ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ОЗЕРСКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ (филиал)
ГОУ ВПО «МОСКОВСКИЙ ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ
(государственный университет)»

Вл. Пономарёв

Организация и программирование учебного стенда SDK1.1

Учебное пособие по дисциплине
«Программирование микроконтроллеров»

Озерск, 2009

УДК 681.3.06
П 56

Пономарёв В.В. Организация и программирование учебного стенда SDK1.1. Учебное пособие по дисциплине «Программирование микроконтроллеров». Редакция 13.09.2009. Озерск: ОТИ МИФИ, 2009. — 136 с.,ил.

В пособии приведены краткие сведения о микроконтроллерах, описывается архитектура микроконвертера ADuC842-62, логика работы его таймеров, АЦП, ЦАП, ШИМ, интерфейсов UART и I^2C. Приведены сведения об организации учебного стенда SDK1.1, предназначенного для изучения программирования микроконтроллеров на основе архитектуры 8051/8052 (MCS-51/52). Даны рекомендации по программированию отдельных устройств стенда.
Пособие предназначено для студентов-программистов, обучающихся по специальности 230105 — «Программное обеспечение вычислительной техники и автоматизированных систем».

Рецензенты:
1. Доцент кафедры ВТ СПб ГУ ИТМО, к.т.н. А.О. Ключев
2. Ст. преподаватель кафедры СиА ОТИ МИФИ В.В. Шустов

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ МИФИ

Содержание

Введение

Настоящее учебное пособие предназначено для студентов специальности 230105 — «Программное обеспечение вычислительной техники и автоматизированных систем», изучающих дисциплину «Программирование микроконтроллеров» в качестве дополнительной в цикле дисциплин специализации. Для успешного освоения данной дисциплины студенты должны предварительно изучить дисциплины «Организация ЭВМ и систем», «Архитектура вычислительных систем», «Программирование на ассемблере» и «Электроника». В связи с этим изучение дисциплины «Программирование микроконтроллеров» выполняется на старших курсах.

Пособие ориентировано на изучение конкретного микроконтроллера — учебного стенда SDK1.1, поэтому материал пособия описывает устройство, архитектуру и программирование только этого стенда. Тем не менее, поскольку в основе учебного стенда лежит микроконвертер, построенный на основе расширенного ядра 8052, пособие косвенно отражает также основные особенности архитектуры и программирования этого ядра.

Пособие состоит из шести основных глав. Первая глава, «Вступление», кратко раскрывает предметную область с тем, чтобы сформировать общее представление о назначении и областях применения микроконтроллеров.

В главе «Общие сведения о микроконтроллерах» рассматриваются характерные черты микроконтроллеров, и предваряется детальное описание архитектуры и устройств учебного стенда SDK1.1.

Глава «Микроконвертер ADuC842» подробно описывает архитектуру и состав микроконвертера ADuC842 — организацию памяти, устройство, назначение и структуру таймеров, портов, интерфейсов, систему команд. В этой главе приведены все необходимые сведения для программирования многочисленных устройств микросхемы. Несмотря на справочный характер этой главы, без её материала невозможно начать программирование такого сложного прибора, каким является микроконвертер ADuC842.

В главе «Основы программирования ядра 8051/8052» приводятся основные сведения, необходимые для начала программирования на ассемблере микросхемы учебного стенда. Материал этой главы ориентирован на архитектуру ядер 8051/8052.

В главе «Программирование устройств SDK1.1» описываются внешние (по отношению к микроконвертеру) устройства учебного стенда, приводятся рекомендации по их программированию на ассемблере.

В главе «Организация работы» описывается последовательность разработки программ для учебного стенда и их доставки в микроконтроллер.

В главе «Симулятор SIMSDK11» описывается авторское программное обеспечение для отладки программ для учебного стенда SDK1.1.

В главе «Примерные темы практических заданий» приведены примеры заданий для выполнения практических работ.

Вступление

Термином «микроконтроллер» обозначают два различающихся понятия. Традиционно микроконтроллер — это функционально самостоятельный *прибор*, включающий в себя:

- микропроцессор,
- память,
- набор портов для прямой передачи сигналов,
- набор интерфейсов для организации связи с другими устройствами,
- средства управления в виде кнопок, переключателей и клавиатур,
- средства отображения информации в виде светодиодных и семисегментных индикаторов и знаковых или графических дисплеев, а также
- средства сопряжения с внешней средой в виде разъемов.

Все это размещается на одной монтажной плате и заключается в корпусе для обеспечения механической прочности прибора и удобства обращения с ним. Как правило, это небольшая по габаритам конструкция, имеющая размеры порядка 150×150×25 мм (рисунок 1).

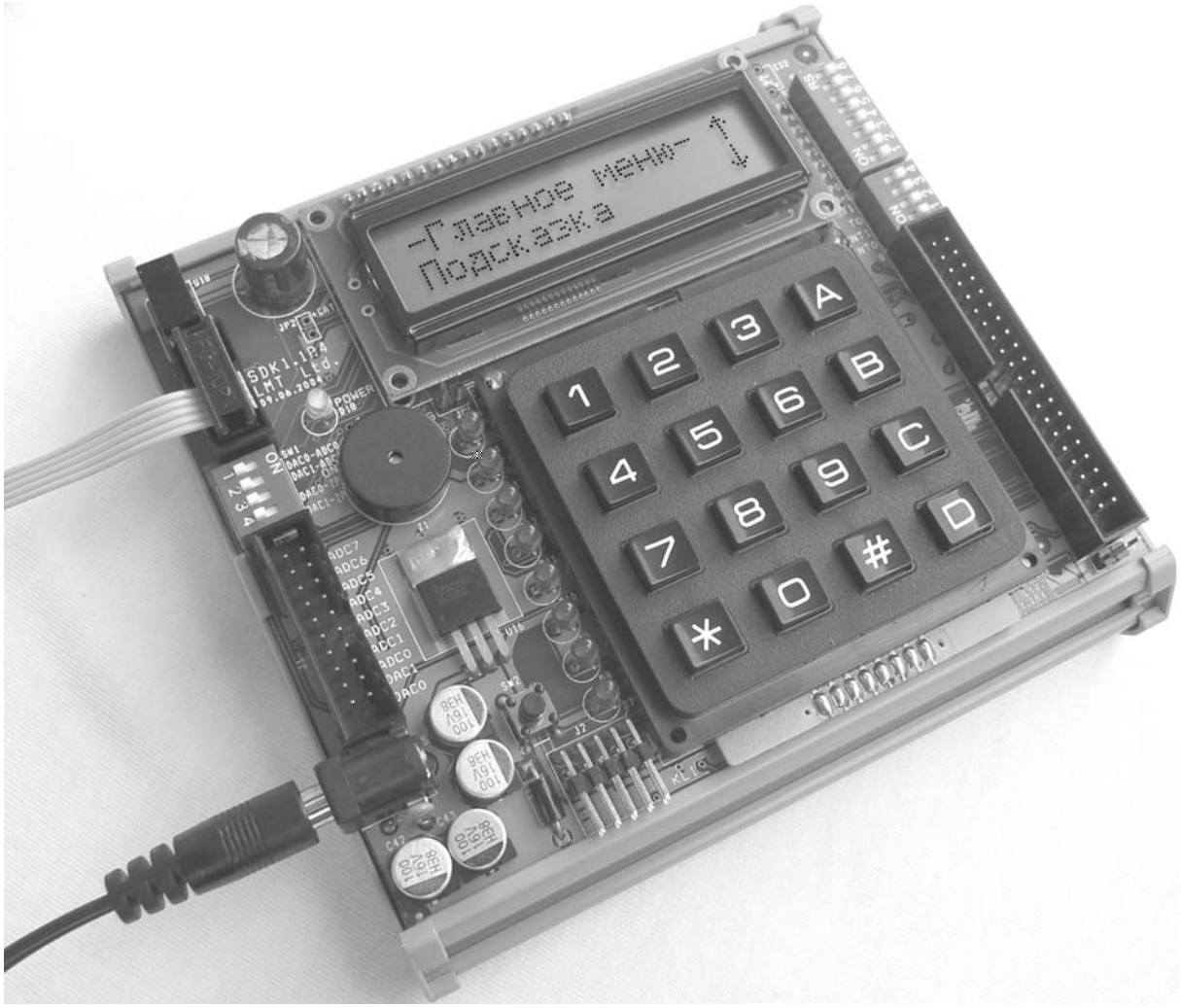


Рисунок 1. Учебный стенд (микроконтроллер) SDK1.1

С другой стороны, микроконтроллером называют также одну-единственную *микросхему*, по сути являющуюся микропроцессором, которая включает в себя собственно микропроцессор, память, порты, а также некоторые другие внутренние схемы, такие, как таймеры и интерфейсы.

С этой точки зрения микроконтроллер — функционально самостоятельная схема, которая в зависимости от программы, заложенной в ее память, может определенным образом принимать информацию по встроенным каналам связи, обрабатывать ее и передавать результаты обработки во внешнюю среду через те же каналы связи. Микроконтроллер-микросхема имеет размеры, как правило не превышающие 50 мм в наибольшем измерении (рисунок 2, справа). Например, микросхема учебного стенда SDK1.1 (рисунок 2, слева), имеет размер всего 10×10×2 мм (без учета ножек).

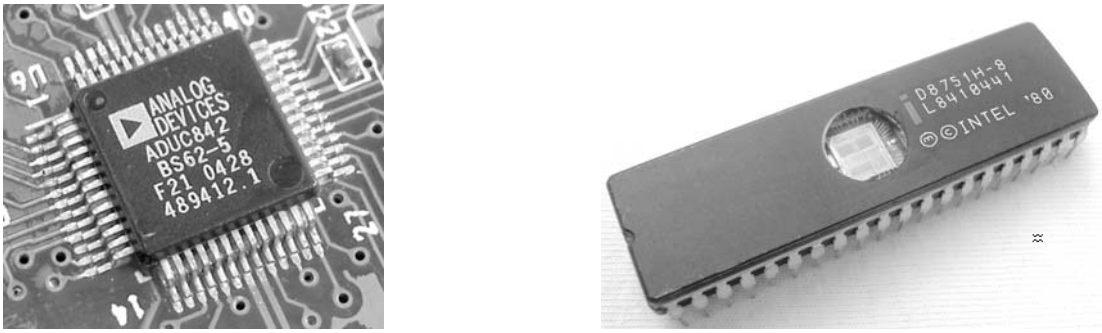


Рисунок 2. Микроконвертер ADuC842 и микроконтроллер 8751 (справа)

Некоторые микросхемы называют *микроконвертерами*. Микроконвертер отличается от обычного микроконтроллера наличием преобразователя аналогового сигнала в цифровой — аналого-цифровым преобразователем, сокращенно АЦП (*ADC*). Необходимость АЦП вытекает из назначения микроконтроллера работать с внешними сигналами и обрабатывать их. Во многих случаях внешние сигналы имеют аналоговый вид, а их обработка осуществляется в цифровой форме. Так как АЦП необходим практически каждому микроконтроллеру, его встраивают непосредственно в микросхему. Как правило, микроконвертеры включают в себя и цифро-аналоговые преобразователи, сокращенно ЦАП (*DAC*).

Развитие микроконтроллеров привело к появлению сигнальных процессоров и технологий цифровой обработки сигналов (*DSP — Digital Signal Processor, Processing*). *Сигнальный процессор* — это микропроцессор, предназначенный для обработки сигналов определенного вида в реальном времени. Основное отличие сигнального процессора от обычного микропроцессора заключается в оптимизации архитектуры для обеспечения быстрых вычислений в вещественных (а не целых) числах. Часто в сигнальный процессор встраивается реализация алгоритма какого-либо известного преобразования, используемого в обработке сигналов, например, быстрого преобразования Фурье. Сигнальные процессоры строятся на основе супергарвардской архитектуры (SHARC).

Современный микроконтроллер-прибор строится на основе микроконтроллера-микросхемы. Использовать микросхему саму по себе затруднительно по многим причинам. Во-первых, она слишком мала для того, чтобы обращаться с ней, как с устройством. Во-вторых, несмотря на самостоятельность, ей требуются хотя бы разъемы для подключения к источникам и приемникам информации, а также для получения питания. Не прикручивать же провода прямо к ножкам, которые имеют микроскопический размер?

В третьих, микросхема не имеет средств для отображения информации, например о своем состоянии, равно как и для интерактивного ее получения. Как минимум, ей требуется кнопка для сброса в начальное состояние. Кроме этого, часто нужно обеспечить управление встроенной в микросхему программой. Для этой цели используют кнопки, переключатели и минимизированные, упрощенные клавиатуры.

Наконец, микросхема, являясь электронным прибором, нуждается в согласовании электрических характеристик сигналов, которые поступают на ее контакты, с сигналами, которые допускаются микросхемой. При передаче информации во внешнюю среду сигналы, которые микросхема выставляет на свои контакты, нужно усилить и привести к необходимому диапазону. Иначе говоря, микросхему нужно защитить от порчи при помощи других схем, которые выполняют роль буферов, согласователей и преобразователей между защищаемой микросхемой и внешними, реальными сигналами.

Вследствие указанных причин микросхему помещают на монтажную плату, на которой размещают все необходимые дополнительные схемы, устройства и разъемы. Плата при этом играет роль связника между компонентами устройства, так как она обеспечивает электрические соединения ножек микросхем, разъемов и устройств. Роль проводников выполняют тонкие металлические полоски, приклеенные к поверхности платы. Ножки компонентов платы припаивают к этим полоскам, и всю плату покрывают защитным лаком, что обеспечивает необходимую защиту от случайного электрического и механического воздействия, а также от окисления вследствие эксплуатации в агрессивных средах.

Так микросхема микроконтроллера обрастает необходимыми для нормального функционирования средствами, как бы обвязывается, опутывается ими. Про сопутствующие микросхеме компоненты иногда так и говорят — *обвязка*. В целом получается микроконтроллер, имеющий вид технического устройства.

В зависимости от предназначения, микроконтроллеры (в дальнейшем просто МК) можно условно разделить на две группы: промышленные и лабораторные. Назначение промышленных МК — управление устройствами, которые серийно выпускаются промышленностью, такими, как станки, роботы, бытовые приборы и т.п. Эти МК являются частью устройства и не

отделимы от них. Промышленные МК, в свою очередь, можно разделить на встраиваемые и блочные.

Встраиваемые МК, как следует из их названия, встраиваются в технические устройства для управления ими. Им присущи жестко заданные конструктивные, структурные и схемные особенности, связанные с устройством, в которое они встраиваются. Как правило, встраиваемые МК представляют собой плату, электрическое соединение с которой осуществляется при помощи одного разъема, установленного в устройстве. Встраиваемые МК содержат минимум обвязки, обусловленный обработкой информации вполне определенного, заданного вида. Назначение таких МК нельзя изменить после изготовления, а программа их функционирования «зашивается» в память при изготовлении и не изменяется в течение всего срока службы.

Блочные МК отличаются от встраиваемых тем, что допускают изменение как назначения, так и встроенной программы. Конструктивно блочные МК отвечают определенным требованиям, так как они устанавливаются в специальные унифицированные каркасные конструкции, и работают в сочетании с другими блоками. Блочные МК имеют сходство с лабораторными, так как они служат одной цели. С их помощью решают задачи разработки технических устройств. Блочность конструкции позволяет формировать разные конфигурации оборудования выбором тех или иных блоков, используемых для эксперимента, а стандартность блоков обеспечивает унифицированное сопряжение устройств.

Лабораторные МК предназначены в основном для экспериментирования. Это именно те МК, которые мы будем изучать. Они имеют произвольные массогабаритные характеристики, равно как и интерфейсные, предназначенные для согласования с внешними устройствами. Часто такие МК насыщены всевозможными схемами и устройствами для обеспечения их универсальности, что практически не присуще промышленным МК. С помощью лабораторных МК разрабатываются, например, встраиваемые МК. По большей части с помощью лабораторных МК решаются частные, узкие практические задачи, но эти задачи могут быть любыми, в чем и проявляется универсальность данного вида МК. Важной особенностью лабораторных и блочных МК является возможность частой замены встроенной в них программы.

Нельзя обойти вниманием МК, встроенные в смарт-карты и потребительские товары, такие, например, как MP3-плейеры или мобильные телефоны. Их трудно отнести к какой-нибудь группе, их вообще трудно отнести к группе микроконтроллеров. И тем не менее, не будь МК, не появились бы, наверное, и мобильные телефоны, и другие полезные технические усовершенствования. Важно то, что разработка программы для такого устройства принципиально ничем не отличается от разработки программы для обычного МК. Иногда этот вид управляющих устройств называют *firm*

ware. Особенностью этого вида является возможность легкой замены управляющей программы. Для этого используется обычный, настольный компьютер, к которому устройство подключается через интерфейс *USB*.

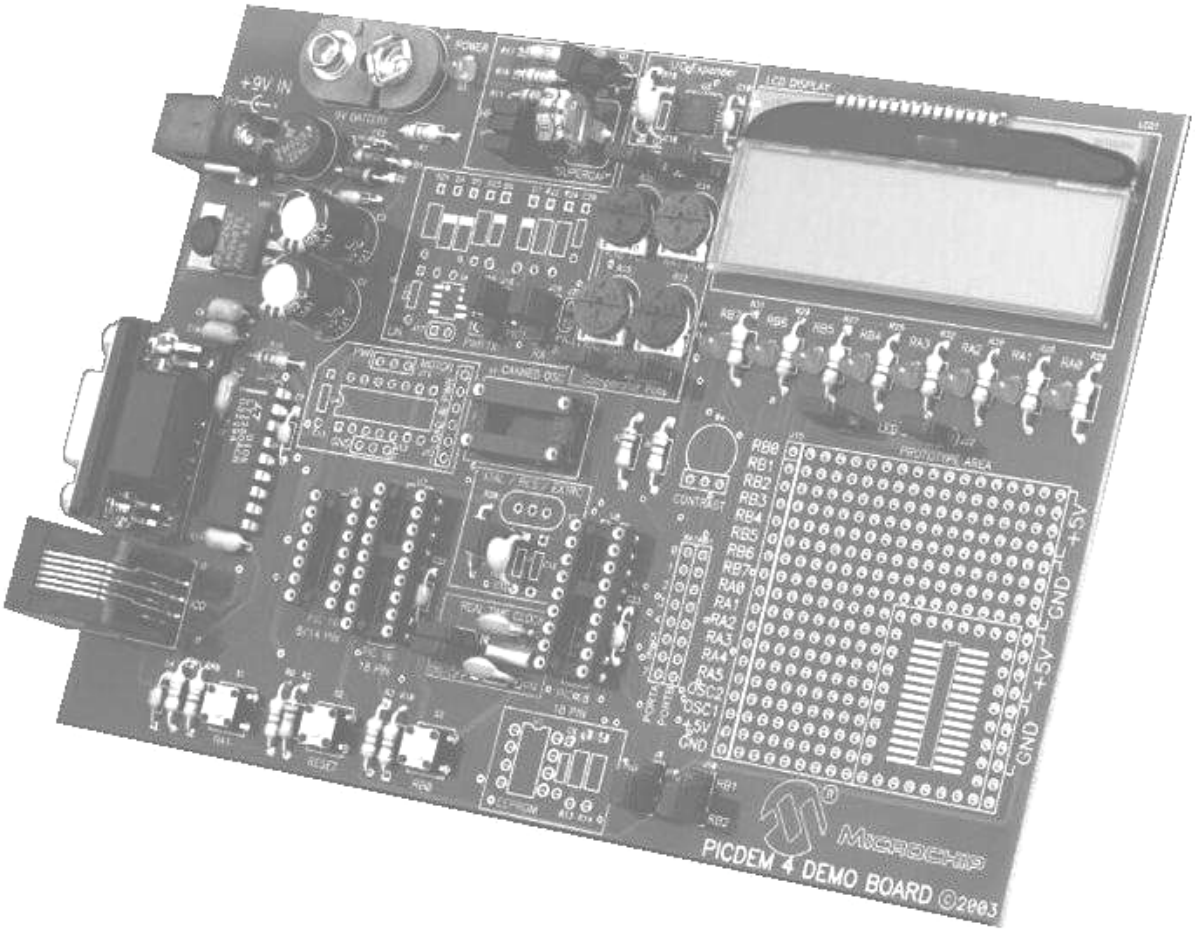


Рисунок 3. Лабораторная плата PICDEM 4 для испытания МК фирмы Microchip

Данная классификация ни в коей мере не описывает все возможные конструкции и области применения МК. Она приведена лишь с целью упорядочить общее представление об особенностях МК разных типов.

Назначение МК — управление каким-либо техническим устройством. В этом смысле МК — это автомат, реализующий определенную логику работы данного устройства. Области применения микроконтроллеров настолько разнообразны, что их непросто перечислить. МК может управлять металлорежущим станком, технологическим процессом приготовления пончиков, принтером, карбюратором, кофеваркой — чем угодно, лишь бы была линия передачи информации, а устройство обладало необходимыми механизмами для выполнения команд.

Именно в микроконтроллерах наиболее полно проявляется универсальная сущность программируемых систем — «дайте мне подходящую программу, и я сделаю все, что угодно». В отличие от настольного компьютера, микроконтроллер оснащается устройствами для приема и передачи сигналов, при помощи которых он интегрируется в *реальные* технические

устройства и *реально* управляет процессами в них. Он предназначен именно для управления другими устройствами. И только возможности устройства ограничивают возможности микроконтроллера.

Следует помнить, что микроконтроллер сам по себе является бесполезным устройством до тех пор, пока он изолирован от управляемых устройств. В этом случае он может выполнять разве что роль дорогих электронных часов или портативной электронной игрушки, если у него есть подходящий индикатор и соответствующая программа «зашита» в его память.

Предметная область

Прежде, чем приступать к изучению программирования микроконтроллеров, необходимо определить, что является предметной областью.

Здесь можно выделить как минимум три составляющих части — *информационный аспект* внешнего устройства, *интерфейсный аспект* каналов связи, и *архитектурный аспект* самого МК. Эти важнейшие аспекты предметной области находятся в тесной взаимосвязи благодаря существованию *временно́го аспекта*, который устанавливает временно́е соотношение между сигналами.

