1001 001r rrrr 0001
ST	-Z, Rr	Store Indirect and Pre-Dec	Z ← Z – 1, (Z) ← Rr	None	2	1	1001 001r rrrr 0010
STD	Z+q6,Rr	Store Indirect with Displacement	(Z + q6) ← Rr	None	2	1	10q0 qq1r rrrr 0qqq
STS	a16, Rr	Store Direct to SRAM	(a16) ← Rr	None	2	2	1001 001r rrrr 0000 aaaa aaaa aaaa aaaa
LPM		Load Program Memory	R0 ← (Z)	None	3	1	1001 0101 1100 1000
IN	Rd, P	In Port	Rd ← P	None	1	1	1011 0ppd dddd pppp
OUT	P, Rr	Out Port	P ← Rr	None	1	1	1011 1ppr rrrr pppp
PUSH	Rr	Push Register on Stack	(SP) ← Rr, SP – 1 →SP	None	2	1	1001 001r rrrr 1111
POP	Rd	Pop Register from Stack	SP + 1 →SP, Rd ← (SP)	None	2	1	1001 000d dddd 1111
КОМАНДЫ РАБОТЫ С БИТАМИ							
SBI	P*,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2	1	1001 1010 pppp pbbb
CBI	P*,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2	1	1001 1000 pppp pbbb
LSL	Rd	Logical Shift Left (экв. ADD Rd, Rd)	C←Rd(7), Rd(n+1)←Rd(n) n=6…0, Rd(0)←0	Z,C,N,V,S	1	1	1000 11dd dddd dddd
LSR	Rd	Logical Shift Right	C←Rd(0), Rd(n) ← Rd(n+1) n=0…6, Rd(7)←0	Z,C,N,V,S	1	1	1001 010d dddd 0110
ROL	Rd	Rotate Left Through Carry (экв. ADC Rd, Rd)	C←Rd(7), Rd(n+1)←Rd(n) n=6…0, Rd(0)←C,	Z,C,N,V	1	1	0001 11dd dddd dddd
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1), C←Rd(0)	Z,C,N,V	1	1	1001 010d dddd 0111
ASR	Rd	Arithmetic Shift Right	C←Rd(0), Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1	1	1001 010d dddd 0101
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4), Rd(7..4)←Rd(3..0)	None	1	1	1001 010d dddd 0010
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1	1	1111 100d dddd 0bbb
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1	1	1111 101d dddd 0bbb
BSET	S	Flag Set	SREG(s) ← 1 SREG(s)		1	1	1001 0100 0sss 1000
BCLR	S	Flag Clear	SREG(s) ← 0 SREG(s)		1	1	1001 0100 1sss 1000
SEC		Set Carry	C ← 1	C	1	1	1001 0100 0000 1000
CLC		Clear Carry	C ← 0	C	1	1	1001 0100 1000 1000
SEN		Set Negative Flag	N ← 1	N	1	1	1001 0100 0010 1000
CLN		Clear Negative Flag	N ← 0	N	1	1	1001 0100 1010 1000
SEZ		Set Zero Flag	Z ← 1	Z	1	1	1001 0100 0001 1000
CLZ		Clear Zero Flag	Z ← 0	Z	1	1	1001 0100 1001 1000
SEI		Global Interrupt Enable	I ← 1	I	1	1	1001 0100 0111 1000
CLI		Global Interrupt Disable	I ← 0	I	1	1	1001 0100 1111 1000
SES		Set Signed Test Flag	S ← 1	S	1	1	1001 0100 0100 1000
CLS		Clear Signed Test Flag	S ← 0	S	1	1	1001 0100 1100 1000
SEV		Set Twos Complement Overflow	V ← 1	V	1	1	1001 0100 0011 1000
CLV		Clear Twos Complement Overflow	V ← 0	V	1	1	1001 0100 1011 1000
SET		Set T in SREG	T ← 1	T	1	1	1001 0100 0110 1000
CLT		Clear T in SREG	T ← 0	T	1	1	1001 0100 1110 1000
SEH		Set Half Carry Flag in SREG	H ← 1	H	1	1	1001 0100 0101 1000
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1	1	1001 0100 1101 1000
NOP		No Operation	None		1	1	0000 0000 0000 0000
SLEEP		Sleep (see specific description)	None		3	1	1001 0101 1000 1000
WDR		Watchdog Reset	None		1		1001 0101 1010 1000