Информационный аспект описывает внешнее устройство в виде сигналов. ***Сигнал*** — изменение некоторой физической величины во времени, обеспечивающее передачу информации. Характеристика сигнала, используемая для представления информации, называется *параметром* сигнала. Микроконтроллеры имеют дело с электрическими сигналами, которые передаются по электрическим линиям связи (проводникам) в виде напряжения или тока. Параметрами электрического сигнала могут быть *амплитуда*, *частота*, *длительность* и *фаза*.

Сигналы, описывающие устройство, удобно разделить на две группы. Первая группа объединяет *контролируемые сигналы*, исходящие из устройства и относящихся к нему компонентов (рисунок 4). Контролируемые сигналы передают текущие характеристики информационной среды. Вторую группу составляют *сигналы управления*, или *команды*. Они описывают те или иные действия, которые устройство может выполнить. Информационный аспект, таким образом, — это совокупность контролируемых сигналов и сигналов управления.

При программировании микроконтроллера прежде всего нужно выяснить, какие сигналы используются для передачи информации, в каком направлении эти сигналы передаются, какую информацию несут, каким образом информация закодирована, в какой момент времени сигналы имеют смысл. Информационный аспект составляет логическую информационную модель системы.

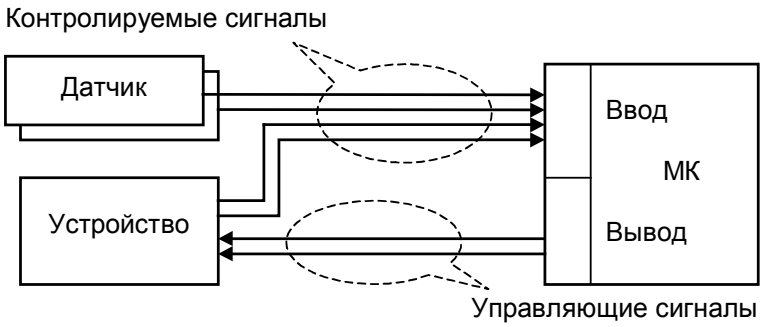


Рисунок 4. Информационный аспект предметной области

Интерфейсный аспект описывает интерфейсы и протоколы, используемые для установления связи и передачи информации между устройствами.

Интерфейс — это совокупность унифицированных программных, аппаратных и конструктивных средств, необходимых для реализации взаимодействия между различными информационными устройствами. Он описывает физические линии передачи, состав и структуру информационных и управляющих сигналов, логические и физические уровни сигналов, временные диаграммы взаимодействия устройств, способы синхронизации, методы кодирования и передачи информации, а также конструкцию соединительных устройств (разъемов).

Протокол — это описание последовательности действий, необходимых для установления соединения и передачи информации при помощи конкретного интерфейса. Он определяет информационный поток, передаваемый интерфейсом.

Интерфейсный аспект учитывается, когда для передачи информации между МК и устройством используется тот или иной стандартный интерфейс или протокол. Здесь программист должен уметь понимать временные диаграммы, описывающие информационный поток интерфейса, а также знать, как сформировать его посредством имеющихся в его распоряжении аппаратных средств.

Архитектурный аспект описывает хранение, обработку и передвижение информации внутри микроконтроллера. Он представляет структуру и архитектуру МК и состоит из карты распределения памяти, регистров и системы команд процессора, системы прерываний и логических связей между компонентами МК. Архитектурный аспект определяет, как записываются сигналы в память, какие преобразования могут быть выполнены с сигналами, как микроконтроллер отсчитывает временные интервалы, как реагирует на происходящие события и т.п.

Архитектурный аспект состоит из двух частей. Первая часть связана с архитектурой вычислительного устройства (процессора), которая лежит в основе МК. Эта часть архитектуры более или менее постоянна для целых групп (семейств) микроконтроллеров, и обычно обозначается, как *ядро* МК. Ядро определяет систему команд и типы данных, устройство памяти и

вычислительные операции, архитектуру основных устройств — таймеров, интерфейсов, системы прерываний. Архитектура ядра задается производителем главной микросхемы микроконтроллера-прибора.

Вторая часть связана с организацией устройств, которые окружают ядро. Именно внешние по отношению к ядру устройства определяют прохождение сигналов внутри микроконтроллера-прибора. Эта часть архитектуры МК определяется производителем микроконтроллера-прибора и является уникальной для конкретного устройства.

При программировании микроконтроллера знание организации микроконтроллера-прибора имеет самое важное значение. Например, текстовый индикатор (дисплей) может быть подключен непосредственно к портам микросхемы микроконтроллера, и в этом случае для управления им следует использовать команды записи в порт. Однако дисплей может быть включен как часть адресного пространства внешней памяти данных, и тогда для управления дисплеем следует использовать команды записи во внешнюю память.

Если в настольной вычислительной системе можно не знать деталей того, какие информационные потоки формируются в системе, то в отношении микроконтроллеров это не так. Знание организации микроконтроллера существенно важно для успешного его программирования. Организация конкретного микроконтроллера выясняется посредством анализа (прочтения) его принципиальной схемы, а также при помощи сопроводительной документации.

Еще одним важным аспектом, связанным с программированием МК, является процесс разработки и загрузки готовой управляющей программы во внутреннюю память МК. Для разработки управляющих программ используются кросс-компиляторы, а сама разработка ведется при помощи персонального компьютера (ПК). Для отладки программ часто используются симуляторы, имитирующие работу конкретного микроконтроллера на ПК, а также эмуляторы, которые позволяют пошагово выполнять программу, уже размещенную в микроконтроллере.

Размещение готовой программы в ПЗУ микроконтроллера выполняется при помощи специальных программных средств — загрузчиков (процесс загрузки называется также *доставкой*).

1. Общие сведения о микроконтроллерах

Как и всякая вычислительная система, МК содержит процессор, память, внешние устройства, а также схемы управления. Однако внутренняя организация микроконтроллеров обладает множеством особенностей. В первую очередь это связано с тем, что процессор, память и устройства находятся в одной микросхеме. Значительное влияние на организацию микроконтроллера оказывает его направленность на обработку информации в виде сигналов. Наконец, применение микроконтроллеров для управления самыми разнообразными устройствами предполагает наличие большой номенклатуры устройств с разнообразными характеристиками.

Большинство микроконтроллеров имеют 8-ми разрядную архитектуру. В этих микроконтроллерах шина данных имеет 8 разрядов и основной элемент данных — это байт. Тем не менее, выпускается большое количество микросхем с 4-х, 16-ти и 32-х разрядной шиной данных. Первые применяются в устройствах, в которых не требуется обработка больших объемов информации. Вторые используются там, где нужна высокая скорость обработки.

Ядро

Для обеспечения широкой гаммы устройств, микросхемы микроконтроллеров строятся на основе ядер. *Ядро* (core) — это постоянная часть целого семейства микросхем (чипов), включающая процессор со схемами синхронизации и управления (рисунок 5). Ядро определяет систему команд, организацию памяти и общие принципы обработки сигналов. Разные микросхемы семейства при этом отличаются количеством и структурой памяти, наличием тех или иных устройств, предельными частотными характеристиками. Это способствует получению более гибкого соотношения цена—характеристики конкретного чипа.

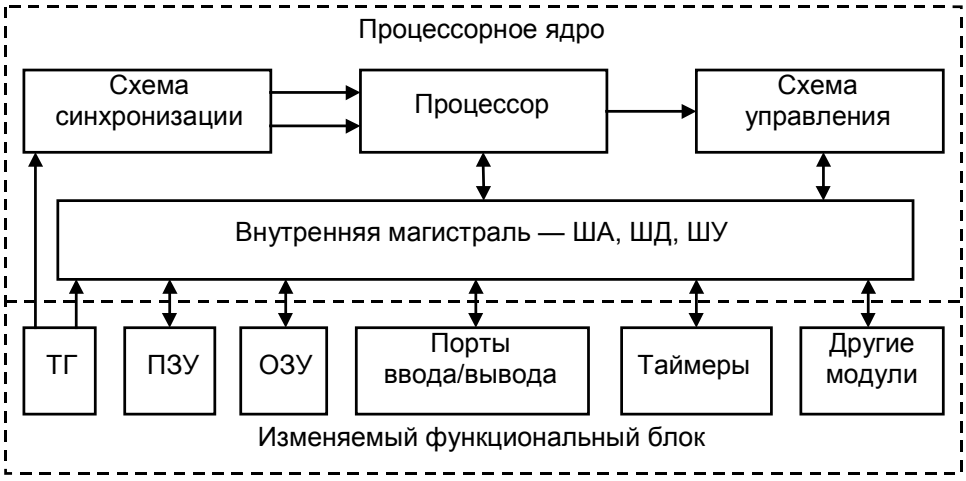


Рисунок 5. Ядро и функциональные модули

Микроконвертер ADuC842 построен на основе широко распространенного ядра 8052 (по названию микросхемы), которое, в свою очередь, является расширением еще более распространенного ядра 8051, являющегося классическим примером архитектуры 8-ми разрядного микроконтроллера общего назначения. Архитектура этого микроконтроллера носит также название MCS-51.

Синхронизация

Согласованная работа процессора и устройств системы осуществляется на основе тактовых сигналов (рисунок 6), которые формирует *задающий*, или *тактовый генератор* ТГ (рисунок 5). Интервал времени между двумя тактовыми импульсами называется *тактом*.

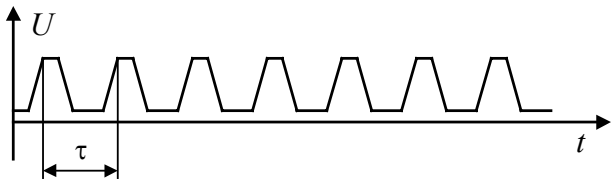


Рисунок 6. Тактовые синхронизирующие импульсы

Длительность такта τ является одной из важнейших характеристик. Она не только определяет скорость работы процессора, но используется также для измерений. Тактовые синхроимпульсы являются источником всех временных интервалов, которые получают делением тактовой частоты.

В МК системах частота и источник тактовых сигналов являются предметом тщательного анализа и расчета. В современных микроконтроллерах для управления частотой тактирующих сигналов используется схема программирования частоты ядра (*Programmable Core Clock*). С её помощью можно понижать частоту, поступающую от тактового генератора на внутренние схемы чипа. Делитель частоты (целое число) записывается в специальный регистр, управляющий схемой частоты ядра. Схема управления частотой называется PLL, а регистр управления схемой содержит буквы PLL в своем названии.

Понижение частоты ядра используется как для выбора рабочей частоты микроконтроллера, так и для снижения энергопотребления.

В учебном стенде SDK1.1 частота тактовых интервалов по умолчанию составляет 2 МГц, а длительность такта — 0,477 мкс.

Машинный цикл

Работа процессора строится на основе *машинных циклов*. В течение одного машинного цикла выполняется однократное обращение к памяти для чтения (записи) информации или внутренняя обработка информации.

Большая часть команд МК исполняется в течение одного-двух машинных циклов, в котором читается и исполняется одна машинная инст

рукция (команда). Если машинная инструкция предполагает чтение или запись данных, дополнительное обращение к памяти происходит во время дополнительных машинных циклов, число которых зависит от сложности команды. Кроме того, дополнительные машинные циклы могут использоваться для выполнения сложных арифметических команд, таких, как деление и умножение. Обращения к памяти при этом не происходит, и машинные циклы используются для выполнения последовательности преобразований внутри процессора.

Одной из важных характеристик микроконтроллера-микросхемы является длительность выполнения машинного цикла в тактах. Так, например, микроконвертер ADuC812 выполняет машинный цикл за 12 тактов, а микроконвертер ADuC842, используемый в учебном стенде SDK1.1, — всего за один такт. Иначе говоря, при одной и той же частоте тактового генератора второй микроконтроллер выполняет команды быстрее.

Время выполнения машинной команды в машинных циклах и в тактах используется для расчета временных интервалов программных задержек.

Архитектура памяти

Микроконтроллеры строятся на основе двух архитектур — традиционной фон-неймановской (принстонской) и гарвардской.

В архитектуре фон-Неймана (рисунок 7) память системы предназначена одновременно для хранения кода, данных и стека. Достоинством этой архитектуры является простота организации системы.

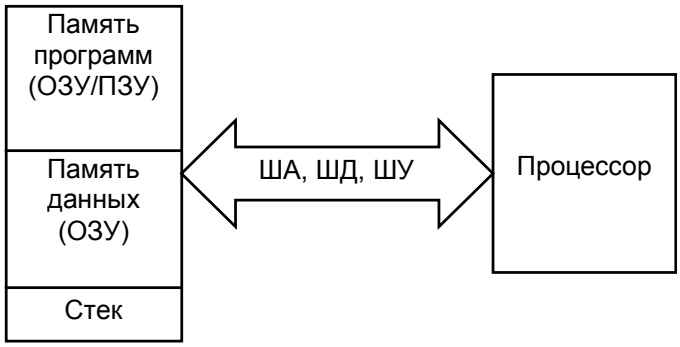


Рисунок 7. Фон-неймановская организация вычислительной системы

Одним из недостатков такой архитектуры является необходимость одинаковым образом адресовать элементы кода и элементы данных. Однако если элементы кода занимают значительное адресное пространство, то элементы данных во многих случаях занимают не более 256 байт, что позволяет использовать 8-битный адрес. Сокращение адресной части команд выборки операндов уменьшает размер команды и увеличивает производительность системы.

В МК наряду с фон-неймановской архитектурой большое распространение получила гарвардская архитектура, в которой память программ и память данных физически разделены (рисунок 8).

Система команд процессора содержит разные команды для чтения команд и для чтения/записи данных. Это позволяет оптимизировать команды для выполнения конкретных операций. В целом Гарвардская архитектура схожа с архитектурой фон-Неймана, но определяет большее количество адресных пространств.

Рисунок 8. Гарвардская организация вычислительной системы

Микроконтроллеры, построенные на основе ядер 8051/8052, выполнены по гарвардской архитектуре, память программ в них отделена от памяти данных, а стек располагается в памяти данных.

Память

Память — это устройство для хранения информации в виде двоичных слов. Память МК хранит информацию трех видов:

- *Управляющая* информация описывает логику, алгоритм работы системы. Она состоит из последовательности команд, образующей одну или несколько управляющих программ (подпрограмм).
- *Оперативная* информация отражает текущее состояние системы. Она состоит из данных, отражающих информационный аспект системы, и данных, необходимых для организации алгоритмов.
- *Стековая* информация хранит отложенные процессы и данные.

Отличительной особенностью микроконтроллеров является одновременное использование памяти самых разных типов и назначения. Эта особенность проявляется как в организации вычислительного процесса, так и в способах доступа к той или иной памяти. Как правило, в микроконтроллере, кроме основной, энергозависимой памяти (ОЗУ, оперативное запоминающее устройство), присутствует энергонезависимая память (ПЗУ, постоянное запоминающее устройство). Она используется для хранения констант и данных, которые не должны теряться между отключениями питания, а также для хранения управляющей программы.

Важнейшими характеристиками памяти являются скорость её работы, измеряемая временем выборки, а также способность памяти сохранять информацию при отключенном питании или при пониженном напряжении (при работе МК в режиме энергосбережения или ожидания).

Время выборки памяти — это время, которое проходит от начала обращения к памяти до момента получения информации. Время выборки оказывает значительное влияние на производительность устройства. В микроконтроллерах часто используется *статическая память*, обладающая самой высокой скоростью доступа. Кроме того, статическая память может

работать при значительно сниженном напряжении питания и не требует схем регенерации, необходимых для динамической памяти, используемой в качестве ОЗУ в портативных компьютерах.

В микроконтроллерах, как правило, отсутствует память в виде жесткого диска, которая в обычной системе является основным хранилищем программ и данных. Вместо этого управляющая программа микроконтроллера «зашивается» в постоянное запоминающее устройство (ПЗУ) и может сохраняться в нем длительное время (десять и более лет). В зависимости от назначения, в микроконтроллерах могут применяться разные типы ПЗУ.

В *масочные* ПЗУ (mask-ROM) информация заносится непосредственно во время изготовления. Это значительно снижает конечную стоимость устройства. *Однократно программируемые* ПЗУ (OTPROM — One-Time Programmable ROM) программирует пользователь при помощи специального устройства — программатора. В обоих случаях изменение управляющей программы в дальнейшем невозможно, поэтому ПЗУ таких типов используются только в серийном производстве, а управляющие программы тщательно отлаживаются.

В лабораторных микроконтроллерах используются ПЗУ, которые можно неоднократно перепрограммировать, модифицируя управляющую программу. Исторически первыми появились ПЗУ с оптическим (ультрафиолетовым) стиранием (EPROM — Erasable Programmable ROM). В таких ПЗУ запись информации производится электрическими сигналами с повышенным напряжением, а стирание — облучением памяти ультрафиолетовым светом через специальное окошко (рисунок 2).

В настоящее время в микроконтроллерах широко используются электрически стираемые ПЗУ (EEPROM — Electrically Erasable Programmable ROM) а также последняя их разновидность — Flash ПЗУ (Flash-EEPROM).

Отличительной особенностью использования программируемых ПЗУ является способ записи в них новой информации. Прежде всего память должна быть стерта. При этом все биты стираемых байтов устанавливаются в значение «1». Далее записываемая информация сбрасывает некоторые биты в «0».

Flash-ПЗУ отличается тем, что стирание (и, соответственно, запись) может быть выполнено только для блока байт, а не для одного байта.

Запись информации в перепрограммируемое ПЗУ выполняется либо в специальном внешнем устройстве (программаторе), либо непосредственно в микроконтроллере. В первом случае микросхема памяти извлекается из микроконтроллера и устанавливается в программатор.

Во втором случае программа, которая производит запись, «зашита» в ПЗУ микроконтроллера, а данные для записи передаются в микроконтроллер по каналу связи с, например, персональным компьютером. Перепрограммирование ПЗУ требует определенного времени, которое следует учитывать при разработке программы записи. Кроме того, перед записью но

вой информации следует убедиться, что стирание выполнилось успешно, либо проверять успешность записи новой информации.

Микроконтроллеры на основе ядер 8051/8052 в большинстве модификаций используют внутреннее статическое ОЗУ объемом не более 256 байт для памяти данных, а в качестве памяти программ в них используются ОЗУ, ПЗУ и их сочетания. Микроконвертер ADuC842 оснащен 2304 байт ОЗУ для хранения данных, 8, 32 или 64 Кбайт Flash/EEPROM для памяти программ, и дополнительным ПЗУ типа Flash/EEPROM объемом 4 Кбайт.

Порты

Порт является важной отличительной особенностью МК, и предназначен для прямой передачи информации между МК и управляемым устройством. Порт — это совокупность каналов для передачи двоичных сигналов. Несмотря на то, что порт можно использовать для параллельной передачи байтов информации, главное его назначение — обеспечение связи микроконтроллера с внешней средой посредством *независимых сигналов*.

Однонаправленный порт передает сигналы только в одном направлении — либо в микроконтроллер, либо из него. *Двунаправленный* порт может передавать сигналы в двух направлениях, но в отдельный момент времени — только в одну сторону.

Порты могут выполнять альтернативные функции. Например, они могут быть использованы для передачи адреса и данных из внутренней магистрали микроконтроллера в схемы внешней памяти.

Кроме того, часть каналов портов может использоваться для передачи сигналов интерфейсов или других устройств, расположенных в микросхеме. Разработчик микроконтроллера-устройства при этом выбирает, для какой цели используется тот или иной порт, а программист учитывает это при проектировании управляющей программы.

Микроконтроллеры на основе ядер 8051/8052 оснащены четырьмя двоичными двунаправленными портами, обозначаемыми P0—P3.

Таймеры

Таймер — это устройство для отсчета интервалов времени. Временные интервалы чрезвычайно важны в работе МК системы. Они дают возможность согласовать временные характеристики информационного, интерфейсного и архитектурного аспектов. С их помощью организуются необходимые задержки в исполнении алгоритмов для того, чтобы момент передачи сигнала одной стороной совпадал с моментом его приема другой стороной.

Таймеры используются также для счета (подсчета импульсов). Эта возможность определяется схемной организацией таймеров, в основе которой лежат *счетчики* (рисунок 9). Поэтому их называют также таймерами-счетчиками.

Как правило, таймеры могут быть настроены на различные режимы работы при помощи управляющих регистров. Кроме управляющих регистров, таймеры используют регистры, которые хранят текущее значение, используемое для счета.

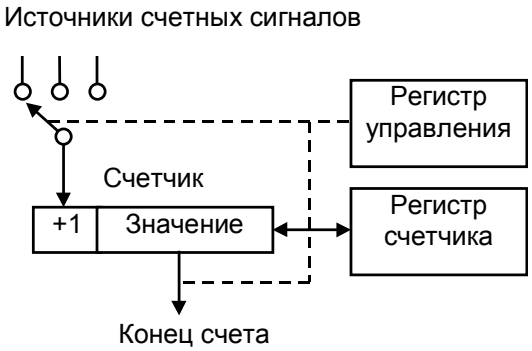


Рисунок 9. Структура таймера-счетчика

Регистры управления предназначены также для выбора источника счетных сигналов и действий, выполняемых по окончании счета (когда значение счета достигает максимума или минимума).

Следует учитывать, что таймеры-счетчики определяют временные интервалы, подсчитывая такты работы процессора или машинные циклы. Поэтому длительность такта τ является основой для расчета временных интервалов.

В микроконтроллеры на основе ядра 8051 встроены два 16-ти разрядных таймер-счетчика, обозначаемых T/C0 и T/C1 (timer/counter), позволяющих выполнять счет до 65536 тактов и отсчитывать временные интервалы до десятков миллисекунд. Микроконтроллеры на основе ядра 8052 оснащены дополнительным 16-ти разрядным таймер-счетчиком T2. Микроконвертер ADuC842 дополнительно имеет таймер 3 для формирования тактовых сигналов последовательного порта и таймер — счетчик временных интервалов, позволяющий отсчитывать длительные интервалы времени (секунды, минуты и часы).