Примечания:

Rd - регистр-приемник результата, $0 \le d \le 31$

Rr - регистр-источник

Rd* - регистр-приемник результата, $16 \le d \le 31$

Rd+1:Rd – двухбайтный регистр-приемник, d = 24, 26, 28, 30

k6 – 6-битная численная константа (0…64)

k8 – 8-битная численная константа (0…255)

a16 – 16-битная абсолют. адресная константа (0…65535)

ra7 – 7-битная относит. адресная константа (-64…+63)

ra12 – 12-битная относит. адресная константа (-2048 …2047)

b - номер бита в регистре (3 бита, 0…7)

s - номер бита в регистре статуса (3 бита, 0…7)

X, Y, Z - регистры косвенной адресации (X=R27:R26, Y=R29:R28; Z=R31:R30)

P - адрес регистра ввода/вывода (0…63)

P* - адрес регистра ввода/вывода (0…31)

q6 – смещение (0…63)

Приложение 4. Язык ассемблер и директивы транслятора МК *AVR*

Синтаксис строки на языке ассемблер рассмотрен в разделе 2.3 данного пособия, система команд – в разделах 1.3…1.5, сводная таблица мнемоник – в Приложении 3. Здесь приведено краткое рассмотрение директив и выражений транслятора с ориентацией на транслятор из среды *AVR-Studio* (подробнее – смотрите встроенный *Help*).

Директивы синтаксически выделяются открывающей точкой, они управляют работой транслятора и не порождают кодов машинных команд (в отличии от мнемоник инструкций).

Группа директив, определяющих тип сегмента исходного текста включает директивы *.CSEG* (*Code SEGment* – то есть раздел команд), *.ESEG* (*EEPROM SEGment* – то есть раздел описания данных в энергонезависимой памяти *EEPROM*), *.DSEG* (*Data SEGment* – то есть раздел описания переменных в энергозависимой памяти *SRAM*). Директивы не имеют параметров, область действия – строки ниже директивы, до следующей директивы данной группы. При отсутствии этих директив текущий раздел считается разделом команд (*Code SEGment).*

Группа директив, распределяющих память микроконтроллера включает директивы *.ORG, .DB, .DW, .BYTE*. Директива *.ORG* (*Set program origin*) устанавливает новое значение счетчика адреса в текущем сегменте памяти, ее операнд – константа, являющаяся адресом. Директивы *.DB* (*Data Byte*), *.DW* (*Data Word*) служат для размещения констант в памяти (в областях *EEPROM* и *FlashROM*), операндами служат значения констант, записанных через запятую. Как правило, директиве предшествует метка, позволяющая задать символическое значения адреса данной константы (или массива констант). Например, *CONST1: .DW 0, 1,1, 0* размещает числа 0, 1, 1, 0 в памяти программ начиная с текущего адреса *CONST1* побайтно (последнее число – по адресу *CONST1+6*. Другой пример:

```
.ESEG               ; Сегмент EEPROM
.ORG 0x20           ; По адресу CONST2 = 0x20
CONST2:    .DB 0x80; Разместить число 0x80
```

Директива *.BYTE* резервирует заданное число байт в области памяти данных (в SRAM или EEPROM), но не влияет на их содержимое. Пример:

```
.DSEG
var1:    .BYTE 1            ; резервирует 1 байт для переменной var1
table:   .BYTE tab_size     ; резервирует tab_size байт
.CSEG
         ldi r30,low(var1)  ; Загружает нижнюю часть адреса в Z регистр
         ldi r31,high(var1) ; Загружает верхнюю часть адреса в Z регистр
         ld r1,Z            ; Загружает значение переменной VAR1 в регистр R1
```

Группа директив, присваивающих имена и значения символическим именам включает директивы *.DEF, .EQU, .SET*. Директива *.DEF counter=R16* задает символическое имя *counter* регистру общего назначения *R16*. Директива *.EQU label = expression* задает метке *label* единственное численное константное значение *expression*. Директива *.SET label = expression* почти идентична директиве *.EQU* за тем лишь исключением, что может в одной сборке текста несколько раз присваивать метке разные значения; естественно, реально будет иметь эффект только последнее присвоение.

Директива *.DEVICE* <номер_МК> настраивает транслятор на систему команд конкретной модели МК (располагается в начале сборки).

Директива *.INCLUDE <file_name>* указывает транслятору имя файла, текст которого должен транслироваться в данный момент, с возвращением в текущий файл; допускается вложение. В среде *AVR-Studio* по адресу *..\AVR Tools\AvrAssembler\Appnotes* находятся готовые файлы с именем типа <номерМК*_def.inc*>, в них находятся описания символьных имен регистров данного МК, а также директива *.device* настройки на систему команд (например, *AT90S2313*).

Директива *.EXIT* прекращает процесс трансляции данного файла.

Директивы *MACRO* и *.ENDMACRO* служат для поддержки мощного механизма макрокоманд (название макроассемблер в свое время использовалось для его рекламы). Его основная идея состоит в том, что повторяющиеся куски текста программы заменяются макрокомандой с уникальным именем, тем самым улучшается читабельность текста и снижаются затраты на отладку. Механизм макрокоманд конкурирует с механизмом подпрограмм, обсуждение этого вопроса см. в разделе 2.3. Директива *.MACRO* открывает начало вставляемого куска, обязательным операндом идет уникальное имя, присваиваемое данному куску. Директива *.ENDMACRO* завершает описание куска текста программы, операндов не имеет. Строки текста между директивами называются текстом макрокоманды, он может содержать в местах расположения операндов команд формальные параметры, обозначаемые @0, @1 и т.д., которые заменяются реальными описаниями операндов в макровызове – имя макрокоманды с перечислением через запятую фактических значений операндов при выполнении транслятором макроподстановки.

В процессе отладки объемных программ ранее широко использовались так называемые файлы листинга. Они содержали, кроме строк исходного текста колонку адресов текущего сегмента памяти и колонку машинных кодов, порожденных в данной строке исходного текста (в *hex*-коде). Директивы *.LIST* и *.NOLIST* разрешают и прекращают вывод текста в файл листинга, директива *.LISTMAC* разрешает вывод текста макроподстановки.

Расширенная версия ассемблера (AVR Assembler 2), ориентированная на объемные и сложные проекты, выполняемые на ассемблере, имеет в дополнение группу директив условной трансляции, подобную одноименному механизму в языке C/C++.

Выражения используются в поле описания операнда при описании констант, вычисляемых на этапе трансляции, удобны для повышения читабельности текста программы и переносимости текста на другие МК или адаптации к изменяемым параметрам проекта.

Для описания констант кроме собственно численных значений используют:

- символические имена, ранее заданные пользователем с помощью директив, например, адреса ячеек памяти и численные значения констант;
- операторы (арифметические, логические, сдвига, сравнения), например, в выражении (1<<4)|(1<<1) = 0b00010010 использованы два оператора сдвига влево ‘<<’, скобки и оператор побитового «ИЛИ» ‘|’;
- выражения, например, LOW(ЧИСЛО) и HIGH(ЧИСЛО) вычисляют значение младшего и старшего байта ЧИСЛА разрядностью два байта.

Приложение 5. Технические характеристики МК *AVR*

Паспортные данные МК, как и любых других электронных компонент, состоят из таблицы предельных эксплуатационных параметров, таблицы электрических параметров, таблицы динамических (временных и частотных) параметров, разнообразных диаграмм (зависимостей параметров от напряжения питания, частоты тактирования, температуры кристалла и пр.), схемы расположения выводов в конкретном корпусе (цоколевка), тестовых схем и чертежей корпусов микросхемы.