Интерфейсы

Интерфейсы предназначены для организации связи МК с устройствами и с другими МК при помощи стандартных протоколов и уровней сигналов. Под интерфейсом в микроконтроллере понимается схема, обеспечивающая выработку сигналов в соответствии с заданным протоколом. Для передачи интерфейсных сигналов внешнему устройству используются как выделенные выводы микросхемы микроконтроллера, так и порты, которые в этом случае выполняют альтернативные функции.

Различают параллельные и последовательные интерфейсы. Последовательный интерфейс передает информационное слово, используя одну линию связи. Слово передается последовательно бит за битом. Параллельные интерфейсы передают слова целиком сразу по восьми линиям. В МК

чаще применяются последовательные интерфейсы, например, RS-232, RS-432, I^2C, SPI, CAN и другие.

При помощи интерфейсов несколько микроконтроллеров могут образовывать сеть. При этом один или несколько микроконтроллеров выступают в качестве *ведущего* (*ведущих*), в то время как другие являются *ведомыми*. Ведущий микроконтроллер вырабатывает сигналы синхронизации для обмена информацией в соответствии с используемым интерфейсом. Ведомые микроконтроллеры сканируют эти сигналы для определения состояния интерфейса в данный момент.

Все микроконтроллеры на основе ядер 8051/8052 оснащены последовательным интерфейсом, называемым также *последовательным портом*, обеспечивающим обмен информацией по протоколу RS-232. Для этого на кристалле микросхемы микроконтроллера расположен так называемый универсальный асинхронный последовательный приемопередатчик (UART).

Микроконвертер ADuC842 оснащен также встроенными схемами, обеспечивающими обмен информацией по интерфейсам SPI и I^2C.

Другие устройства

Среди других внешних устройств следует отметить специфичные для МК мониторы питания, сторожевые таймеры, а также АЦП и ЦАП.

Микроконтроллеры часто работают в автономном режиме, когда контроль за состоянием системы со стороны человека отсутствует. Для исключения зависания системы используется *сторожевой таймер* (*Watchdog Timer, WDT*), который при необходимости производит автоматический сброс системы. Работа сторожевого таймера основа на подсчете сигналов, которые управляющая программа должна генерировать время от времени. Если в течение заранее определенного времени сигналы не поступают, сторожевой таймер либо выполняет системный сброс, переводя программу в исходное состояние, либо генерирует сигнал прерывания.

Монитор питания (*Power Supply Monitor, PSM*) предназначен для контроля напряжения питания и удерживает микроконтроллер от старта до момента стабилизации рабочего напряжения. При недопустимом падении напряжения питания он генерирует сигнал прерывания, назначение которого — оповестить управляющую программу о необходимости сохранить важные данные.

Преобразователи аналогового сигнала в цифровой (АЦП) и цифрового сигнала в аналоговый (ЦАП) составляют неотъемлемую часть любой измерительной системы. Они осуществляют преобразование аналоговых сигналов в цифровые и обратное преобразование. АЦП и ЦАП реализуются либо в составе микроконтроллера-микросхемы, либо в виде отдельных устройств.

2. Микроконвертер ADuC842

Учебный лабораторный стенд SDK1.1, который используется для изучения организации и программирования микроконтроллеров, построен на основе современного многофункционального микроконвертера ADuC842 производства фирмы *Analog Devices* (в дальнейшем просто микроконвертер). Это весьма насыщенный устройствами прибор, с помощью которого можно выполнять сложные преобразования сигналов с большой скоростью. Микроконвертер построен на основе расширенного ядра 8052. Его характеристики:

- высокоскоростное ядро на основе архитектуры 8052;
- программная совместимость с архитектурой 8051;
- производительность до 20 MIPS;
- машинный цикл, исполняемый за один такт;
- программируемая частота ядра (PLL);
- номинальная частота такта 16,777 МГц, максимальная — 20 МГц;
- до 62 Кбайт Flash-ПЗУ памяти программ;
- 4 Кбайта Flash-ПЗУ данных;
- 256+2048 байт внутренней памяти данных (ОЗУ);
- адресация до 16 Мбайт внешней памяти данных;
- 8-канальный 12-разрядный АЦП, выполняющий до 420К выборок/с;
- контроллер прямого доступа к памяти (DMA);
- два 12-разрядных ЦАП с вольтовым выходом;
- два широтно-импульсных модулятора (PWM), 6 режимов;
- 12 источников прерываний, два приоритета;
- двойной указатель данных с автоинкрементом;
- расширенный до 11 разрядов указатель стека;
- 4 параллельных порта;
- 3 таймера-счетчика, таймер последовательного порта;
- счетчик временных интервалов (TIC);
- последовательные интерфейсы RS-232, I^2C, SPI;
- монитор питания (PSM);
- сторожевой таймер (WDT);
- внутренний температурный сенсор (датчик).

Программирование такого сложного прибора может вызвать значительные затруднения, так как требует понимания принципов работы многих устройств, таких, например, как АЦП или интерфейс I^2C. Однако на практике используется не вся совокупность устройств микроконвертера, а только та из них, которая непосредственно вовлечена в процесс управления конкретным устройством.

Управляющая программа для такого устройства может иметь значительный объем, поэтому его программирование обычно выполняется на языке высокого уровня, например, на языке Си.

Организация памяти

Память программ

Ядро микроконвертера построено по гарвардской архитектуре — память программ отделена от памяти данных. Адресное пространство памяти программ составляет 64 Кбайт (16-битный адрес), из которых для размещения программ доступно до 62 Кбайт (рисунок 10).

В памяти программ располагаются инструкции управляющей программы, непосредственные данные, константы в виде отдельных операндов либо в виде таблиц, а также элементы данных.

Для адресации памяти программ в основном используется неявный (недоступный для прямых записи и чтения) программный счетчик PC (*Program Counter*), но есть несколько инструкций, которые позволяют адресовать константы, расположенные в памяти программ, при помощи указательного регистра DPTR (*Data Pointer*).

Последние 2 Кбайта адресного пространства памяти программ (область памяти с адресами 0F000h–0FFFFh) используются самим микроконвертером — здесь размещен внутренний загрузчик и инструкции для отладки при помощи эмулятора (внешнего отладчика). При обращении к этой области памяти возвращается инструкция NOP.

Память программ представляет собой электрически стираемое Flash-ПЗУ, расположенное в микроконвертере, поэтому перепрограммирование управляющей программы может быть осуществлено непосредственно самим микроконвертером.

Рисунок 10. Память программ 62 Кбайт

Верхние 6 Кбайт области 62 Кбайт отводятся для размещения загрузчика пользователя — запись программы в эту область возможна только при помощи внутреннего загрузчика (в SDK1.1 здесь размещен загрузчик программ HEX202-03). Загрузчик пользователя предназначен для оперативной доставки в оставшиеся 56 Кбайт программ пользователя в режиме ULOAD (User Load).

Память данных

Память данных представлена статическим ОЗУ IRAM объемом 256 байт и внутренней внешней памятью данных On-Chip XRAM объемом 2048 байт (рисунок 11).

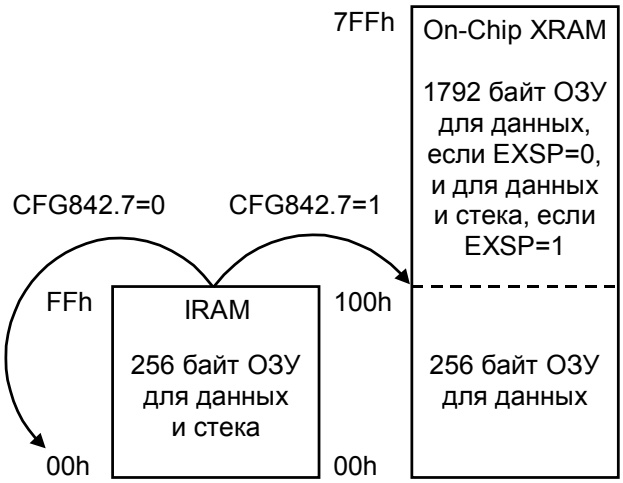


Рисунок 11. Память данных

Основной областью ОЗУ являются нижние 128 байт IRAM. Эта область доступна для чтения и записи информации любым методом адресации. Структура этой области ОЗУ представлена на рисунке 12.

78h									7Fh	
70h									77h	
68h									6Fh	
60h									67h	
58h									5Fh	Свободная память
50h									57h	
48h									4Fh	
40h									47h	
38h									3Fh	
30h									37h	
28h									2Fh	Битовое поле
20h									27h	
18h	R0	R1	R2	R3	R4	R5	R6	R7	1Fh	Банк 3
10h	R0	R1	R2	R3	R4	R5	R6	R7	17h	Банк 2
08h	R0	R1	R2	R3	R4	R5	R6	R7	0Fh	Банк 1
00h	R0	R1	R2	R3	R4	R5	R6	R7	07h	Банк 0

Рисунок 12. Распределение первых 128 байт ОЗУ

Первые 32 байта ОЗУ можно использовать как регистры, обозначаемые R0—R7, и образующие 4 банка по 8 регистров. Следующие 16 байт образуют *битовое поле* из 128 бит. Каждый бит этого поля имеет свой соб

ственный *битовый адрес*. Оставшиеся 80 байт используются программистом для хранения временных данных (для переменных).

Каждая ячейка памяти имеет свой собственный адрес, независимо от того, к какой части внутренней памяти данных она принадлежит. Так, регистр R0 банка 0 имеет адрес 00h, а регистр R7 банка 3 имеет адрес 1Fh.

Регистр Rn может относиться к четырем ячейкам памяти, имеющим адреса n, n+8, n+16 и n+24. Выбор той или иной ячейки памяти для конкретного регистра зависит от текущего банка, который определяется битами RS0–RS1 регистра PSW (рисунок 13).

C	AC	F0	RS1	RS0	OV	F1	P	Биты PSW
X	X	X	0	0	X	X	X	Банк 0
X	X	X	0	1	X	X	X	Банк 1
X	X	X	1	0	X	X	X	Банк 2
X	X	X	1	1	X	X	X	Банк 3

Рисунок 13. Выбор текущего банка в регистре состояния

Ячейки памяти с адресами 20h–2Fh занимает битовое поле. Соотношение между битами байтов битового поля и их шестнадцатеричными битовыми адресами поясняет рисунок 14.

Байт	Номер бита							
Адрес	7	6	5	4	3	2	1	0
2Fh	7F	7E	7D	7C	7B	7A	79	78
2Eh	77	76	75	74	73	72	71	70
2Dh	6F	6E	6D	6C	6B	6A	69	68
2Ch	67	66	65	64	63	62	61	60
2Bh	5F	5E	5D	5C	5B	5A	59	58
2Ah	57	56	55	54	53	52	51	50
29h	4F	4E	4D	4C	4B	4A	49	48
28h	47	46	45	44	43	42	41	40
27h	3F	3E	3D	3C	3B	3A	39	38
26h	37	36	35	34	33	32	31	30
25h	2F	2E	2D	2C	2B	2A	29	28
24h	27	26	25	24	23	22	21	20
23h	1F	1E	1D	1C	1B	1A	19	18
22h	17	16	15	14	13	12	11	10
21h	0F	0E	0D	0C	0B	0A	09	08
20h	07	06	05	04	03	02	01	00

Рисунок 14. Адреса битов битового поля

Эта часть памяти может использоваться двояким образом. Если команда предполагает операцию с байтом, то указанный в ней адрес указы

вает на один из байтов в битовом поле (как обычно). Если же команда предполагает операцию с битом, то указанный в ней адрес является номером (адресом) одного из битов битового поля.

Регистры специальных функций SFR

Область ОЗУ с адресами 080h—0FFh также используется двояким образом. При прямом методе адресации эта часть представляет собой поле регистров специальных функций (*Special Function Registers*, рисунок 15).

Регистры специальных функций — это ячейки памяти, предназначенные для управления внутренними устройствами микроконвертера. С их помощью можно задать режим работы конкретного устройства, проверить его состояние, выполнить передачу данных в устройство или из него. Подробнее регистры описываются в соответствующих разделах.

SPICON F8 * 04h	DAC0L F9 00h	DAC0H FA 00h	DAC1L FF 00h	DAC1H FF 00h	DACCON FF 04h		
B F0 * 00h	ADCOFSL F1 00h	ADCOFSH F2 20h	ADCGAINL F3 00h	ADCGAINH F4 00h	ADCCON3 F5 00h		SPIDAT F7 00h
I2CCON E8 * 00h							ADCCON1 EF 40h
ACC E0 * 00h							
ADCCON2 D8 * 00h	ADCDATAL D9 00h	ADCDATAH DA 00h					PSMCON DF DEh
PSW D0 * 00h		DMAL D2 00h	DMAH D3 00h	DMAP D4 00h	D5		PLLCON D7 53h
T2CON C8 * 00h		RCAP2L CA 00h	RCAP2H CB 00h	TL2 CC 00h	TH2 CD 00h		
WDCON C0 * 10h		CHIPID C2 A6h				EDARL C6 00h	EDARH C7 00h
IP B8 * 00h	ECON B9 00h			EDATA1 BC 00h	EDATA2 BD 00h	EDATA3 BE 00h	EDATA4 BF 00h
P3 B0 * FFh	PWM0L B1 00h	PWM0H B2 00h	PWM1L B3 00h	PWM1H B4 00h			SPH B7 00h
IE A8 * 00h	IEIP2 A9 A0h					PWMCON AE	CFG842 AF 00h
P2 A0 * FFh	TIMECON A1 00h	HTHSEC A2 00h	SEC A3 00h	MIN A4 00h	HOUR A5 00h	INTVAL A6 00h	DPCON A7 00h
SCON 98 * 00h	SBUF 99 00h	I2CDAT 9A 00h	I2CADD 9B 55h		T3FD 9D 00h	T3CON 9E 00h	
P1 90 * FFh	I2CADD1 91 7Fh	I2CADD2 92 7Fh	I2CADD3 93 7Fh				
TCON 88 * 00h	TMOD 89 00h	TL0 8A 00h	TL1 8B 00h	TH0 8C 00h	TH1 8D 00h		
P0 80 * FFh	SP 81 07h	DPL 82 00h	DPH 83 00h	DPP 84 00h			PCON 87 00h

Рисунок 15. Регистры специальных функций (SFR)

Часть регистров специальных функций (те, шестнадцатеричный адрес которых заканчивается нулем или восьмеркой), являются бит-адресуемыми. Каждому биту в этих регистрах сопоставлен собственный битовый адрес, а для некоторых битов определены специальные названия.

На рисунке 16 приведена расшифровка обозначений карты регистров SFR. На рисунке 17 приведена карта битов регистров SFR.

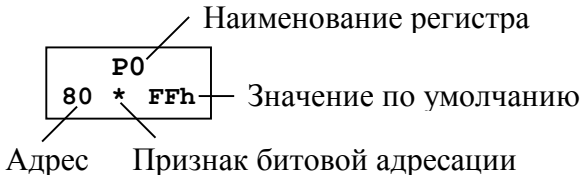


Рисунок 16. Расшифровка карты регистров SFR

ISPI FF 0	WCOL FE 0	SPE FD 0	SPIM FC 0	CPOL FB 0	CPHA FA 1	SPR1 F9 0	SPR0 F8 0	SPICON
F7 0	F6 0	F5 0	F4 0	F3 0	F2 0	F1 0	F0 0	B
MDO EF 0	MDE EE 0	MCO ED 0	MDI EC 0	I2CM EB 0	I2CRS EA 0	I2CTX E9 0	I2CI E8 0	I2CCON
E7 0	E6 0	E5 0	E4 0	E3 0	E2 0	E1 0	E0 0	ACC
ADCI DF 0	DMA DE 0	CCONV DD 0	SCONV DC 0	CS3 DB 0	CS2 DA 0	CS1 D9 0	CS0 D8 0	ADCCON2
CY D7 0	AC D6 0	F0 D5 0	RS1 D4 0	RS0 D3 0	OV D2 0	F1 D1 0	P D0 0	PSW
TF2 CF 0	EXF2 CE 0	RCLK CD 0	TCLK CC 0	EXEN2 CB 0	TR2 CA 0	CNT2 C9 0	CAP2 C8 0	T2CON
PRE3 C7 0	PRE2 C6 0	PRE1 C5 0	PRE0 C4 1	WDIR C3 0	WDS C2 0	WDE C1 0	WDWR C0 0	WDCON
PSI BF 0	PADC BE 0	PT2 BD 0	PS BC 0	PT1 BB 0	PX1 BA 0	PT0 B9 0	PX0 B8 0	IP
RD B7 1	WR B6 1	T1 B5 1	T0 B4 1	INT1 B3 1	INT0 B2 1	TxD B1 1	RxD B0 1	P3
EA AF 0	EADC AE 0	ET2 AD 0	ES AC 0	ET1 AB 0	EX1 AA 0	ET0 A9 0	EX0 A8 0	IE
A7 1	A6 1	A5 1	A4 1	A3 1	A2 1	A1 1	A0 1	P2
SM0 9F 0	SM1 9E 0	SM2 9D 0	REN 9C 0	TB8 9B 0	RB8 9A 0	TI 99 0	RI 98 0	SCON
97 1	96 1	95 1	94 1	93 1	92 1	T2EX 91 1	T2 90 1	P1
TF1 8F 0	TR1 8E 0	TF0 8D 0	TR0 8C 0	IE1 8B 0	IT1 8A 0	IE0 89 0	IT0 88 0	TCON
87 1	86 1	85 1	84 1	83 1	82 1	81 1	80 1	P0

Рисунок 17. Биты регистров SFR

В целом биты битового поля и биты регистров SFR образуют битовое пространство в 256 битов.

Область памяти с адресами 080h—0FFh может быть также использована для хранения данных (переменных) при использовании косвенного метода адресации, однако чаще она используется для стека.

Область стека

Стек располагается в области ОЗУ. По умолчанию указатель стека установлен на ячейку с адресом 07h. Так как стек растет вверх, первая ис

пользуемая ячейка стека имеет адрес 08h. Иначе говоря, по умолчанию область стека занимает банк регистров с номером 1, а если недостаточно, — то банки 2 и 3 (и далее область битового поля). Обычно программист не использует дополнительные банки регистров R0–R7, поэтому область стека не конфликтует с ячейками памяти, если стек используется не часто.

Как правило, для размещения стека используется вторая половина ОЗУ (память с адресами выше 07Fh). Для размещения данных эта память может быть использована только при помощи косвенного метода адресации, а команды для записи в стек и извлечения из стека как раз используют косвенный метод адресации через регистр стека SP. В обычном режиме (режиме совместимости с ядром 8051/8052) при достижении указателем стека значения 0100h указатель стека переходит на начало памяти, то есть на адрес 0h, и стек начинает переписывать банки регистров.

В микроконвертере есть режим расширенной адресации стека, когда по достижении адреса 0100h область стека перемещается в область внутренней внешней памяти On-Chip XRAM (рисунок 11).

Для организации расширенной адресации стека используются дополнительный регистр указателя стека SPH, содержащий старшие 3 бита указателя стека, и регистр конфигурации CFG842.

Если бит EXSP в регистре CFG842 не установлен, стек используется обычным образом. Так, если указатель достигает значения 100h, то он переходит к значению 00h, и указывает на ОЗУ. Если же бит EXSP в регистре CFG842 установлен, указатель стека при достижении значения 100h начинает указывать на область 100h–7FFh внутренней внешней памяти данных.

Внешняя память данных

Кроме внутренней памяти данных, микроконвертер может обращаться к внешней, расположенной вне микросхемы, памяти типа ОЗУ. Такая память называется XRAM (*External RAM*). В учебном стенде SDK1.1 установлено 512 Кбайт такой памяти.

Для обращения к внешней памяти данных используются специальные инструкции MOVX, использующие косвенный метод адресации через 24-х разрядный регистр данных DPTR. Он состоит из трех 8-разрядных регистров — DPL (*Data Pointer Low*), DPH (*Data Pointer High*) и DPP (*Data Pointer Page*). Так как в микросхему встроена внутренняя внешняя память данных, для выбора внешней или внутренней памяти используется бит XRAMEN регистра конфигурации CFG842.

Если бит XRAMEN не установлен (по умолчанию), то используется внешняя память, а если установлен, то внутренняя память занимает нижние 2 Кбайта адресного пространства внешней памяти. Иначе говоря, если бит XRAMEN установлен, то инструкция MOVX обращается к внутренней памяти, если адрес ниже 800h, и к внешней памяти, если адрес больше или равен 800h.

ПЗУ данных

Микроконвертер содержит в своем составе 4 Кбайта Flash-ПЗУ для сохранения данных, например, на время отключения питания. Эта память организована в 1024 банка (страницы) по 4 байта. Такая организация есть следствие блочной архитектуры памяти типа Flash/EE.

Для доступа к этой памяти сначала при помощи регистров EADRL и EADRH выбирается страница, затем данные могут быть считаны или записаны при помощи четырех регистров данных EDATA1–EDATA4. Выполняемая операция определяется командой, записанной в регистр ECON (таблица 1).

Таблица 1. Команды ECON для управления Flash памятью данных

Команда	***Описание***
01h Read Page	Чтение 4-х байт из выбранной страницы в регистры данных EDATA1–EDATA4.
02h Write Page	4 байта из регистров данных EDATA1–EDATA4 записываются в выбранную страницу.
04h Verify Page	Проверка соответствия регистров EDATA1–EDATA4 выбранной странице. Если последующее чтение регистра ECON возвращает 0, содержимое страницы соответствует содержимому регистров EDATA1–EDATA4.
05h Erase Page	Стирает выбранную страницу. Все байты страницы, на которую указывают регистры EADRL, EADRH, получают значение FFh.
06h Erase All	Стирает содержимое всей памяти данных.
81h ReadByte	Прямо адресуемый при помощи EADRL и EADRH байт читается в регистр EDATA1 из памяти данных.
82h WriteByte	Прямо адресуемый при помощи EADRL и EADRH байт записывается из регистра EDATA1 в память данных.
0Fh EXULOAD	Регистр ECON остается в нормальном режиме.
F0h ULOAD	Регистр ECON переходит в режим ULOAD.