Табл. П5.1 Максимально допустимые значения

Параметр	Предельные значения
Рабочая температура	–40...+85/105°C (модели ATtiny28x) –55...+125°C (остальные модели)
Температура хранения	–65...+150°C
Напряжение на любом выводе (кроме вывода $\overline{\text{RESET}}$) относительно вывода GND	–1.0...V_{CC} + 0.5 В
Напряжение на выводе $\overline{\text{RESET}}$ относительно вывода GND	–1.0...+13 В
Напряжение питания	6.0 В
Максимальный ток канала ввода/вывода	40 мА 60 мА (модели ATtiny28x, вывод PA2)
Максимальный ток выводов V_{CC} и GND	100 мА (модели ATtiny15L) 300 мА (модели ATtiny28x) 200 мА (остальные модели)

Табл. П5.1 Статические параметры

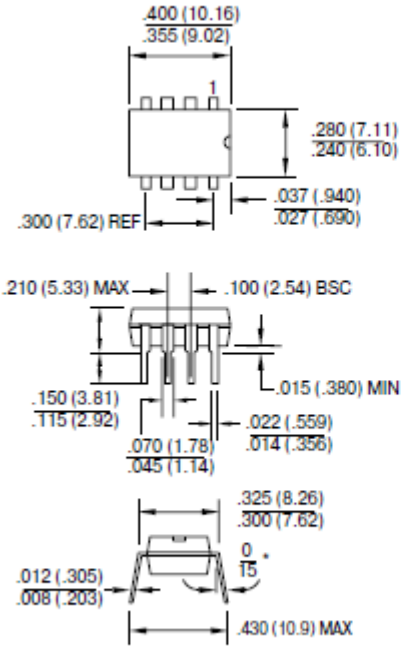
Параметр		Условия	Модель	min	typ	max	Ед. изм.
V_{IL}	Входное напряжение низкого уровня	Вывод XTAL1	Все модели	–0.5		$0.1V_{CC}$	В
		Остальные выводы	Все модели	–0.5		$0.3V_{CC}$	В
V_{IH}	Входное напряжение высокого уровня	Вывод XTAL1	ATtiny	$0.7V_{CC}$		V_{CC} + 0.5	В
			ATmega	$0.8V_{CC}$		V_{CC} + 0.5	
		Вывод $\overline{\text{RESET}}$	ATtiny	$0.85V_{CC}$		V_{CC} + 0.5	В
			ATmega	$0.9V_{CC}$		V_{CC} + 0.5	
		Остальные выводы	Все модели	$0.6V_{CC}$		V_{CC} + 0.5	В
V_{OL}	Выходное напряжение низкого уровня линий портов ввода/вывода	I_{OL} = 20 мА, V_{CC} = 5 В	Все модели			0.6	В
		I_{OL} = 10 мА, V_{CC} = 3 В	Все модели			0.5	
V_{OH}	Выходное напряжение высокого уровня линий портов ввода/вывода	I_{OH} = –3 мА, V_{CC} = 5 В	ATtiny, ATmega161/163/323	4.2			В
		I_{OH} = –20 мА, V_{CC} = 5 В	Остальные				
		I_{OH} = –1.5 мА, V_{CC} = 3 В	ATtiny, ATmega161/163/323	2.3			
		I_{OH} = –10 мА, V_{CC} = 3 В	Остальные				
		I_{OH} = –1 мА, V_{CC} = 2.5 В	AT90C8534	1.44			

Табл. П5.1 Продолжение

Параметр		Условия	Модель	min	typ	max	Ед. изм.
I_{IL}	Ток утечки на входе	V_{CC} = 5.5 В, на выводе – лог. «0» (абсолютное значение)	ATmega8515x			3	мкА
			Остальные			8	
I_{IH}	Ток утечки на входе	V_{CC} = 5.5 В, на выводе – лог. «1» (абсолютное значение)	ATmega8x			3	мкА
			Остальные			8	
RRST	Сопротивление подтягивающего резистора в цепи сброса		Все (кроме ATtiny)	100		500	кОм
$R_{I/O}$	Сопротивление подтягивающего резистора линии порта ввода/вывода		Все	35		120	кОм
		Рабочий режим, V_{CC} = 3 В, $f \leq 4$ МГц	Все модели			3.0	мА
		Рабочий режим, V_{CC} = 5 В, $f > 4$ МГц	ATiny11x/12x			10	
sI_{CC}	Ток потребления, семейство Tiny	Режим «Idle», V_{CC} = 3 В, $f \leq 4$ МГц	Все модели			1.2	мА
		Режим «Idle», V_{CC} = 5 В, $f > 4$ МГц	ATiny11x/12x			3.5	
		Режим «Power Down» WDT – вкл., V_{CC} = 3 В	Все модели		9.0	15.0	мкА
		Режим «Power Down» WDT – выкл., V_{CC} = 3 В	Все модели		<1.0	2.0	
	Ток потребления, семейство Mega	Рабочий режим, V_{CC} = 3 В, f = 4 МГц	Все модели			6.0	мА
		Рабочий режим, V_{CC} = 5 В, f = 8 МГц	Все модели			15.0	
		Режим «Idle», V_{CC} = 3 В, f = 4 МГц	Все модели			2.5	мА
		Режим «Idle», V_{CC} = 5 В, f = 8 МГц	Все модели (кроме ATmega161x)			8.0	
		Режим «Power Down» WDT – вкл., V_{CC} = 3 В	Все модели		<1.0	30.0	мкА
		Режим «Power Down» WDT – выкл., V_{CC} = 3 В	Все модели			8.0	

Табл. П5.2 Чертежи корпусов микроконтроллеров

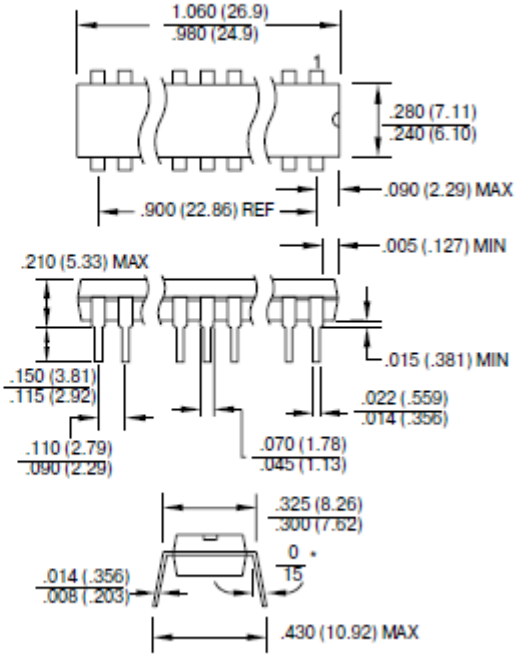
8-выводный пластиковый корпус с двурядным расположением выводов (PDIP-8)



* Размер для справок
Размеры в дюймах и миллиметрах

1

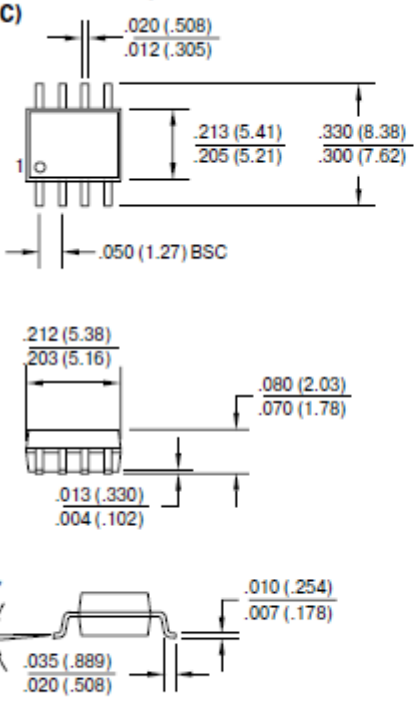
20-выводный пластиковый корпус с двурядным расположением выводов (PDIP-20)