Кроме этого, предусмотрена возможность прямой адресации отдельного байта. Соответствие между байтами страниц и их прямым адресом приведено на рисунке 18 (в скобках).

Следует иметь ввиду, что регистр ECON управляет не только памятью данных, но и памятью программ. Назначение регистра определяется командой, которая записывается в этот же регистр, переключая его либо в нормальный режим (управление памятью данных), либо в режим пользовательской загрузки программ ULOAD.

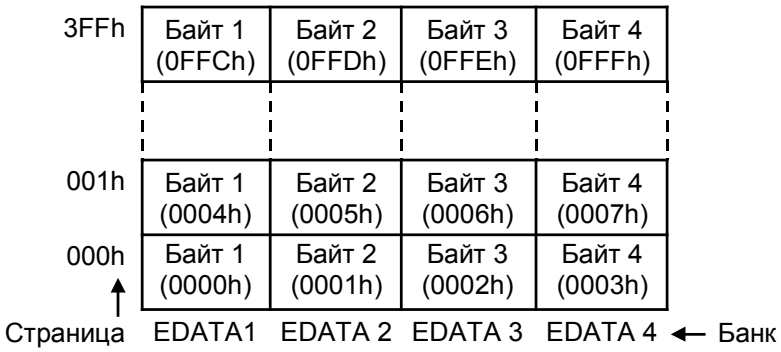


Рисунок 18. Адреса байтов ПЗУ данных

Регистр флагов

Регистр флагов (рисунок 19) предназначен для индикации текущего состояния процессора. Он определяет четыре флага, указывающие на результат операции, и четыре флага, имеющие вспомогательное назначение.

CY	AC	F0	RS1	RS0	OV	F1	P
7	6	5	4	3	2	1	0

Рисунок 19. Регистр флагов микроконвертера ADuC842

Флаг P (*Parity* —четность) модифицируется каждой командой и устанавливается в 1, если аккумулятор содержит четное число двоичных единиц. В противном случае флаг сбрасывается в 0.

Флаг OV (*OVerflow* — переполнение) устанавливается некоторыми командами для индикации переполнения результата арифметической операции. Предназначен для контроля выполнения операций над числами со знаком в дополнительном коде.

Флаг AC (*Additional Carry* — дополнительный перенос) устанавливается, если при выполнении арифметической команды возник перенос в разряд 3 результата (или произошел заем из разряда 3). Предназначен для выполнения операций с двоично-кодированными десятичными числами.

Флаг CY (*Carry* — перенос) устанавливается, если при выполнении арифметической операции произошел перенос в разряд 7 результата (или произошел заём из разряда 7). Этот флаг используется также в качестве промежуточного бита при циклическом сдвиге аккумулятора. Кроме того, флаг CY используется при выполнении битовых операций и в этой связи трактуется как битовый аккумулятор. Этот флаг обозначается также как CF или просто C.

Флаги RS0 и RS1 предназначены для выбора банка регистров R0—R7 (дополнительно см. рисунок 13.) Эти флаги устанавливает и сбрасывает программист для выбора того или иного текущего банка регистров. По умолчанию оба флага равны нулю и текущий банк регистров — 0.

Флаги F0 и F1 — пользовательские. Программист использует их по своему усмотрению.

Порты

Порты используются для передачи двоичных сигналов. В микроконвертере предусмотрено четыре восьмиразрядных порта, обозначаемых P0, P1, P2 и P3. Все порты являются двунаправленными, то есть обеспечивают передачу сигналов как в микроконтроллер (ввод данных), так и из него (вывод). При этом часть разрядов порта может быть использована для ввода, в то время как другая часть — для вывода.

Все порты являются бит-адресуемыми. Это означает, что имеется возможность программно управлять только одним разрядом любого порта.

В реальных устройствах, управляемых микроконтроллером, количество двоичных сигналов редко достигает нескольких десятков. Как правило, это всего несколько сигналов. С другой стороны, часто необходимо, чтобы микроконтроллер одновременно обеспечивал передачу информации по одному из встроенных интерфейсов, или, например, записывал большие объемы информации во внешнюю память данных. Для снижения количества выводов микросхемы, порты могут выполнять *альтернативные функции*.

Порты P0 и P2 используются при обращении к внешней памяти. При этом младшая часть 16-разрядного адреса передается через порт P0, а старшая — через порт P2. Данные передаются через порт P0. Таким образом, порт P0 используется и для передачи адреса, и для передачи данных, но происходит это в разные моменты времени (происходит временно́е *мультиплексирование* порта). Сначала через порт выводится адрес, который защелкивается (запоминается) во внешнем регистре при помощи высокого сигнала на выводе ALE (*Address Latch Enable*, разрешение защелки адреса), а затем передается байт информации. Программисту не нужно заботиться о записи адреса или данных в порт P0 при обращении к внешней памяти данных, так как микроконтроллер выполняет это автоматически при обращении к области памяти, отсутствующей на кристалле. Необходимо только, чтобы внешние по отношению к микроконтроллеру схемы памяти и защелка адреса были должным образом аппаратно реализованы.

Если при обращении к внешней памяти используется восьмиразрядный адрес, то порт P2 не участвует в передаче адреса и может быть использован обычным образом для передачи сигналов.

При обращении к внешней памяти в защелки каналов порта P0 записываются единицы.

Порт P3 в альтернативном режиме используется для передачи специальных управляющих и информационных сигналов в соответствии с таблицей 2.

Схема одного канала порта P0 приведена на рисунке 20.

При записи в порт в обычном режиме информационный бит поступает на вход регистра-защелки по внутренней магистрали микроконтроллера и

защелкивается в нем по сигналу «Запись SFR», вырабатываемому схемой управления.

Таблица 2. Альтернативные функции порта P3

Вывод порта	***Альтернативная функция***
P3.0	RxD — вход последовательного интерфейса
P3.1	TxD — выход последовательного интерфейса
P3.2	INT0 — внешнее прерывание 0
P3.3	INT1 — внешнее прерывание 1
P3.4	T0 — вход таймера-счетчика 0
P3.5	T1 — вход таймера-счетчика 1
P3.6	WR — строб записи во внешнюю память данных
P3.7	RD — строб чтения из внешнюю память данных

Сигнал «*Control*» имеет низкое значение, и переключатель S направляет инвертированный входной сигнал на вход нижнего выходного полевого транзистора VT2. Одновременно низкий сигнал «*Control*» переводит выход элемента И-НЕ в высокое состояние, закрывая верхний транзистор VT1, и отключая выход канала от источника питания V_{DD}.

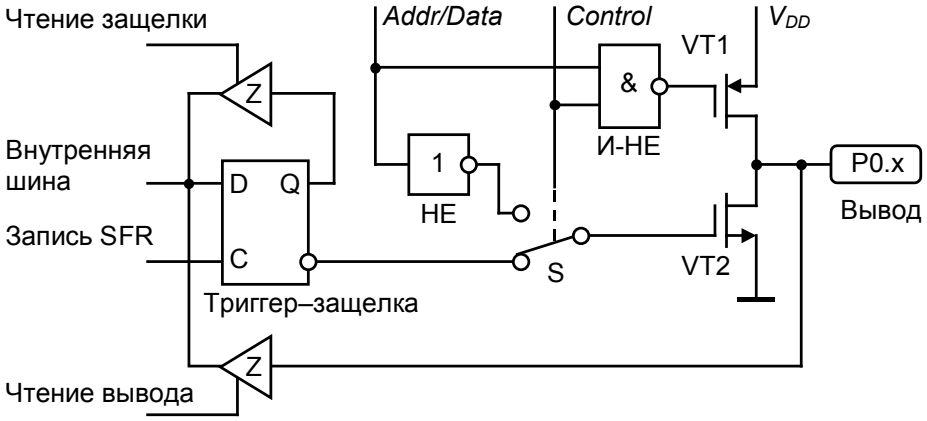


Рисунок 20. Канал порта P0

Такое состояние выхода называется «открытым стоком» и требует, чтобы нагрузка выходного транзистора VT2 обеспечивалась вне микросхемы с помощью дополнительного резистора.

В режиме вывода адреса/данных высокое состояние сигнала «*Control*» подключает вход транзистора VT2 к инвертору сигнала «*Addr*/*Data*», а вход транзистора VT1 подключается к выходу схемы И-НЕ, которая также переходит в режим инверсии. Одинаковый сигнал на входах выходных транзисторов переводит их в противоположные состояния. Когда сигнал «*Addr*/*Data*» имеет низкое состояние, транзистор VT1 закрыт, а транзистор VT2 открыт, и на выходе канала низкое напряжение. При высоком сигнале «*Addr*/*Data*» наоборот, верхний транзистор открыт, а нижний закрыт, и на выходе канала высокое напряжение.

В режиме чтения в зависимости от операции открывается либо верхняя, либо нижняя ключевая буферная схема Z, и на внутреннюю магист

раль микроконтроллера поступает либо значение выхода Q триггера-защелки (чтение защелки), либо значение, поступившее на вывод канала (чтение вывода). При этом в триггер-защелку должно быть записано значение 1, которое отключает нижний выходной транзистор и переводит выходной каскад канала в режим «высокий импеданс» так, что выходная схема не мешает прохождению сигнала с вывода микросхемы на внутреннюю магистраль микроконтроллера.

Схема одного канала порта P1 приведена на рисунке 21.

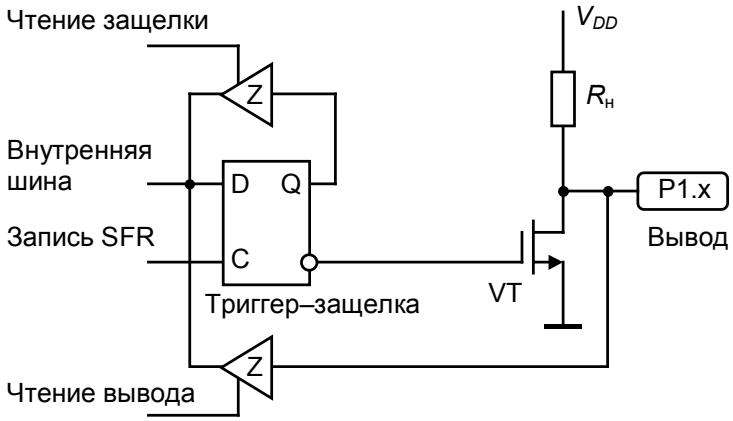


Рисунок 21. Канал порта P1

Порт P1 используется только по прямому назначению — для ввода и вывода двоичных сигналов, поэтому в нем отсутствуют схемы, связанные с альтернативными функциями. Нагрузка выхода канала (резистор Rн) включена непосредственно в микросхему.

Схема порта P2 примерно соответствует схеме порта P0, за исключением того, что вместо выходного транзистора VT1 в нем подключен регулируемый (управляемый) нагрузочный резистор.

Схема одного канала порта P3 приведена на рисунке 22.

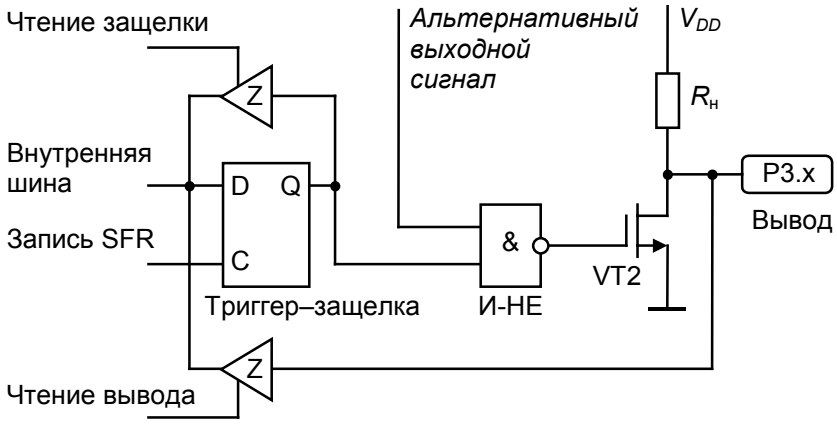


Рисунок 22. Канал порта P3

При обычном выводе альтернативный сигнал равен единице и схема И-НЕ работает как инвертор сигнала, записанного в триггер-защелку. При выполнении альтернативных функций вывода в регистр-защелку записы

вается единица, схема И-НЕ снова переходит в режим инверсии, но инвертируется сигнал альтернативной функции. При чтении схема канала порта P3 работает аналогично порту P1.

Таймеры

Микроконвертер ADuC842 имеет 5 встроенных таймеров. Таймеры 0 и 1 обеспечивают совместимость с ядром 8051, таймер 2 — с ядром 8052, и два таймера расширяют ядро 8052.

Таймеры-счетчики 0 и 1

Таймеры-счетчики 0 и 1 представляют собой два программируемых 16-разрядных таймера-счетчика, обозначаемых T/C0 и T/C1. Оба устройства могут быть использованы как в качестве таймеров, так и в качестве счетчиков.

В режиме таймера содержимое счетчика увеличивается на единицу при выполнении каждого машинного цикла. Так как машинный цикл выполняется за 1 такт задающего генератора, единица отсчета равна 1 такту.

В режиме счета на счетный вход таймера подается сигнал с вывода T0 или T1 микросхемы. При переходе состояния сигнала на одном из этих входов из 1 в 0 происходит увеличение соответствующего счетчика. Для распознавания сигнала счета входной сигнал 1 должен удерживаться на входе счетчика как минимум один машинный цикл, а максимальная частота счетных сигналов не может превышать частоты тактового генератора.

Управление таймерами-счетчиками производится при помощи регистров SFR TCON и TMOD, а текущие значения счетчиков доступны через пары регистров TL0/TH0 и TL1/TH1.

Регистр TMOD задает режимы работы таймеров. Назначение битов этого регистра приведено в таблице 3.

Таблица 3. Регистр режима работы TMOD таймеров 0 и 1

Имя	*Таймер*	*Разряд*	*Назначение*
GATE	1	7	Блокировка. Если GATE=1, таймер 1 включается единицей на входе INT1, при этом бит TR1 должен быть установлен. Если GATE=0, таймер включается при установке бита TR1.
C/T	1	6	Счетчик/таймер. Если этот бит установлен, таймер 1 работает как счетчик, иначе как таймер.
M1	1	5	Вместе с битом 4 задает режим таймера 1.
M0	1	4	Вместе с битом 5 задает режим таймера 1.
GATE	0	3	Блокировка. Если GATE=1, таймер 0 включается единицей на входе INT0, при этом бит TR0 должен быть установлен. Если GATE=0, таймер включается при установке бита TR0.
C/T	0	2	Счетчик/таймер. Если этот бит установлен, таймер 0 работает как счетчик, иначе как таймер.
M1	0	1	Вместе с битом 0 задает режим таймера 0.

M0	0	0	Вместе с битом 1 задает режим таймера 0.

Режим работы таймера определяется комбинациями его битов M0/M1. Они определяют по четыре режима работы для каждого из таймеров. Отдельный режим работы обозначается цифрой от 0 до 3. Соответствие между значениями битов M0 и M1 и номером режима приведено в таблице 4.

Таблица 4. Режимы работы таймеров 0 и 1

M1	*M0*	*Режим*	*Описание*
0	0	0	13-разрядный счетчик
0	1	1	16-разрядный счетчик
1	0	2	8-разрядный счетчик с автозагрузкой
1	1	3	Таймер 0: два 8-разрядных счетчика. Таймер 1: остановлен

В режиме 0 оба таймера работают в режиме 13-разрядного счета, при этом в регистре TL используется 5 разрядов, а в регистре TH — 8 (рисунок 23). Принято считать, что регистры TL выполняют предварительное деление частоты счетных сигналов на 32.

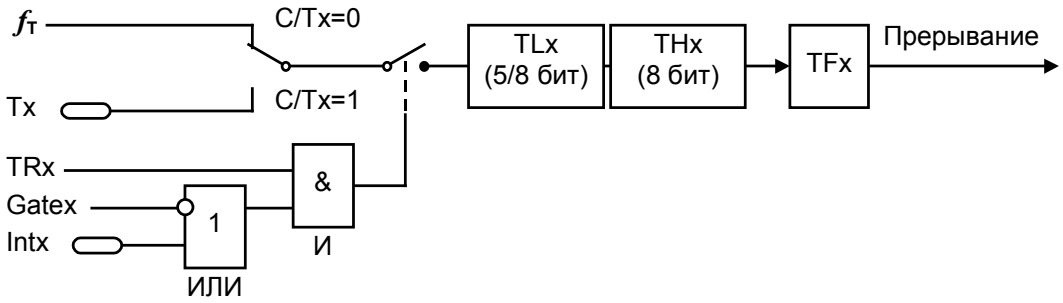


Рисунок 23. Логика работы таймера в режимах 0 и 1

В режиме 1 оба таймера работают в режиме 16-разрядного счета (рисунок 23). В режимах 0 и 1 счетчики TL и TH таймеров при достижении максимального значения счета (8191 для 13-разрядного режима и 65535 для 16-разрядного) переходят в значение 0 с одновременной установкой флагов переполнения TF0 (TF1) в регистре TCON.

В режиме 2 оба таймера работают в режиме 8-разрядного счета с автозагрузкой. По достижении конца счета в регистре TL в него вместо нулевого значения записывается содержимое регистра TH. Это позволяет выполнять счет до произвольных значений, равных 256 —TH (рисунок 24).

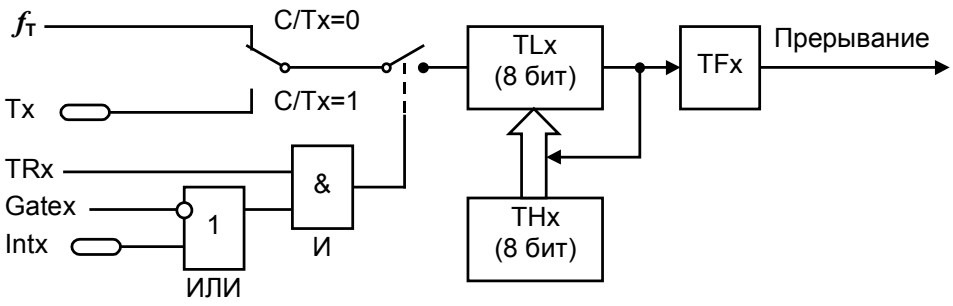


Рисунок 24. Логика работы таймера в режиме 2

В режиме 3 работает только таймер 0. При этом младший регистр счетчика TL0 работает как 8-разрядный таймер-счетчик с битами управления таймера 0, а старший регистр TH0 работает как 8-разрядный счетчик. В этом режиме при достижении конца счета в регистре TL0 устанавливается флаг прерывания TF0, а при достижении конца счета в регистре TH0 устанавливается флаг прерывания TF1 (рисунок 25).

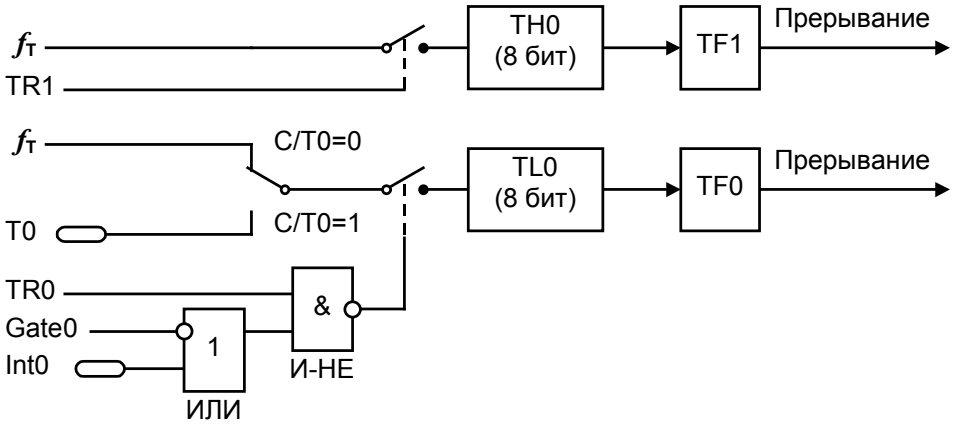


Рисунок 25. Логика работы таймера 0 в режиме 3

Для управления таймерами используется регистр TCON, назначение битов которого приведено в таблице 5. Заметим, что четыре младших бита этого регистра предназначены не для управления таймерами, а для выбора сигналов прерывания, поступающих на выводы INT0 и INT1 микросхемы.

Таблица 5. Регистр управления TCON таймерами 0 и 1

Имя	***Разряд***	***Назначение***
TF1	7	Флаг переполнения таймера 1. Устанавливается аппаратно. Сбрасывается аппаратно при обслуживании прерывания.
TR1	6	Бит управления (запуска) таймера 1. Устанавливается/сбрасывается программно для запуска/останова таймера.
TF0	5	Флаг переполнения таймера 0. Устанавливается аппаратно. Сбрасывается аппаратно при обслуживании прерывания.
TR0	4	Бит управления (запуска) таймера 0. Устанавливается/сбрасывается программно для запуска/останова таймера.
IE1	3	Флаг внешнего прерывания 0. Устанавливается аппаратно при срезе сигнала на выводе INT0 или если сигнал равен нулю, в зависимости от бита IT0. Сбрасывается аппаратно.
IT0	2	Флаг триггера прерывания 0. Если установлен, прерывание 0 возникает при переходе 1–0 на входе INT0. Если сброшен, прерывание 0 возникает при низком уровне INT0.
IE0	1	Флаг внешнего прерывания 1. Устанавливается аппаратно при срезе сигнала на выводе INT1 или если сигнал равен нулю, в зависимости от бита IT1. Сбрасывается аппаратно.
IT0	0	Флаг триггера прерывания 1. Если установлен, прерывание 1 возникает при переходе 1–0 на входе INT1. Если сброшен, прерывание 1 возникает при низком уровне INT1.

Таймер 2

Таймер 2, обозначаемый T/C2, используется как 16-разрядный таймер-счетчик или как генератор частоты приема/передачи. Он управляется регистром T2CON (бит-адресуемый регистр). Назначение битов этого регистра приведено в таблице 6. Кроме него, таймер 2 использует регистры захвата/перезагрузки (RCAP2L и RCAP2H) и регистры счетчика (TL2 и TH2).