* Размер для справок
Размеры в дюймах и миллиметрах

2

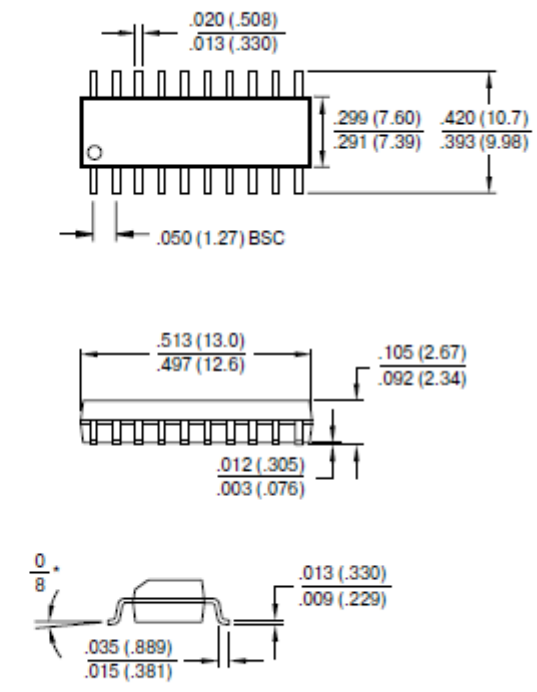
8-выводный SOIC корпус, ширина 0.20 in (EIAJ SOIC)