Таблица 6. Регистр управления таймером 2

Имя	*Разряд*	*Назначение*
TF2	7	Флаг переполнения. Устанавливается аппаратно, сбрасывается программно. Не устанавливается, если либо RCLK, либо TCLK установлены в 1.
EXF2	6	Внешний флаг таймера 2. Устанавливается аппаратно при защелкивании информации в регистрах захвата или при перезагрузке вследствие перехода 1–0 на выводе P1.1 (T2EX) при EXEN2=1. Сбрасывается программно.
RCLK	5	Разрешение тактовых сигналов приема. Если установлен, таймер 2 используется для тактирования приема в режимах 1 и 3.
TCLK	4	Разрешение тактовых сигналов передачи. Если установлен, таймер 2 используется для тактирования передачи в режимах 1 и 3.
EXEN2	3	Разрешение внешнего сигнала T2EX. Устанавливается для разрешения захвата или перезагрузки вследствие перехода 1–0 на выводе P1.1 (T2EX).
TR2	2	Бит запуска/останова таймера 2.
CNT2	1	Выбор режима работы. Если установлен, таймер 2 работает как счетчик сигналов на выводе T2. Если сброшен, таймер 2 работает как таймер.
CAP2	0	Выбор режима захвата/перезагрузки. Если установлен, разрешен захват по переходу 1–0 на выводе P1.1 при EXEN2=1. Если сброшен, разрешается перезагрузка по переполнению или переходу 1–0 на выводе P1.1 при EXEN2=1. Бит игнорируется, если либо RCLK, либо TCLK установлены в 1.

Режим работы таймера-счетчика 2 устанавливается сочетаниями битов регистра T2CON так, как это показано в таблице 7.

Таблица 7. Режимы работы таймера 2

RCLK или TCLK	*CAP2*	*TR2*	*Режим*
0	0	1	16-разрядный таймер-счетчик с перезагрузкой.
0	1	1	16-разрядный таймер-счетчик с захватом.
1	X	1	Генератор частоты приемопередатчика.
X	X	0	Отключен.

Логика работы таймера 2 в режиме автозагрузки приведена на рисунке 26. 16-битное значение перезагрузки записывается в регистры RCAP2L (младшая часть) и RCAP2H (старшая часть). В регистры TL2 и TH2 перед началом работы записываются те же значения. После окончания счета устанавливается флаг прерывания TF2, который вызывает перезагрузку регистров TL2 и TH2 значениями из регистров RCAP2L и RCAP2H.

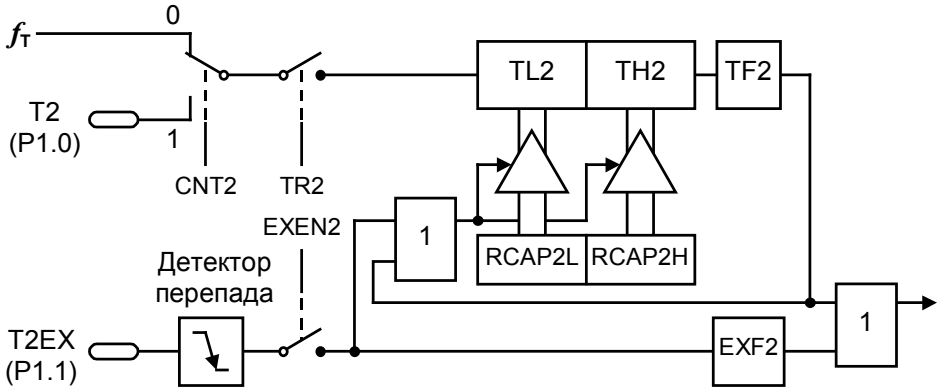


Рисунок 26. Логика работы таймера 2 в режиме автозагрузки

Перезагрузка может быть вызвана также переходом 1⇒0 внешнего сигнала, поступающего на вывод T2EX микросхемы, при этом бит EXEN2 должен быть установлен.

Логика работы таймера 2 в режиме захвата приведена на рисунке 27.

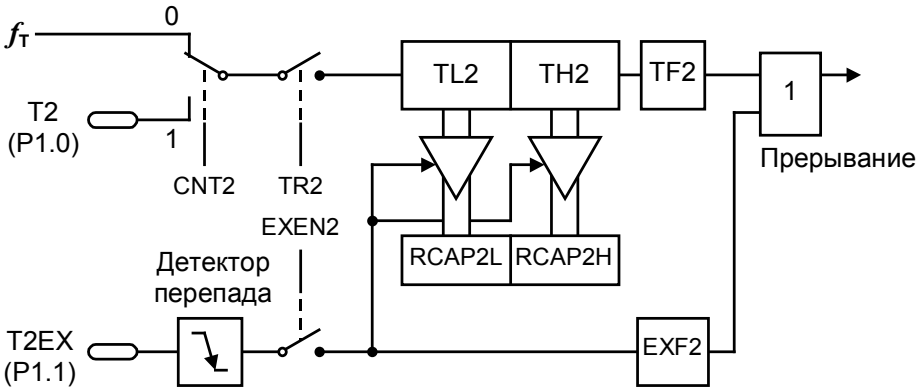


Рисунок 27. Логика работы таймера 2 в режиме захвата

Захват (*capture*) производится переходом 1⇒0 внешнего сигнала, поступающего на вывод T2EX микросхемы, если бит EXEN2 установлен. При этом значения регистров счетчика запоминаются в регистрах захвата RCAP2x. Одновременно в регистре T2CON устанавливается внешний флаг таймера EXF2, который может быть использован для генерирования прерывания так же, как и флаг переполнения TF2.

Таймер 2 используется как генератор частоты приема/передачи в случае, если в регистре T2CON установлен хотя бы один из битов — RCLK или TCLK. В этом случае частота приема/передачи для режимов 0 и 2 рассчитывается по формуле

$f = (f_т / 16) \times f_{ov}$.

Здесь f_{ov} — частота переполнения таймера.

Для режимов 1 и 3 частота приема/передачи рассчитывается по формуле

$f = f_т / (16 \times (65536 — RCAP2))$.

Здесь RCAP2 — 16-разрядное значение, записанное в регистрах RCAP2L и RCAP2H.

Благодаря тому, что таймер 2 использует 16-битовые значения, с его помощью можно получить больший диапазон частот приема/передачи.

Логика работы таймера 2 в режиме генератора частоты приведена на рисунке 28.

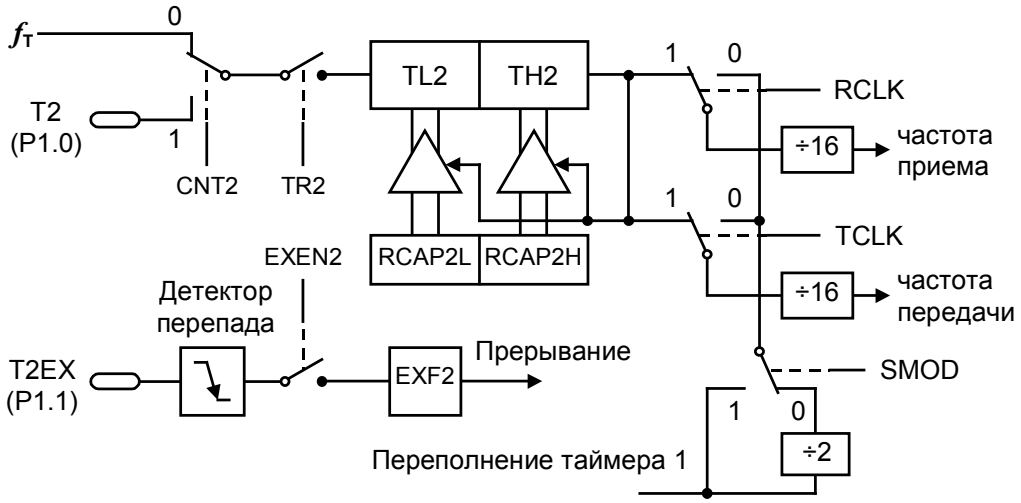


Рисунок 28. Логика работы таймера 2 в режиме генератора частоты

Таймер-генератор

Таймер 3 используется в качестве генератора частоты приема/передачи последовательного порта. Он позволяет получить частоту в большем диапазоне частот с меньшей относительной погрешностью (до 0,8%).

Логика работы таймера 3 приведена на рисунке 29.

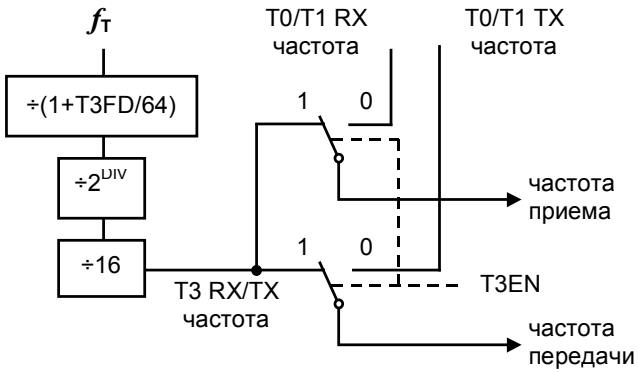


Рисунок 29. Логика работы таймера 3

Для управления таймером 3 используются регистр управления T3CON и регистр дробного делителя частоты T3FD. Назначение разрядов регистра управления T3CON приведено в таблице 8. Значения битов DIV0–DIV2 задают двоичный делитель в соответствии с таблицей 9.

Для настройки частоты приема/передачи F_{UART} прежде всего определяется рабочая частота f_T ядра, значение которой зависит от настройки регистра PLL (о регистре PLL см. ниже).

Таблица 8. Регистр T3CON управления таймером 3

Имя	*Разряд*	*Назначение*
T3BAUDEN	7	Если установлен, таймер используется как генератор частоты приемопередатчика. PCON.7, T2CON.4 и T2CON.5 в этом случае игнорируются.
—	6, 5, 4, 3	Не используется.
DIV2	2	Двоичный делитель частоты.
DIV1	1	то же
DIV0	0	то же

Таблица 9. Делитель DIV

DIV2	*DIV1*	*DIV0*	*Делитель*
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Далее рассчитывается значение двоичного делителя DIV по формуле

$$DIV = log\ (f_T / 16 / F_{UART}) / log\ (2).$$

Полученное значение усекается до меньшего целого значения. Затем рассчитывается дробный делитель FD по формуле

$$FD = 2 \times f_T / 2^{DIV-1} / F_{UART} - 64.$$

Полученное значение округляется. Значение DIV записывается в регистр T3CON, а значение FD — в регистр T2FD.

Для проверки полученных значений рассчитывается фактическая частота приема/передачи по формуле

$$F_{UART\ факт.} = 2 \times f_T / 2^{DIV-1} / (FD + 64).$$

В качестве примера рассчитаем двоичный и дробный делители для частоты приеме/передачи 9600 при частоте ядра 2 МГц. Двоичный делитель:

$$DIV = log\ (2097152 / 16 / 9600) / log(2) = 3{,}77 \approx 3.$$

Дробный делитель:

$$FD = 2 \times 2097152 / 2^2 / 9600 - 64 = 45{,}2 \approx 45 = 2Dh.$$

Фактическая частота приема/передачи:

$$F_{UART\ факт.} = 2 \times 2097152 / 2^2 / (45 + 64) \approx 9620.$$

Полученная относительная погрешность:

$$\delta = (9620 - 9600) / 9600 \approx 0{,}002\ (0{,}2\%).$$

В таблице 10 приведены рассчитанные значения дробного и двоичного делителей для типичных скоростей из предположения, что рабочая частота ядра f_T равна 2 МГц (2097152 Гц).

Таблица 10. Двоичный и дробный делители при частоте ядра 2 МГц

FUART	*DIV*	*T3CON*	*T3FD*
57600	1	81h	09h
38400	1	81h	2Dh
19200	2	82h	2Dh
9600	3	83h	2Dh

Для получения большей скорости приема/передачи (до 230400) следует выбрать большую частоту ядра (см. раздел PLL).

Счетчик временных интервалов

Счетчик временных интервалов TIC предназначен для отсчета интервалов времени от 1/128 секунды до 255 часов. Он тактируется внешним генератором часовой частоты (32768 Гц) и продолжает работать, когда микроконвертер переходит в режим снижения энергопотребления.

Структура счетчика TIC приведена на рисунке 30.

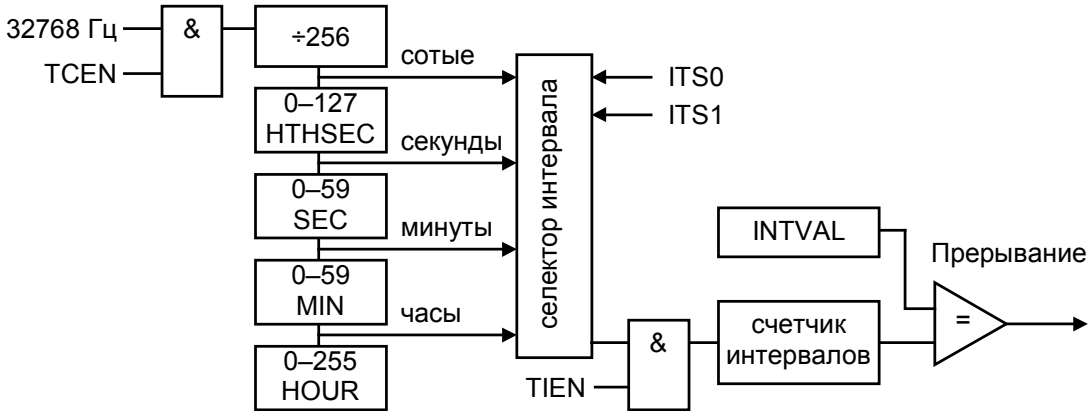


Рисунок 30. Структура счетчика временных интервалов TIC

Счетчик временных интервалов управляется шестью регистрами SFR. Регистр TIMECON является управляющим. Назначение разрядов этого регистра приведено в таблице 11.

Часовая частота предварительно делится на 256 с получением интервалов времени в 1/128 секунды, которые принимаются за 1/100 секунды. Далее эта частота последовательно поступает на последующие счетчики сотых долей (счетчик HTHSEC), секунд (счетчик SEC), минут (счетчик MIN) и часов (счетчик HOUR). С каждым счетчиком связан соответствующий регистр, в который можно записать любое допустимое значение. Это значение складывается с сигналами от предыдущего в цепочке счетчика. При достижении максимального значения счетчик выдает сигнал переполнения и продолжает считать от нуля. Сигнал переполнения поступает на последующий счетчик и на селектор интервала. Селектор передает один из сигналов переполнения на 8-битный счетчик интервалов, если разрешены прерывания. Значение счетчика интервалов сравнивается со значением в регистре INTVAL. При наступлении равенства происходит прерывание.

Таким образом, счетчик TIC может подсчитать до 255 сигналов переполнения.

Таблица 11. Регистр TIMECON управления таймером TIC

Имя	*Разряд*	*Назначение*
—	7	Не используется.
TFH	6	Если установлен, то счетчик часов считает в диапазоне 0–23, а если очищен, то в диапазоне 0–255.
ITS1	5	Выбор частоты подсчета.
ITS0	4	Выбор частоты подсчета.
STI	3	Если установлен, по окончании счета бит TIEN очищается. Если сброшен, по окончании счета таймер автоматически перезагружается.
TII	2	Устанавливается при совпадении значений 8-битного счетчика и регистра INTVAL. Очищается программно.
TIEN	1	Если установлен, разрешает подсчет временных интервалов 8-битным счетчиком.
TCEN	0	Если установлен, разрешает подачу частоты на вход таймера. При очистке запрещает подачу частоты и сбрасывает регистры HTHSEC, SEC, MIN, HOUR в записанные ранее значения. Эти регистры могут быть изменены только при нулевом бите.

Выбор того или иного интервала для подсчета счетчиком интервалов определяется битами ITS0 и ITS1 так, как показано в таблице 12.

Таблица 12. Выбор подсчитываемых интервалов

ITS1	*ITS0*	*Интервал*
0	0	1/128 секунды
0	1	секунды
1	0	минуты
1	1	часы

Последовательный порт

Последовательный порт предназначен для обмена информацией с внешними устройствами по протоколу интерфейса RS-232. Для этой цели в микросхему встроен универсальный асинхронный приемопередатчик (UART — *Universal Asynchronous Receiver-Transmitter*), работающий в дуплексном режиме последовательной передачи (младшими битами вперед).

Управление последовательным портом осуществляется регистрами SCON, SBUF и PCON. В регистре PCON используется только бит SMOD, который увеличивает в два раза скорость передачи.

Регистр SBUF является буфером для передаваемых и принимаемых символов. Для передачи и приема используются дополнительные сдвиговые регистры. При приеме в сдвиговый регистр приема поступают очередные биты передаваемого байта, накапливаясь в нем, поэтому во время приема байт, принятый непосредственно перед этим, все еще находится в буфере SBUF и может быть считан до завершения приема всех битов. Ра

бота управляющей программы происходит параллельно с работой последовательного порта.

Как только в буфер приемопередатчика SBUF помещается символ, начинается его передача. Для переключения приемопередатчика в режим приема в регистре SCON нужно установить бит REN (*Receive Enable*). Байт принимается, если бит RI регистра SCON очищен. Бит RI устанавливается аппаратно при приеме очередного байта и указывает на окончание приема. Бит RI сбрасывается управляющей программой после того, как принятый байт считан из буфера SBUF.

Последовательный порт может работать в одном из 4-х режимов.

Режим 0. Частота передача битов равна f_T / 12. Всего передается 8 бит. Биты передаются через линию, подключенную к выводу RxD, а на выводе TxD формируются сигналы сдвигающей частоты.

Режим 1. Этот режим используется наиболее часто. Частота передачи битов устанавливается программно при помощи таймера 1. Всего передается 10 бит: стартовый нулевой, 8 бит данных, стоповый единичный. Передаваемые биты проходят по выводу TxD, получаемые — по выводу RxD. Стоповый бит при передаче записывается в разряд TB8 регистра SCON и одновременно с его передачей устанавливается флаг прерывания TI (рисунок 31). При приеме стоповый бит записывается в разряд RB8 регистра SCON. Если стоповый бит равен единице, устанавливается бит RI, иначе прием прекращается.

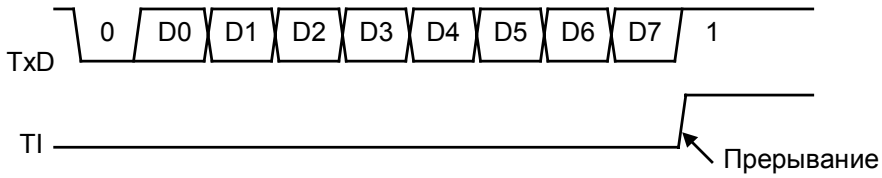


Рисунок 31. Передача в режиме 1

Режим 2. Частота передачи битов равна f_T / 32, если бит SMOD равен нулю, или f_T / 16, если бит SMOD равен единице. Всего передается 11 бит: стартовый нулевой, 8 бит данных, девятый бит, называемый *программируемым*, и стоповый единичный. Программируемый бит чаще всего используется для передачи бита паритета P для контроля правильности передачи. Программируемый бит при передаче записывается в разряд TB8 регистра SCON, а при приеме — в разряд TR8.

Режим 3. Полностью аналогичен режиму 2, но с программируемой частотой приема/передачи.

Назначение разрядов регистра SCON приведено в таблице 13. Режим работы последовательного порта устанавливается сочетанием разрядов SM0 и SM1 регистра SCON в соответствии с таблицей 14. В режимах с программируемой частотой участвует таймер 1. Прерывания таймера 1 должны быть запрещены. Частота передачи/приема рассчитывается по формуле $f = 2^{SMOD} f_{OV} / 32$.

Таблица 13. Регистр управления последовательным портом SCON

Имя	*Разряд*	*Назначение*
SM0	7	Режим Используется вместе с битом 6.
SM1	6	Режим. Используется вместе с битом 7.
SM2	5	Бит запрещения приема, если бит 9 равен 0.
REN	4	Бит разрешения приема.
TB8	3	Программируемый девятый бит передачи.
RB8	2	Программируемый девятый бит приема.
TI	1	Флаг прерывания передачи. Устанавливается аппаратно, сбрасывается программно. Не используется в режиме 0.
RI	0	Флаг прерывания приема. Устанавливается аппаратно, сбрасывается программно. Не используется в режиме 0.

Таблица 14. Режимы работы последовательного порта

SM1	*SM0*	*Режим*	*Описание*
0	0	0	Сдвиговый регистр, постоянная скорость.
0	1	1	8-битовый приемопередатчик, переменная скорость.
1	0	2	9-битовый приемопередатчик, постоянная скорость.
1	1	3	9-битовый приемопередатчик, переменная скорость.

Здесь f_{OV} — частота переполнения таймера. Сам таймер может работать как счетчик или таймер в любом режиме. Наиболее типично использование режима 2 с автозагрузкой предварительно рассчитанной константы BD. В этом случае частота приема/передачи может быть рассчитана по формуле $f = 2^{SMOD} f_T / (32 \times (256 - BD))$.

Константы BD для типичных скоростей приведены в таблице 15.

Таблица 15. Константы автозагрузки для настройки таймера 1

Частота	*fm, МГц*	*SMOD*	*BD*	*256–BD*
19,2 КГц	11,059	1	0FDh	3
9,6 КГц	11,059	0	0FDh	3
4,8 КГц	11,059	0	0FAh	6
2,4 КГц	11,059	1	0F4h	12
1,2 КГц	11,059	0	0F4h	12

АЦП

Микроконвертер ADuC842 содержит в своем составе высокоскоростной прецизионный 8-канальный 12-разрядный аналого-цифровой преобразователь с однополярным питанием, со схемами мультиплексирования (выбора канала), выборки и удерживания, калибровки, а также источник опорного напряжения (ИОН), определяющий диапазон преобразования.