* Размер для справок
Размеры в дюймах и миллиметрах

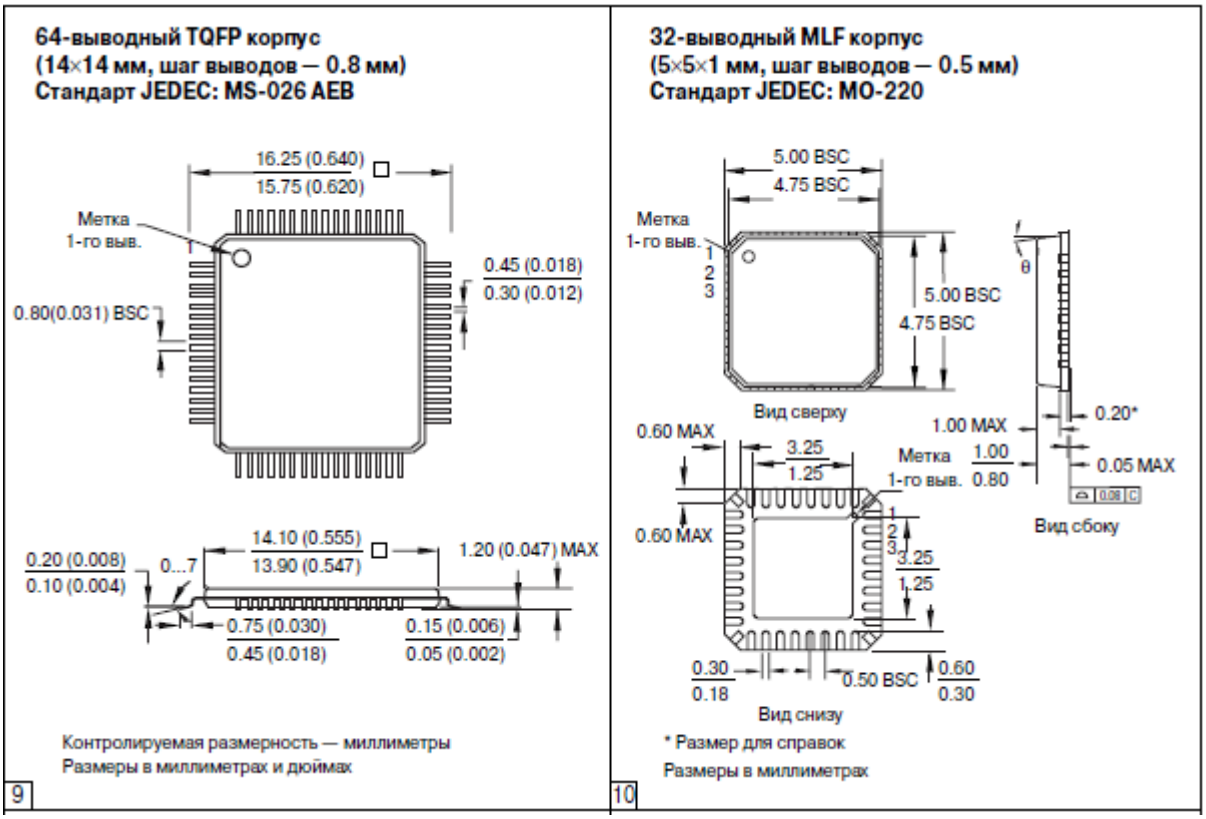
5

20-выводный SOIC корпус, ширина 0.300 in



* Размер для справок
Размеры в дюймах и миллиметрах

6



П6. Словарь аббревиатур и технических терминов

ВЛ(И) – вакуумно-люминесцентный (индикатор) *[Vacuum Fluorescent Display– VFD]*
ЖК(И) – жидко-кристаллический (индикатор) *[Liquid Crystal Display – LCD]*
МК – микроконтроллер *[Microcontroller - MC]*
ПД – память данных *[Data Memory - DM]*
ПЗУ – постоянное запоминающее устройство *[Read Only Memory - ROM]*
ПП – память программ *[Program Memory - PM]*
ППЗУ – программируемое постоянное запоминающее устройство *[Programmable Read Only Memory - PROM]*
П/П – подпрограмма *[Subroutine - SR]*
ППОП – подпрограмма обслуживания прерывания *[Interrupt Subroutine - ISR]*
РА – регистр адреса *[Address Register – AR]*
РД – регистр данных *[Data Register – DR]*
РОН – регистры общего назначения *[General Purpose Registers – GPR]*
СИД – светоизлучающий диод *[Light Emitting Diode - LED]*
СК – счетчик команд *[Program Counter – PC]*
СОЗУ – статическое оперативное запоминающее устройство *[Static Random Access Memory - SRAM]*
УС – указатель стека *[Stack Pointer – SP]*
ША – шина адреса *[Address Bus - AB]*
ШД – шина данных *[Data Bus - DB]*
ШУ – шина управления *[Control Bus - CB]*
ЭСППЗУ – электрически стираемое и программируемое постоянное запоминающее устройство *[Electrically Erasable Programmable Read Only Memory – EEPROM* и *FlashROM]*
IC – [Inegrated Circuit] – микросхема.

П7. Погрешность измерения (из Википедии)

Абсолютная погрешность — ΔX является оценкой абсолютной ошибки измерения. Величина этой погрешности зависит от способа её вычисления, который, в свою очередь, определяется распределением случайной величины X_{meas}. При этом неравенство:

$$\Delta X > | X_{meas} - X_{true} | ,$$

где X_{true} — истинное значение, а X_{meas} — измеренное значение, должно выполняться с некоторой вероятностью близкой к 1. Если случайная величина X_{meas} распределена по нормальному закону, то, обычно, за абсолютную погрешность принимают её среднеквадратичное отклонение. Абсолютная погрешность измеряется в тех же единицах измерения, что и сама величина.

Относительная погрешность — погрешность измерения, выраженная отношением абсолютной погрешности измерения к действительному или измеренному значению измеряемой величины:

$$\delta_X = \Delta X / X_{true}, \quad \delta_X = \Delta X / X_{meas},$$

Относительная погрешность является безразмерной величиной, либо измеряется в процентах.

Приведённая погрешность — погрешность, выраженная отношением абсолютной погрешности средства измерений к условно принятому значению величины, постоянному во всем диапазоне измерений или в части диапазона. Вычисляется по формуле

$$\delta_X = \Delta X / X_n, *100\ \%$$

где X_n — нормирующее значение, которое зависит от типа шкалы измерительного прибора и определяется по его градуировке:

— если шкала прибора односторонняя, то есть нижний предел измерений равен нулю, то X_n определяется равным верхнему пределу измерений;

— если шкала прибора двухсторонняя, то нормирующее значение равно ширине диапазона измерений прибора.

Приведённая погрешность является безразмерной величиной, либо измеряется в процентах.