Аналоговые сигналы поступают на вход мультиплексора MUX (рисунок 32), который выбирает один из источников, используя биты выбора канала CS0—CS3 в регистре ADCCON2. Схема выборки и удерживания T/H (track—and—hold) направляет выбранный сигнал на собственно АЦП. АЦП выполняет преобразование аналогового сигнала в 12-разрядное дво

ичное слово, которое записывается в регистры ADCDATAL и ADCDATAH. Формат записи в эти регистры приведен на рисунке 33.

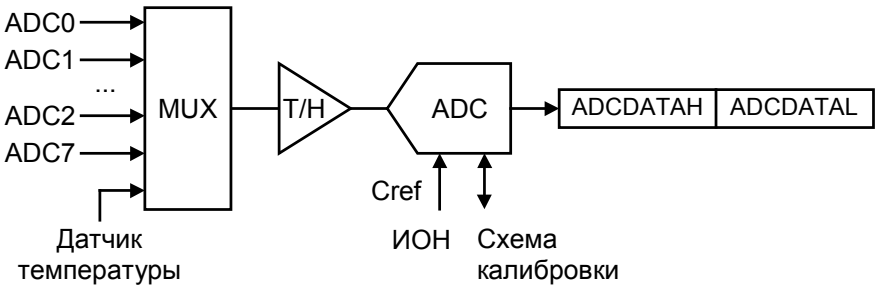


Рисунок 32. Схема АЦП

Далее оцифрованный сигнал при помощи схемы прямого доступа к памяти DMA может быть записан во внешнюю память данных в том формате, в котором он хранится в регистрах АЦП.

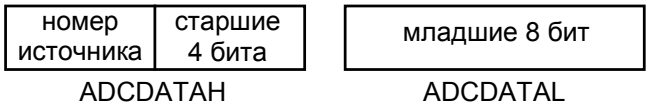


Рисунок 33. Формат записи результата преобразования

Для управления АЦП используется три управляющих регистра ADCCON1, ADCCON2, ADCCON3, два регистра данных ADCDATAL и ADCDATAH, два регистра калибровочного коэффициента по усилению ADCGAINL и ADCGAINH, а также регистры калибровочного коэффициента по смещению нуля ADCOFSL и ADCOFSH.

Регистр ADCCON1 управляет запуском преобразования. Назначение разрядов регистра приведено в таблице 16.

Таблица 16. Регистр ADCCON1 управления АЦП

Имя	*Разряд*	*Назначение*
MD1	7	Устанавливается для выполнения преобразования.
EXT_REF	6	Если 1, используется внешнее опорное напряжение.
CK1	5	Выбор тактовой частоты АЦП.
CK0	4	то же
AQ1	3	Выбор времени задержки схемы T/H.
AQ0	2	то же
T2C	1	Если 1, таймер 2 запускает цикл преобразования.
EXC	0	Если 1, низкий уровень длительностью более 100 нс на выводе P3.5 запускает цикл преобразования.

Запустить преобразование можно программно при помощи бита MD1, сигналом переполнения таймера 2, и при помощи сигнала CONVST, который подается на вывод P3.5. Дополнительно в этом регистре выбирается делитель для получения тактовой частоты преобразования CK (таблица 17), и время задержки для распознавания сигнала AQ в тактах (таблица 18).

Таблица 17. Выбор тактовой частоты преобразования

CK1	CK0	Делитель
0	0	32
0	1	4 (не использовать с CD=0 в PLLCON)
1	0	8
1	1	2

Таблица 18. Выбор задержки сигнала

AQ1	AQ0	Задержка
0	0	1
0	1	2
1	0	3
1	1	4

Регистр ADCCON2 предназначен для выбора источника сигнала а также режима преобразования. Назначение разрядов регистра приведено в таблице 19.

Таблица 19. Регистр ADCCON2 управления АЦП

Имя	Разряд	Назначение
ADCI	7	Устанавливается аппаратно по завершении однократного преобразования или блока преобразований с DMA. Сбрасывается программно или аппаратно при обработке прерывания.
DMA	6	Разрешение преобразования с прямым доступом к памяти.
CCONV	5	Разрешение циклического преобразования.
SCONV	4	Разрешение однократного преобразования.
CS3	3	Выбор источника сигнала преобразования.
CS2	2	то же
CS1	1	то же
CS0	0	то же

АЦП может выполнять однократные и циклические преобразования. Однократное преобразование выполняется при установленном бите SCONV (*Single Conversion*). Результат однократного преобразования записывается в регистры ADCDATAH/L. По окончании преобразования устанавливается флаг прерывания ADCI. Время одного преобразования составляет 16 тактов частоты, поступающей на АЦП, плюс число тактов задержки (таблица 18).

Циклическое преобразование выполняется при установленном бите CCONV (*Continuous Conversion*). В этом режиме АЦП непрерывно преобразует входной аналоговый сигнал с частотой, установленной битами CK и AQ в регистре ADCCON1. Следует иметь в виду, что максимальная частота преобразования отводит управляющей программе на обслуживание одного преобразования всего 2 мкс (максимум 32 машинных цикла), включая время обслуживания прерывания. Поэтому АЦП может работать в режиме прямого доступа к памяти (DMA), в котором результаты преобразования записываются во внешнюю память данных без участия процессора и обрабатываются позднее.

Таблица 20. Выбор источника аналогового сигнала

CS3	CS2	CS1	CS0	Источник
0	0	0	0	ADC0
0	0	0	1	ADC1
0	0	1	0	ADC2
0	0	1	1	ADC3
0	1	0	0	ADC4
0	1	0	1	ADC5
0	1	1	0	ADC6
0	1	1	1	ADC7
1	0	0	0	Датчик температуры (1 мкс для распознавания).
1	0	0	1	DAC0 (при включенном выходном буфере).
1	0	1	0	DAC1 (при включенном выходном буфере).
1	0	1	1	AGND (аналоговая «земля»).
1	1	0	0	Vref (опорное напряжение).
1	1	1	1	Закончить преобразование DMA.

Регистр ADCCON3 используется для калибровки АЦП. Назначение разрядов регистра приведено в таблице 21.

Таблица 21. Регистр ADCCON3 управления АЦП

Имя	Разряд	Назначение
BUSY	7	Бит занятости. Только чтение. Если 1, АЦП выполняет преобразование или калибровку.
—	6	Не используется.
AVGS1	5	Количество выборок во время калибровки.
AVGS0	4	то же
—	3, 2	Не используется.
TYPICAL	1	Тип калибровки — по смещению (0) или по усилению (1).
SCAL	0	Запуск цикла калибровки.

Микросхема поставляется потребителю откалиброванной. Это означает, что результат преобразования скорректирован для уменьшения влияния дрейфа (медленного изменения) постоянных токов в цепях АЦП.

Тем не менее, при эксплуатации устройства может понадобиться его калибровка для повышения точности измерений. Во время цикла калибровки устанавливается преобразование аналогового нуля или опорного напряжения с тем, чтобы результат преобразования тоже был равен нулю или опорному напряжению. Калибровка выполняется несколькими выборками. Количество выборок определяется битами AVGSx по таблице 22.

За один цикл выполняется калибровка либо смещения нуля, либо коэффициента усиления передаточной характеристики. Выбор типа калибровки выполняется битом TYPICAL. Сначала должна выполняться калибровка нуля при подаче на вход сигнала нуля. Затем выполняется калибровка усиления с подачей на вход сигнала опорного напряжения Vref. Калибровочные коэффициенты имеют разрядность 14 бит, 8 младших бит в ре

гистре (ADCOFSL или ADCGAINL), 6 старших бит в регистре (ADCOFSH или ADCGAINH).

Таблица 22. Выбор числа выборок калибровки

AVGS1	*AVGS0*	*Число выборок*
0	0	15
0	1	1
1	0	31
1	1	63

Преобразование аналогового сигнала в цифровой осуществляется в диапазоне от 0 до Vref, которое составляет 2,5 В при использовании внутреннего ИОН. При этом точность измерения составляет $Vref / 2^{12} = 0{,}61$ мВ. Значение преобразования можно вычислить, если умножить точность преобразования на результат преобразования. Так, если результат преобразования равен 964, измеряемое напряжение равно 0,588 В. Максимальное значение преобразования 4095 соответствует опорному напряжению.

Выбор канала преобразования в случае DMA осуществляется предварительной записью номеров каналов в память (рисунок 34).

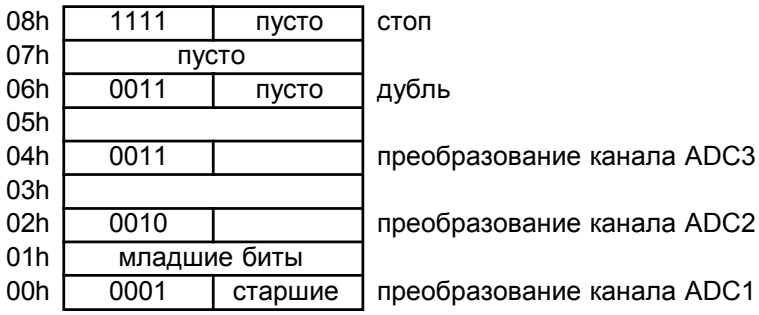


Рисунок 34. Разметка памяти для блока из 3-х выборок с использованием DMA

Схема DMA считывает из памяти номер канала, схема АЦП выполняет преобразование, и схема DMA записывает результат. Для завершения преобразований следует дважды записать номер последнего преобразования и 1111 (стоп). По окончании заполнения блока выборок и разрешенных прерываниях от АЦП вырабатывается прерывание с вектором 33h.

При использовании циклического преобразования в режиме DMA процессор не участвует в преобразовании и записи значений, и продолжает выполнять управляющую программу. Однако следует иметь ввиду, что если для записи выборок используется внешняя, а не внутренняя память данных, порты 0 и 2 недоступны для использования.

Аналоговые сигналы ADC0–ADC7 поступают на вход мультиплексора через выводы микросхемы, соответствующие порту 1 (иначе говоря, через порт 1). Для настройки этого порта как источника аналоговых сигналов в него нужно записать значение 0FFh. При записи в этот порт нулевых значений разряды порта 1 используются как цифровые входы обычным образом. Порт 1 не может быть использован для вывода сигналов.

При использовании для АЦП внешнего опорного напряжения последнее должно находиться в пределах 1 В — AV_{DD} (AV_{DD} — напряжение аналогового питания). Соответственно изменяется и величина дискретизации (точность) измерения.

ЦАП

Микроконвертер ADuC842 оснащен двумя цифро-аналоговыми преобразователями (ЦАП). Цифро-аналоговый преобразователь формирует выходное аналоговое напряжение в соответствии с цифровым значением, записанным в его регистры данных. Выходное напряжение лежит в диапазоне 0—Vref или 0—AVDD. При использовании диапазона 0—Vref АЦП должен быть включен.

Для управления ЦАП используется регистр DACCON. Назначение разрядов этого регистра приведено в таблице 23. Преобразуемое значение записывается в пару регистров данных DAC0L и DAC0H или DAC1L и DAC1H. Результат преобразования появляется на выводах микросхемы DAC0 или DAC1.

Таблица 23. Регистр DACCON управления ЦАП

Имя	*Разряд*	*Назначение*
MODE	7	Если 1, преобразование 8-битное, иначе 12-битное.
RNG1	6	Если 1, выходной диапазон канала 1 0–AVDD, иначе 0–Vref.
RNG0	5	Если 1, выходной диапазон канала 0 0–AVDD, иначе 0–Vref.
CLR1	4	Если 1, выход канала 1 соответствует коду, иначе равен 0 В.
CLR0	3	Если 1, выход канала 0 соответствует коду, иначе равен 0 В.
SYNC	2	Бит синхронизации.
PD1	1	Если 1, канал 1 включен.
PD0	0	Если 1, канал 0 включен.

При 8-битном преобразовании используются только регистры DACxL. При 12-битном преобразовании в регистрах DACxH используются младшие 4 бита.

Если бит SYNC установлен, выходное напряжение обновляется сразу после того, как будет изменен младший регистр (DACxL). Поэтому в этом режиме, называемом асинхронным, сначала записывается старшая часть двоичного слова в регистр DACxH, а затем младшая в регистр DACxL.

Если бит SYNC не установлен, значения регистров данных можно изменять произвольно. Выходное напряжение в этом случае появляется на выходах DAC0 и DAC1 после установки бита SYNC.

ШИМ

Широтно-импульсный модулятор (ШИМ, PWM — *Pulse-Width Modulator*) предназначен для формирования последовательности импульсов переменной ширины. Схема ШИМ тактируется при помощи одного из четырех источников тактовых сигналов, которые дополнительно могут быть

поделены на программируемый делитель CDIV, и формирует выходные импульсы в одном из шести режимов. Схема ШИМ приведена на рисунке 35.

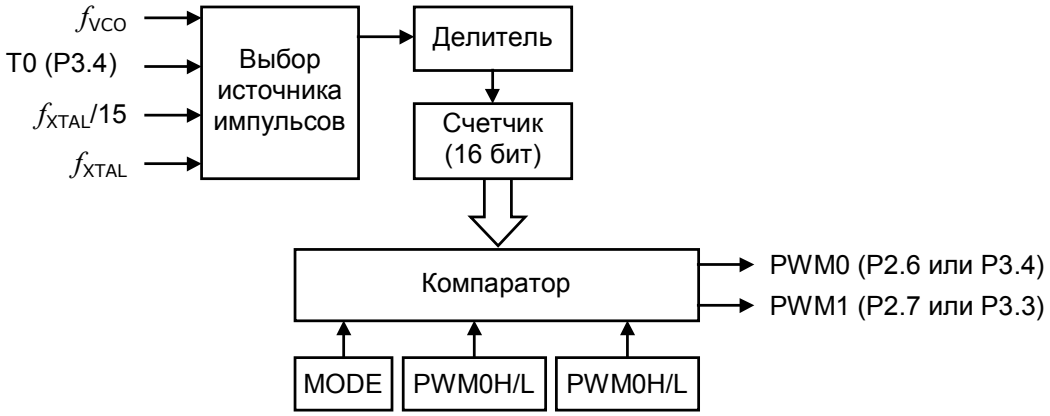


Рисунок 35. Схема ШИМ

Схема ШИМ управляется при помощи регистра PWMCON, назначение разрядов которого приведено в таблице 24.

Таблица 24. Регистр PWMCON управления ШИМ

Имя	***Разряд***	***Назначение***
SNGL	7	Отключает выход ШИМ в порты для операций с ними.
MD2	6	Выбор режима работы.
MD1	5	то же
MD0	4	то же
CDIV1	3	Выбор делителя тактовых сигналов.
CDIV0	2	то же
CSEL1	1	Выбор источника тактовых сигналов.
CSEL0	0	то же

Выбор тактовой частоты приведен в таблице 25.

Таблица 25. Выбор источника тактовых сигналов ШИМ

CSEL1	***CSEL0***	***Источник тактовых сигналов***
0	0	Частота на выводах XTAL, деленная на 15 (2184,5 Гц).
0	1	Частота на выводах XTAL (32768 Гц).
1	0	Внешняя частота на выводе P3.4 (T0).
1	1	Исходная частота PLL (16777216 Гц).

Выбор делителя тактовой частоты приведен в таблице 26.

Таблица 26. Выбор делителя частоты тактовых сигналов ШИМ

CDIV1	***CDIV0***	***Делитель***
0	0	1
0	1	4
1	0	16
1	1	64

Выбор режима ШИМ приведен в таблице 27.

Таблица 27. Выбор режима работы ШИМ

MD2	*MD1*	*MD0*	*Режим*
0	0	0	ШИМ отключен.
0	0	1	Режим 1. Одиночный ШИМ переменного разрешения.
0	1	0	Режим 2. Двойной 8-битный ШИМ.
0	1	1	Режим 3. Двойной 16-битный ШИМ.
1	0	0	Режим 4. Двойной NRZ 16-битный Σ-Δ ЦАП.
1	0	1	Режим 5. Двойной 8-битный ШИМ.
1	1	0	Режим 6. Двойной RZ 16-битный Σ-Δ ЦАП.
1	1	1	Зарезервировано.

Длительность импульсов задается регистрами данных PWM0H/L и PWM1H/L. В цикле преобразования счетчик ШИМ сравнивается со значениями в регистрах данных для определения моментов изменения сигналов на выходе ШИМ.

При программировании прежде всего нужно установить источник тактирующих сигналов и режим работы. При этом происходит сброс счетчика. В 16-битных режимах (1, 3, 4 и 6) сначала следует записывать регистры PWMxL. Значения запоминаются в скрытых регистрах. При последующей записи в регистры PWMxH происходит обновление регистров PWMxL и новые значения используются в последующих циклах.

Выбор выходов ШИМ PWM0/PWM1 на выводы P2.6/P2.7 или на выводы P3.4/P3.3 производится с помощью регистра конфигурации CFG842.

Режим 1

В режиме 1 (рисунок 36) схема работает как 16-битный ШИМ с переменным разрешением (один канал). Длительность импульсов определяется значением в регистрах PWM1H/L, а ширина — значением в регистрах PWM0H/L.

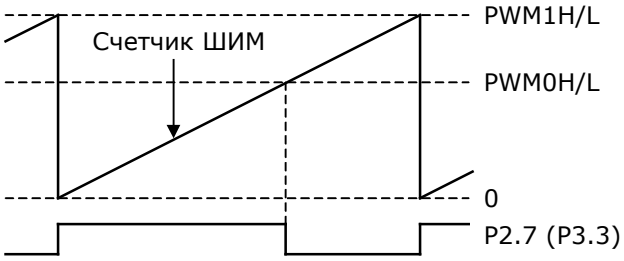


Рисунок 36. ШИМ в режиме 1

Например, при тактирующей частоте 16777216 Гц и значении в регистрах PWM1H/L, равном 65535, получим длительность 3,9 мс при частоте циклов 256 Гц (16777216/65536). Если в регистры PWM1H/L записать значение 4095, получим длительность 244 мс при частоте 4096 Гц.

Режим 2

В режиме 2 схема работает как два 8-битных ШИМ с программируемой длительностью и шириной импульсов (рисунок 37).

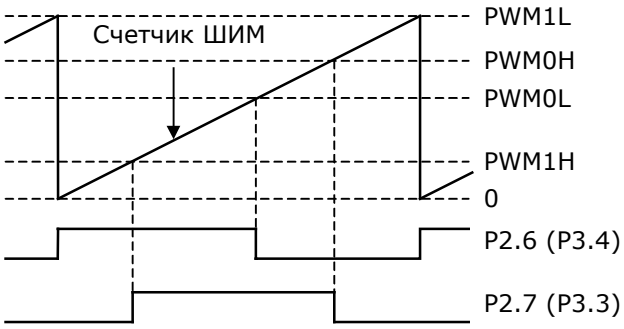


Рисунок 37. ШИМ в режиме 2

Значение в регистре PWM1L определяет длительность импульсов в обоих каналах. Обычно в него записывается 255, но при необходимости можно записать и меньшее значение. Например, записав в PWM1L значение 100, получим ШИМ с точностью отсчета в 1%.

Канал 0 переходит в 0 при равенстве счетчика и регистра PWM0L. Канал 1 переходит в 1 при равенстве счетчика и PWM1H, и в 0 при равенстве счетчика и PWM0H. При записи 0 в регистр PWM1H оба канала переходят в 1 одновременно.

Режим 3

В режиме 3 счетчик ШИМ считает от 0 до 65535, задавая постоянную длительность импульсов $65536/f_T$ (рисунок 38).

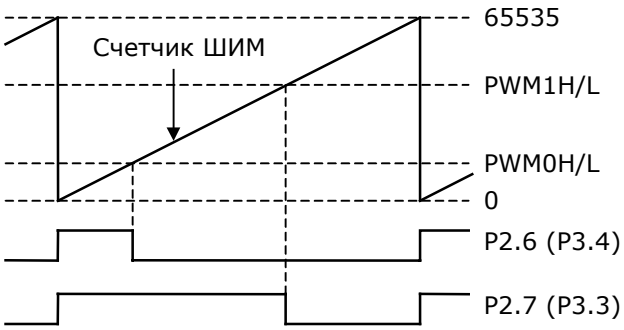


Рисунок 38. ШИМ в режиме 3

Ширина импульсов канала 0 задается регистрами PWM0H/L, а ширина импульсов в канале 1 — регистрами PWM1H/L. При переходе счетчика через 0 оба канала переходят в состояние 1.

Режим 4

В режиме 4 ШИМ работает примерно по той же схеме, что и цифро-аналоговый преобразователь. Обычно в этом режиме используется частота тактирования 16777216 Гц. Выходы каналов обновляются в каждом периоде частоты тактирования, то есть через 60 нс. В течение 65536 тактов вы

ход канала имеет высокий уровень такое количество циклов, какое записано в регистрах PWM0H/L или PWM1H/L.

Например, если в регистры PWM0H/L записать 4010h (16400), что немного больше 1/4 максимального FFFFh (65535), выход канала 0 будет иметь низкий уровень в течение 3 периодов и высокий уровень в течение 1 периода. Последовательность выходных импульсов в этом случае показана на рисунке 39.

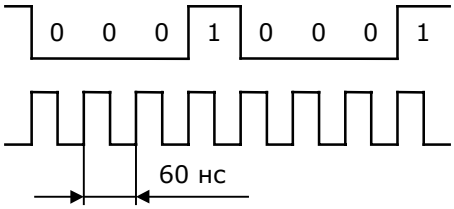


Рисунок 39. Выход канала ШИМ в режиме 4

При этом образуется 16 «недостающих» периодов с высоким уровнем, так как при соотношении 1:4 получится только 16384 таких периода. Это компенсируется меньшим на 1 количеством последовательных периодов с низким уровнем в конце серии из 65536 периодов. В этом случае за периодом с высоким уровнем будет следовать не три, а два периода с низким уровнем.

Чтобы получить более быстрое преобразование при меньшем разрешении, следует записывать нули в младшие разряды регистров данных.

Режим 5

В режиме 5 схема работает как два независимых 8-разрядных ШИМ с программируемой длительностью и шириной импульсов (рисунок 40).

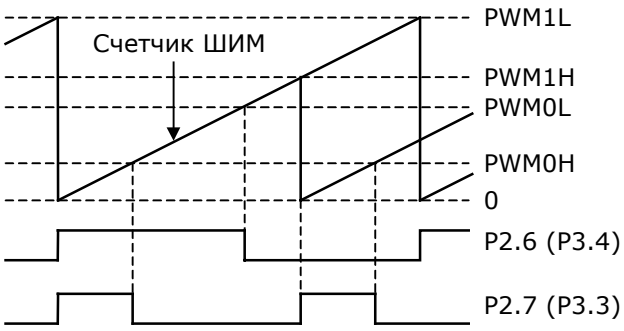


Рисунок 40. ШИМ в режиме 5

Длительность импульсов канала 0 задается регистром PWM1L, ширина импульсов — регистром PWM0L. Для канала 1 используются регистры PWM1H и PWM0H соответственно.

Режим 6

В режиме 6 схема работает так же, как и в режиме 4, за исключением того, что в каждом периоде выходные импульсы с высоким уровнем переходят в низкий уровень (RZ — *return to zero*, возврат к нулю). Вид выход

ных импульсов для тех же параметров, что были рассмотрены при описании режима 4, приведен на рисунке 41.

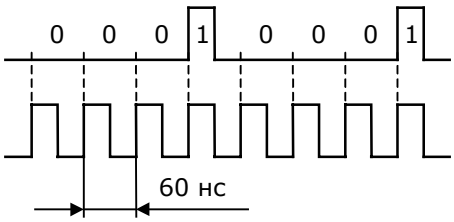


Рисунок 41. Выход канала ШИМ в режиме 6

При этом для компенсации недостающих периодов с высоким уровнем в конце серии из 65536 периодов формируются четверки периодов с двумя высокими уровнями вместо одного. Следует иметь ввиду, что из-за того, что в каждом периоде происходит возврат к нулю, суммарное выходное напряжение каналов в этом режиме вдвое меньше, чем в режиме 4.

Двойной указатель данных

В микроконвертере ADuC842 имеется два набора регистров для обращения к внешней памяти данных (два DPTR). Второй регистр DPTR называется теневым. DPTR управляются регистром DPCON, назначение разрядов которого приведено в таблице 28.

Таблица 28. Регистр DPCON управления указателями данных

Имя	*Разряд*	*Назначение*
—	7	Не используется.
DPT	6	Если установлен, регистры автоматически меняются местами после каждой инструкции MOVC или MOVX.
DP1m1	5	Режим теневого регистра DPTR.
DP1m0	4	то же
DP0m1	3	Режим основного регистра DPTR.
DP0m0	2	то же
—	1	Не используется для того, чтобы инструкция INC DPCON переключала DPTR, не изменяя остальные биты.
DPSEL	0	Если сброшен, используется основной DPTR, а если установлен, то теневой.

Для каждого из регистров DPTR можно раздельно установить один из четырех режимов работы, приведенных в таблице 29.

Данная организация позволяет упростить блочные пересылки данных. При выборе режима инкремента (декремента) соответствующий DPTR увеличивается (уменьшается) на единицу после каждой инструкции пересылки данных типа MOVC или MOVX, в которых используется DPTR. Кроме того, есть возможность автоматически менять используемый для адресации DPTR после каждой инструкции (если установлен разряд DPT).

Таблица 29. Выбор режима DPTR

m1	*m0*	*Делитель*
0	0	Режим совместимости с 8052.
0	1	DPTR инкрементируется после инструкции пересылки.
1	0	DPTR декрементируется после инструкции пересылки.
1	1	Младший бит инвертируется после инструкции пересылки.

PLL

Микроконвертер ADuC842 тактируется при помощи внешнего кварцевого резонатора часовой частоты 32768 Гц. Схема PLL умножает эту частоту на 512 с получением частоты 16777216 Гц (обозначаемая здесь как 16 МГц). Далее эта частота делится схемой PLL для получения частоты ядра $f_Т$.

Целочисленный делитель CD определяется значениями битов CD0–CD2 в регистре PLLCON, управляющим схемой PLL. Назначение разрядов этого регистра приведено в таблице 30. Частота ядра, которая может быть получена при различных значениях битов CD0–CD2, приведена в таблице 31.

Таблица 30. Регистр PLLCON управления частотой ядра

Имя	*Разряд*	*Назначение*
OSC_PD	7	Если установлен, генерация частоты останавливается (режим Power Down). Таймер TIC продолжает считать.
LOCK	6	Только для чтения. Устанавливается схемой PLL, если часовая частота обнаружена при старте. Сбрасывается схемой PLL, если часовая частота не обнаружена. В этом случае частота 16 МГц генерируется схемой PLL с точностью 20%.
—	5, 4	Не используется.
FINT	3	Если установлен, обслуживание прерывания происходит на частоте 16 МГц, независимо от установленной частоты ядра. После обслуживания прерывания частота ядра восстанавливается.
CD2	2	Делитель частоты.
CD1	1	то же
CD0	0	то же

Таблица 31. Частота ядра

CD2	*CD1*	*CD0*	*Делитель*	*Частота ядра*
0	0	0	1	16 777 216 Гц
0	0	1	2	8 388 608 Гц
0	1	0	4	4 194 304 Гц
0	1	1	8	2 097 152 Гц (по умолчанию)
1	0	0	16	1 048 576 Гц
1	0	1	32	524 288 Гц
1	1	0	64	262 144 Гц
1	1	1	128	131 072 Гц

Сторожевой таймер

Сторожевой таймер (WDT — *watchdog timer*) вызывает системный сброс (*reset*) или прерывание после того, как управляющая программа перейдет в ошибочное состояние, например, вследствие зацикливания или сильного электрического импульса. Ошибочное состояние определяется, если в течение заданного интервала времени управляющая программа не установила бит WDE управляющего регистра WDCON. Назначение разрядов регистра WDCON приведено в таблице 32.

Таблица 32. Регистр WDCON управления сторожевым таймером

Имя	Разряд	Назначение
PRE3	7	Установка сторожевого интервала.
PRE2	6	то же
PRE1	5	то же
PRE0	4	то же
WDIR	3	Если 1, генерируется прерывание, если 0 — сброс.
WDS	2	Устанавливается таймером по истечении интервала.
WDE	1	Бит разрешения.
WDWR	0	Бит разрешения записи в регистр управления.

Сторожевой интервал устанавливается битами PRE0—3 в соответствии с таблицей 33. Бит PRE3 зарезервирован и должен быть равен нулю.

Таблица 33. Выбор сторожевого интервала

PRE2	PRE1	PRE0	Интервал (мс)
0	0	0	15,6
0	0	1	31,2
0	1	0	62,5
0	1	1	125
1	0	0	250
1	0	1	500
1	1	0	1000
1	1	1	2000

Запись в регистр WDCON должна осуществляться двумя последовательными инструкциями, как показано в следующем примере:

```
CLR   EA          ; запрещение прерываний
SETB  WDWR        ; разрешение записи в WDCON
MOV   WDCON,#72h  ; разрешение отслеживания интервала в 2 сек
SETB  EA          ; разрешение прерываний
```

Монитор питания

Монитор питания PSM предназначен для отслеживания напряжения питания микросхемы. Назначение разрядов управляющего регистра PSMCON приведено в таблице 34.

Таблица 34. Регистр PSMCON управления монитором питания

Имя	*Разряд*	*Назначение*
CMPD	6	Только для чтения. Если 1, то напряжение питания больше точки отслеживания, если 0, то меньше.
PSMI	5	Бит прерывания. Устанавливается аппаратно и программно.
TPD1	4	Если TPD0=0, а TPD1=1, точка отслеживания 2,93 В.
TPD0	3	Если TPD0=1, а TPD1=0, точка отслеживания 3,08 В.
—	2, 1	Не используется.
PSMEN	0	Бит разрешения.

Начальное значение регистра PSMCON равно 0DEh. Разряд 7 не используется. Монитор питания работает только для микросхем с номинальным напряжением цифрового питания DV_{DD}, равным 3 В. При снижении напряжения питания ниже точки отслеживания в течение времени более 250 мс возникает прерывание с вектором 43h.

Регистр конфигурации

Назначение разрядов регистра CFG842 приведено в таблице 35.

Таблица 35. Регистр конфигурации

Имя	*Разряд*	*Назначение*
EXSP	7	Если 1, используется расширенный указатель стека.
PWPO	6	Если 1, ШИМ использует выводы P3.4/P3.3, иначе P2.6/P2.7.
DBUF	5	Если 1, выходной буфер ЦАП отключен.
EXTCLK	4	Если 1, используется внешний тактирующий сигнал на P3.4.
—	3, 2	Не используется. Всегда должно быть равно 0.
MSPI	1	Если 1, интерфейс SPI использует выводы P3.3 (MISO), P3.4 (MOSI) и P3.5 (SCLOCK).
XRAMEN	0	Если 1, внутренняя внешняя память данных используется и занимает нижние 2 Кбайта адресного пространства.

Начальное значение регистра равно 0.

Регистр идентификации

Регистр CHIPID предназначен для идентификации микросхемы. Он работает только на чтение и возвращает код A6h для микросхемы ADuC842 с объемом памяти программ 62 Кбайта.

Система прерываний

Схематически схема прерываний изображена на рисунке 42. В системе 11 уровней прерываний от 14 источников. Каждому уровню соответствует адрес в памяти программ — вектор прерывания. При возникновении прерывания управление передается по адресу вектора прерывания. Для управления системой прерываний используется регистры IE, IP и IEIP2.

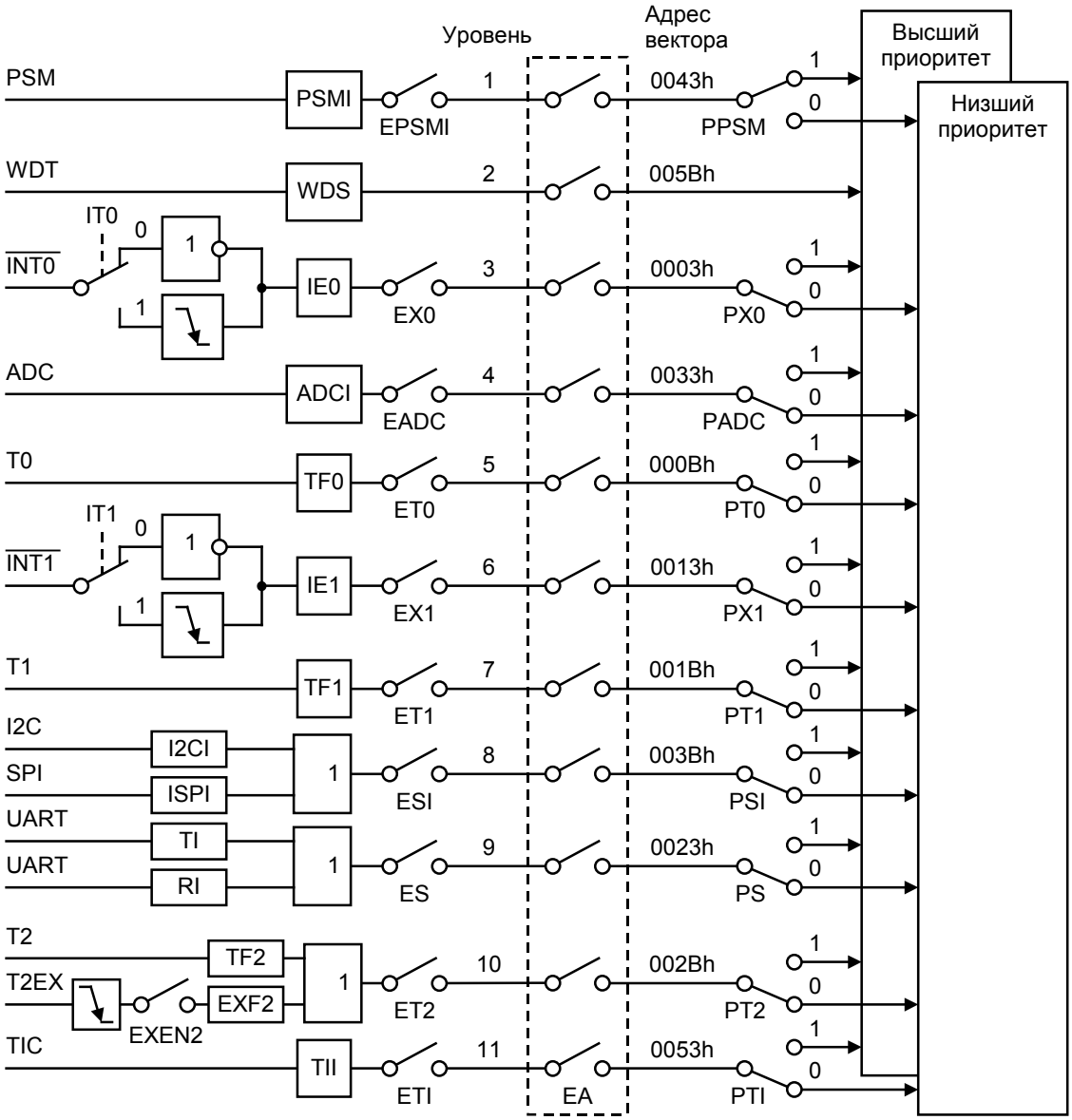


Рисунок 42. Схема системы прерываний

Разряды регистра IE (*Interrupt Enable*) управляют разрешением основных прерываний. Назначение битов этого регистра приведено в таблице 36.

Бит 7 регистра IE разрешает или запрещает прерывания вообще. Если он не установлен, все прерывания запрещены. Если бит EA установлен, то прерывания могут возникать по тем или иным событиям, только если установлен бит, разрешающий прерывания по данному событию.

Каждому уровню прерывания соответствует также один из двух уровней приоритета — низший и высший. Текущий приоритет данного уровня определяется регистром IP (*Interrupt Priority*). Назначение разрядов этого регистра приведено в таблице 37.

Для маскирования дополнительных уровней прерывания и задания их приоритетов в микроконвертере используются дополнительный регистр IEIP2. Назначение разрядов регистра IEIP2 приведено в таблице 38.

Таблица 36. Регистр масок прерываний IE микроконвертера ADuC842

Имя	*Разряд*	*Назначение*
EA	7	Разрешение прерываний. Если установлен, прерывания разрешены. Если сброшен, все прерывания запрещены.
EADC	6	Разрешение прерываний АЦП.
ET2	5	Разрешение прерываний по переполнению таймера 2.
ES	4	Разрешение прерывания приемопередатчика.
ET1	3	Разрешение прерываний по переполнению таймера 1.
EX1	2	Разрешение прерывания по внешнему сигналу INT1.
ET0	1	Разрешение прерываний по переполнению таймера 0.
EX0	0	Разрешение прерывания по внешнему сигналу INT0.

Таблица 37. Регистр приоритетов прерываний IP микроконвертера ADuC842

Имя	*Разряд*	*Назначение*
—	7	Не используется.
PADC	6	Приоритет прерывания АЦП.
PT2	5	Приоритет прерывания таймера 2.
PS	4	Приоритет прерывания приемопередатчика.
PT1	3	Приоритет прерываний по переполнению таймера 1.
PX1	2	Приоритет прерывания по внешнему сигналу INT1.
PT0	1	Приоритет прерываний по переполнению таймера 0.
PX0	0	Приоритет прерывания по внешнему сигналу INT0.

Таблица 38. Дополнительный регистр прерываний IEIP2

Имя	*Разряд*	*Назначение*
—	7	Не используется.
PTI	6	Приоритет прерывания счетчика TIC.
PPSM	5	Приоритет прерывания монитора питания.
PSI	4	Приоритет прерывания интерфейса SPI/I2C.
—	3	Должен содержать 0.
ETI	2	Разрешение прерывания таймера TIC.
EPSMI	1	Разрешение прерывания монитора питания.
ESI	0	Разрешение прерывания интерфейса SPI/I2C.

Начальное значение регистра IEIP2 равно A0h.

События, приводящие к возникновению прерываний, могут возникать в произвольные моменты времени. Однако момент возникновения прерывания привязан к окончанию машинного цикла, завершающего исполнение машинной инструкции. Иначе говоря, прерывания откладываются до окончания исполнения текущей команды.

Прежде всего система прерываний определяет, не находится ли она в состоянии блокировки. Блокировка устанавливается в момент входа в прерывание, то есть в начале обслуживания прерывания. При этом могут возникать различные ситуации из-за наличия прерываний разных приоритетов.

Если система находится в состоянии блокировки по прерыванию от события низшего приоритета, то прерывание разрешается (возникает), если событие имеет высший приоритет, и запрещается (откладывается), если событие имеет низший приоритет. Например, если система обслуживает окончание приема байта по последовательному порту, которому назначен низший приоритет, и возникает переполнение таймера 0, также имеющее низший приоритет, то прерывание по переполнению таймера откладывается. Если же таймеру 0 назначен высший приоритет, то обслуживание приемопередатчика приостанавливается и возникает прерывание по переполнению таймера 0.

Если система находится в состоянии блокировки по прерыванию от события, имеющего высший приоритет, то никакое событие, независимо от его приоритета, не может привести к повторному прерыванию.

К моменту окончания исполнения текущей машинной инструкции возможно также возникновение двух и более прерываний, которые считаются одновременными. В этом случае из множества прерываний возникает то, которое имеет более высокий уровень, с учетом приоритетов. Сначала проверяется наличие запроса на прерывание высшего приоритета, затем низшего. Блокировка прерываний в этом случае также имеет место и влияет на обслуживание или откладывание прерывания.

В таблице 39 приведен порядок опроса прерываний одного уровня.

Таблица 39. Приоритет прерываний одного уровня

Источник	*Приоритет*	*Вектор*	*Описание*
PSMI	1 (высший)	43h	Монитор питания
WDS	2	5Bh	Сторожевой таймер
IE0	3	03h	Внешнее прерывание 0
ADCI	4	33h	АЦП
TF0	5	0Bh	Таймер 0
IE1	6	13h	Внешнее прерывание 1
TF1	7	1Bh	Таймер 1
ISPI/I2CI	8	3Bh	Интерфейс SPI/I2C
RI+TI	9	23h	Приемопередатчик
TF2+EXF2	10	2Bh	Таймер 2
TII	11 (низший)	53h	Счетчик временных интервалов TIC

Отложенное прерывание ожидает момента, когда блокировка будет снята. Блокировка снимается инструкцией RETI, которая обязательно должна завершать подпрограмму обслуживания прерывания. Ошибочное применение инструкции RET приводит к постоянной блокировке прерываний, которое также может привести к отказу нормальной работы системы прерываний.

Следует также иметь ввиду, что отложенное прерывание может быть снято программно, посредством очистки флага соответствующего прерывания. Например, если возникло переполнение таймера 0, в регистре TCON устанавливается флаг переполнения TF0, который должен вести к

возникновению прерывания. Однако, если система уже обслуживает какое-нибудь прерывание, прерывание по переполнению таймера 0 может быть отложено. В этом случае управляющая программа может очистить бит TF0, сняв таким образом прерывание.

С другой стороны, можно программно вызвать любое прерывание, если установить соответствующий флаг прерывания. Если при этом нет блокировки прерываний (система не обслуживает никакого прерывания), то сразу за установкой флага прерывания возникнет прерывание.

Прерывание не возникает в случаях, если выполняется команда RETI, или любая команда, обращающаяся к регистрам IE или IP.

Для обслуживания прерывания процессор формирует и выполняет команду LCALL с адресом вектора прерывания. При этом в стек заносится счетчик команд PC, а в счетчик команд заносится вектор прерывания.

Интерфейс I^2C

Интерфейс I^2C используется для связи микросхем друг с другом в сети и для управления устройствами, поддерживающими интерфейс I^2C. В первом случае микроконвертер выступает в качестве либо ведомого (*slave*), либо ведущего устройства (*master*). Это определяется конфигурацией сети. Во втором случае микроконвертер выступает в качестве ведущего по отношению к устройствам. Микроконвертер ADuC842 реализует интерфейс I^2C, поддерживая аппаратный режим ведомого и программный режим ведущего.

Интерфейс управляется регистром I2CCON, назначение разрядов которого в режиме ведущего (мастер) приведено в таблице 40.

Таблица 40. Регистр I2CCON в режиме ведущего

Имя	*Разряд*	*Назначение*
MDO	7	Если бит MDE=1, бит MDO поступает на вывод SDATA.
MDE	6	Если 1, вывод SDATA используется для вывода (TX). Если 0, вывод SDATA используется для ввода (RX).
MCO	5	Данные из этого бита поступают на вывод SCLOCK и формируют синхроимпульсы.
MDI	4	Если бит MDE=0, данные с вывода SDATA защелкиваются в этот бит.
I2CM	3	Если 1, интерфейс работает в режиме ведущего, иначе в режиме ведомого.
—	2, 1, 0	Не используется.

Кроме регистра управления, интерфейсом I^2C используется также регистр данных I2CDAT и четыре регистра адреса периферийного устройства I2CADD, I2CADD1, I2CADD2 и I2CADD3.

Работа микросхемы в режиме ведомого в настоящем пособии не рассматривается. В режиме ведущего сигналы на линиях интерфейса формируются при помощи битов MCO, MDE, MDI и MDO регистра I2CCON.

Описание интерфейса I²C

Интерфейс I²C разработан фирмой *Philips Semiconductors*. В этом интерфейсе для связи отдельных устройств используется две линии (шины), называемые SDA (*serial data*, линия данных) и SCL (*serial clock*, линия синхронизации). Линии SDA в микроконвертере соответствует вывод SDATA, а линии SCL — вывод SCLOCK. Все устройства подключаются к этим двум линиям параллельно (рисунок 43).

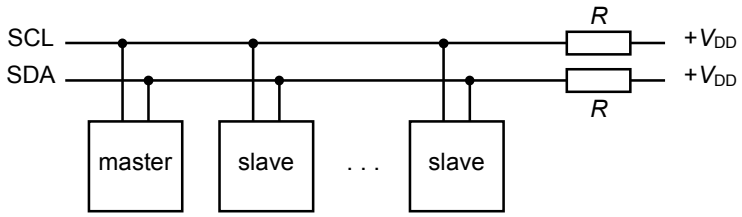


Рисунок 43. Подключение устройств в интерфейсе I²C

В обычном состоянии на линиях интерфейса поддерживается высокий уровень за счет резисторов *R*, которые «подтягивают» уровень на линии к напряжению питания V_{DD}. Отдельные устройства могут поочередно или одновременно изменять состояние любой из линий на низкое. При этом на линиях формируются сигналы (импульсы). Такая схема соединения называется «монтажное И» (*wire AND*).

Если два или более устройств одновременно понижают уровень линии, возникает проблема распознавания того устройства, которое это сделало. Поэтому обычно работа устройств согласуется во времени так, чтобы устройства не понижали уровни линий одновременно (в случае, когда в сети работают два и более мастер-устройств, они могут использовать одновременное понижение линий для их захвата).

Согласование одновременной работы устройств достигается за счет линии синхронизации, которой обычно управляет только одно устройство, а также за счет протокола обмена информацией, который все подключенные устройства должны соблюдать. Протокол определяет, в какие моменты времени какие устройства и какую информацию передают по линии данных.

При обмене информацией одно из устройств является *передатчиком*, а другое *приемником*. Во время одного сеанса связи одно и то же устройство может как принимать, так и передавать информацию. Передатчиком является то устройство, которое передает *байты данных*, а приемником — устройство, которое принимает их. Кроме байт данных, по шинам I²C передается адресная и служебная информация.

Мастер-устройство, периодически понижая уровень линии SCL, создает на ней последовательность импульсов. Каждый импульс предназначен для передачи одного бита (импульс синхронизации *стробирует* импульс данных). Информация на линии данных считается достоверной только тогда, когда уровень линии синхронизации высокий (рисунок 44).

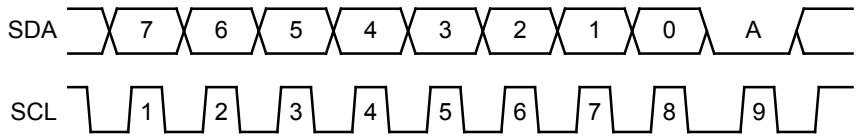


Рисунок 44. Передача одного байта информации по шине I²C

Информация на линии SDA должна изменяться только в промежутках между импульсами синхронизации, когда линия SCL имеет низкий уровень. Информация по шине I^2C передается бит за битом, начиная со старшего и заканчивая младшим. После передачи 8 бит передается еще один, девятый бит, называемый *битом подтверждения* (*acknowledge*). Этот бит является обязательным и его всегда выставляет то устройство, которое принимает информацию. Устройство подтверждает прием байта информации, если бит подтверждения имеет низкий уровень. На диаграммах бит подтверждения обозначают символом A.

Обмен информацией происходит в ходе *сеансов связи*. Для обозначения начала и окончания сеанса связи по линии данных передаются специальные сигналы, называемые *стартовым* и *стоповым*. Эти сигналы передает мастер-устройство (рисунок 45).

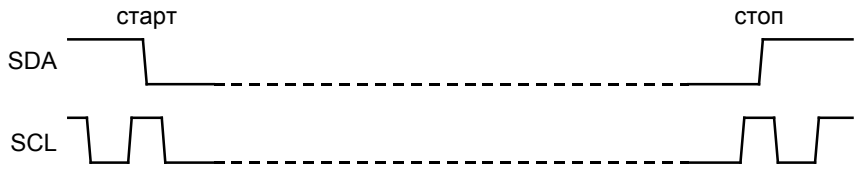


Рисунок 45. Стартовое и стоповое состояния шины I²C

Стартовый сигнал заключается в смене состояния линии данных с высокого на низкое в течение активности (высокого значения) сигнала синхронизации. Стоповый сигнал, наоборот, меняет состояние линии данных с низкого на высокое во время импульса синхронизации. Ведомые устройства постоянно отслеживают состояние линии данных для обнаружения состояния «старт» или «стоп». На диаграммах сеансов связи состояние «старт» обозначается символом S, а состояние «стоп» — символом P. Состояние «старт» может появляться также в течение сеанса для смены направления передачи. В этом случае его называют «*рестартом*» или «*повторным стартом*». На диаграммах это состояние иногда обозначается символом Sr (*start repeat*).

Первым передаваемым байтом после состояния «старт» всегда является *адрес ведомого устройства* (*slave address*). Адрес устройства состоит из 7 бит, поэтому по шине I^2C можно адресовать максимум 2^7=128 устройств (интерфейс предусматривает также 10-разрядную адресацию, но в микроконтроллере она не используется). Восьмым битом первого байта передается *режим*. Режим «*запись*» (нулевой бит) означает, что ведущее устройство в данном сеансе будет передатчиком, а режим «*чтение*» (единичный бит) означает, что ведомое устройство будет приемником.

Существует несколько типичных сеансов связи. На рисунке 46 приведена типичная диаграмма сеанса записи в устройство.

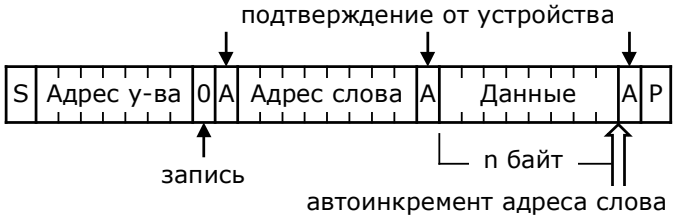


Рисунок 46. Запись в устройство

В сеансе записи по шине передается *n*+2 байт, *n* — количество байт данных. После состояния «старт» мастер передает адрес ведомого устройства и режим «запись» (нулевой восьмой бит). После получения подтверждения от устройства мастер передает адрес первого байта данных, называемого «адресом слова». Далее мастер передает *n* байт данных.

Ведомое устройство, получив адрес слова, запоминает его во внутреннем *указателе данных*, и использует для того, чтобы поместить байт данных в ячейку своей памяти. После приема байта данных ведомое устройство инкрементирует указатель данных, так, что каждый последующий байт размещается в последовательных ячейках памяти устройства, начиная от адреса первого слова. После получения каждого байта устройство выставляет бит подтверждения, понижая уровень линии данных.

Чтобы завершить передачу байт данных, мастер-устройство переводит шину в состояние «стоп».

Типичная диаграмма сеанса чтения из устройства приведена на рисунке 47.

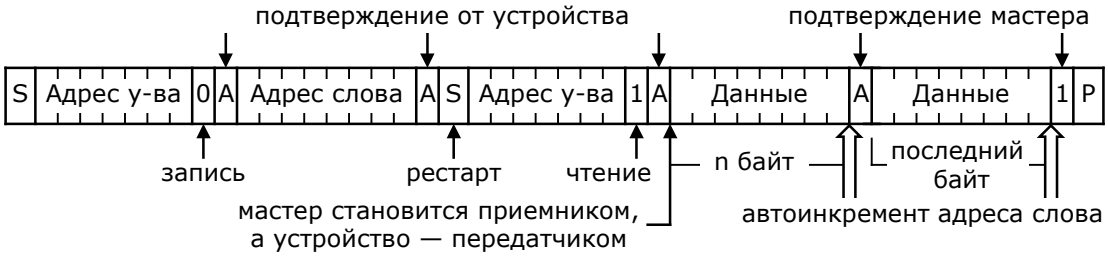


Рисунок 47. Чтение устройства после установки адреса слова

В этом сеансе первые два байта передаются так же, как и в предыдущем. Сначала передается адрес устройства и режим «запись», затем адрес первого слова данных. Поскольку адрес слова передается только во время записи в устройство, а требуется его чтение, далее мастер переводит шину в состояние «рестарт» (обозначенное вторым символом S).

После состояния «рестарт» на шине начинается новый цикл, в котором снова передается адрес устройства и режим «чтение» (единица в восьмом бите), после чего устройство передает, а мастер принимает *n* байт данных.

После приема каждого байта данных мастер выставляет бит подтверждения, однако после приема последнего байта мастер передает не нулевой бит подтверждения, а единичный бит. Это необходимо для того, чтобы ведомое устройство освободило линию данных для передачи состояния «стоп».

Для чтения устройства можно использовать сеанс, показанный на рисунке 48.

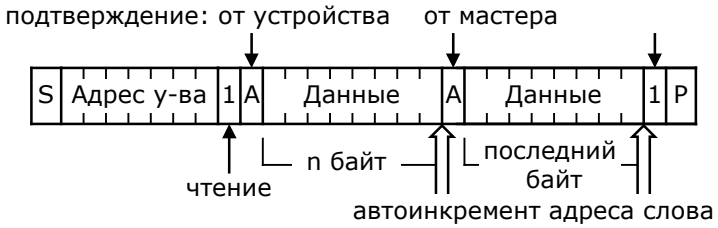


Рисунок 48. Чтение устройства после передачи режима чтения

В этом сеансе мастер передает режим «чтение» и сразу начинает прием байтов данных. Поскольку адрес слова в этом сеансе не передается, ведомое устройство использует текущее значение внутреннего указателя данных. Этот сеанс связи следует использовать только после того, как указатель данных ведомого устройства был установлен в предыдущих сеансах. Иногда для этой цели используется инициализирующий сеанс, показанный на рисунке 49.

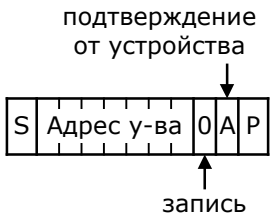


Рисунок 49. Инициализация ведомого устройства

В сеансе инициализации не передаются байты информации. Мастер передает только адрес устройства. Если устройство выставляет бит подтверждения, мастер может сделать вывод, что ведомое устройство присутствует. Одновременно, обнаружив свой адрес, ведомое устройство инициализирует указатель данных в нулевое значение. Заметим, что инициализация может происходить только при первом обращении к устройству. При последующих обращениях указатель данных не изменяется. Кроме того, не все устройства производят инициализацию при первом обращении. Некоторые устройства требуют передачи байта данных с определенным адресом, который является инициализатором. В силу сказанного, предпочтительнее использовать сеансы, в которых адрес слова указан явно.

В случае, если ведомое устройство не подтвердило прием байта, мастер принимает решение о завершении сеанса, формируя состояние «стоп».

Возможно формирование и других сеансов связи. Следует помнить, что после состояния «старт» (или «рестарт») обязательно должен следовать адрес устройства.

При программировании обмена по шине I^2C следует учитывать ширину импульсов и промежутков между ними. Основные временные характеристики приведены на рисунке 50 и в таблице 41.

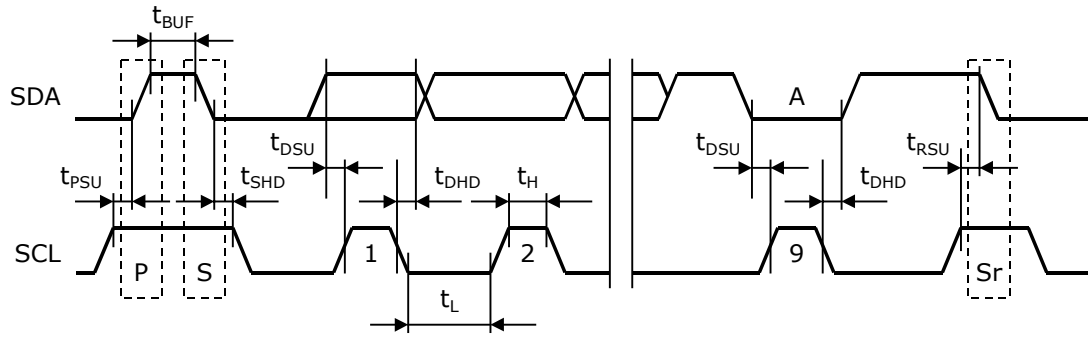


Рисунок 50. Временные характеристики интерфейса I^2C

Таблица 41. Временные характеристики интерфейса I^2C

Параметр	*Описание*	*Min*	*Max*	*Ед. изм.*
t_L	Время между имульсами синхронизации	1,3		мкс
t_H	Ширина импульса синхронизации	0,6		мкс
t_{SHD}	Задержка при старте	0,6		мкс
t_{DSU}	Установка данных	0,1		мкс
t_{DHD}	Задержка данных		0,9	мкс
t_{RSU}	Установка при стопе	0,6		мкс
t_{PSU}	Установка при рестарте	0,6		мкс
t_{BUF}	Время перехода стоп—старт	1,3		мкс

В заключение отметим, что приведенное здесь описание интерфейса I^2C не является полным. Для более детального знакомства с этим интерфейсом рекомендуется изучение соответствующей документации.

Система команд

Все команды условно можно разделить на:

- команды пересылки данных;
- арифметические команды;
- логические команды;
- команды для операций с битами (булевы операции);
- команды управления потоком вычислений (передачи управления).

Команды используют 4 метода адресации:

- непосредственный — операнд записан в команде;
- прямой — адрес операнда записан в команде;
- косвенный — адрес операнда записан в регистре;
- неявный — адрес операнда подразумевается смыслом команды.

Всего имеется 13 типов команд, различающихся количеством операндов и методами адресации (таблицы 42 и 43). Первый байт команды всегда является кодом операции (КОП). Второй и третий байты определяют операнды или адреса, записанные в команде.

Таблица 42. Типы команд

Тип	*Байт 1*	*Байт 2*	*Байт 3*
1	КОП	—	—
2	КОП	#data	—
3	КОП	direct	—
4	КОП	bit	—
5	КОП	rel	—
6	КОП	addr11	—
7	КОП	direct	#data
8	КОП	direct	rel
9	КОП	directs	directd
10	КОП	#data	rel
11	КОП	bit	rel
12	КОП	addr16 (high)	addr16 (low)
13	КОП	#data16 (high)	#data16 (low)

Таблица 43. Обозначения операнда и (или) адреса в команде

Метод	*Описание*
#data	8-битный непосредственный операнд.
direct	8-битный внутренней адрес памяти или регистра SFR.
bit	8-битный адрес бита.
rel	8-битное смещение со знаком.
addr11	11-битный абсолютный адрес (в пределах 2 Кбайт).
addr16	16-битный абсолютный адрес.
#data16	16-битный непосредственный операнд.
directs	8-битный внутренний адрес источника.
directd	8-битный внутренний адрес приемника.

Основным элементом данных является байт. Он используется как непосредственный операнд (константа, содержащаяся в самой команде) и как элемент данных при обращении к портам и другим регистрам SFR, внешней памяти данных и памяти программ. 16-разрядные операнды используются для загрузки указателя данных DPTR и в качестве адреса в инструкциях передачи управления.

Команды пересылки данных

Команды пересылки данных выполняют копирование операндов из одних ячеек памяти в другие или обмен операндов. Мнемоническое обозначение команды пересылки — MOV (move). Часть команд имеют другие обозначения. Пересылки, в которых участвует память программ, обозначаются MOVC (move from code), а пересылки, в которых участвует внешняя память данных, обозначаются MOVX (move to/from XRAM). Команды,

выполняющие обмен операндов, обозначаются XCH и XCHD (*exchange*). В эту же группу входят команды помещения в стек PUSH и извлечения из стека POP.

Список команд пересылки приведен в таблице 44. Буквой «Б» обозначено количество байт в команде, буквой «Ц» — количество машинных циклов (для микроконвертера ADuC842). В последнем столбце буквами «C A O» обозначены флаги CY, AC и OV. Звездочка в столбце обозначает, что флаг может быть изменен командой, минус — флаг не может быть изменен, 0 — флаг сбрасывается, 1 — флаг устанавливается.

Таблица 44. Команды пересылки

Мнемокод	***Операция***	***КОП***	***Б***	***Ц***	***C A O***
MOV A,Rn	A⇐Rn	E8–EF	1	1	– – –
MOV A, @Ri	A⇐(Ri)	E6, E7	1	2	– – –
MOV A, direct	A⇐direct	E5	2	2	– – –
MOV A, #data	A⇐#data	74	2	2	– – –
MOV Rn, A	Rn⇐A	F8–FF	1	1	– – –
MOV Rn, direct	Rn⇐direct	A8–AF	2	2	– – –
MOV Rn, #data	Rn⇐#data	78–7F	2	2	– – –
MOV direct, A	direct⇐A	F5	2	2	– – –
MOV direct, Rn	direct⇐Rn	88–8F	2	2	– – –
MOV direct, @Ri	direct⇐(Ri)	86, 87	2	2	– – –
MOV directd, directs	directd⇐directs	85	3	3	– – –
MOV direct, #data	direct⇐#data	75	3	3	– – –
MOV @Ri, A	(Ri)⇐A	F6, F7	1	2	– – –
MOV @Ri, direct	(Ri)⇐direct	A6, A7	2	2	– – –
MOV @Ri, #data	(Ri)⇐#data	76, 77	2	2	– – –
MOV DPTR, #data16	DPTR⇐#data16	90	3	3	– – –
Пересылки из памяти программ					
MOVC A, @A+DPTR	A⇐PM(A+DPTR)	93	1	4	– – –
MOVC A, @A+PC	A⇐PM(A+PC)	83	1	4	– – –
Пересылки из внешней памяти данных					
MOVX A, @Ri	A⇐XRAM(Ri)	E2, E3	1	4	– – –
MOVX A, @DPTR	A⇐XRAM(DPTR)	E0	1	4	– – –
MOVX @Ri, A	XRAM(Ri)⇐A	F2, F3	1	4	– – –
MOVX @DPTR, A	XRAM(DPTR)⇐A	F0	1	4	– – –
Другие пересылки					
PUSH direct	SP⇐SP+1, (SP)⇐direct	C0	2	2	– – –
POP direct	direct⇐(SP), SP⇐SP–1	D0	2	2	– – –
XCH A, Rn	A⇔Rn	C8–CF	1	1	– – –
XCH A, @Ri	A⇔(Ri)	C6, C7	1	2	– – –
XCH A, direct	A⇔direct	C5	2	2	– – –
XCHD A, @Ri	A[3:0]⇔(Ri)[3:0]	D6, D7	1	2	– – –

Символами PM обозначена память программ, а XRAM — внешняя память данных. Запись типа A[3:0] обозначает часть ячейки памяти или регистра, а именно, биты 0—3 включительно.

Запись Rn обозначает один из регистров R0—R7. Запись типа A⇐Rn означает, что источник — регистр Rn, а получатель — аккумулятор.

В следующем примере регистр R7 копируется в аккумулятор:

```
MOV A,R7
```

Запись Ri обозначает один из регистров R0 или R1. Запись типа A⇐(Ri) означает, что в регистре R0 или R1 находится адрес операнда. В примере в аккумулятор записывается содержимое ячейки 30h:

```
MOV R0,#30h
MOV A,@R0
```

Аккумулятор имеет два разных имени. Имя A используется при неявной адресации. В примере в аккумулятор записывается непосредственное число:

```
MOV A,#30h
```

Имя ACC является прямым адресом аккумулятора (ACC=0E0h) и используется в любом месте, в котором метод адресации *direct*. В следующем примере аккумулятор сохраняется в стеке:

```
PUSHACC
```

Эту команду нельзя записать в виде

```
PUSHA
```

так как не существует метода адресации «аккумулятор» при операциях со стеком. Эту же команду можно записать с явным адресом аккумулятора, однако такая запись менее понятна:

```
PUSH0E0h
```

Явный адрес регистра вместо его обозначения Rn используется в случае, когда необходимо выполнить пересылку из одного регистра в другой. В следующем примере регистр R1 копируется в регистр R0:

```
MOV R0,01h
```

Нельзя записать команду в виде

```
MOV R0,R1h
```

так как не существует метода адресации «из регистра в регистр».

Знак @ («эт») в командах обозначает косвенную адресацию. При этом регистр, перед которым стоит знак @, содержит адрес операнда. В следующем примере ячейка памяти программ копируется в аккумулятор. Адрес ячейки памяти 1000h предварительно записывается в указатель данных DPTR, а аккумулятор очищается, так как он складывается с указателем данных для вычисления адреса операнда:

```
MOV DPTR,#1000h
CLR A
MOVCA,@A+DPTR
```

Команда типа MOVC предназначена для чтения операнда из памяти программ. Записать в память программ таким образом нельзя. Для чтения и записи во внешнюю память данных используется команда типа MOVX и либо указатель данных DPTR, либо один из регистров R0 или R1. При использовании R0 и R1 в ВПД можно адресовать только первые 256 ячеек.

Арифметические команды

К арифметическим командам относятся: сложение ADD, сложение с переносом ADDC, вычитание с заёмом SUBB, умножение MUL, деление DIV, увеличение на единицу INC, уменьшение на единицу DEC и десятичная коррекция DA. Арифметические команды приведены в таблице 45.

Таблица 45. Арифметические команды

Мнемокод	***Операция***	***КОП***	***Б***	***Ц***	***С А О***
ADD A,Rn	A⇐A+Rn	28–2F	1	1	* * *
ADD A, @Ri	A⇐A+(Ri)	26, 27	1	2	* * *
ADD A, direct	A⇐A+direct	25	2	2	* * *
ADD A, #data	A⇐A+#data	24	2	2	* * *
ADDC A, Rn	A⇐A+Rn+CY	38–3F	1	1	* * *
ADDC A, @Ri	A⇐A+(Ri)+CY	36, 37	1	2	* * *
ADDC A, direct	A⇐A+direct+CY	35	2	2	* * *
ADDC A, #data	A⇐A+#data+CY	34	2	2	* * *
SUBB A,Rn	A⇐A–Rn–CY	96–9F	1	1	* * *
SUBB A, @Ri	A⇐A–(Ri)–CY	96, 97	1	2	* * *
SUBB A, direct	A⇐A–direct–CY	95	2	2	* * *
SUBB A, #data	A⇐A–#data–CY	94	2	2	* * *
INC A	A⇐A+1	04	1	1	– – –
INC Rn	Rn⇐Rn+1	08–0F	1	1	– – –
INC @Ri	(Ri)⇐(Ri)+1	06, 07	1	2	– – –
INC direct	direct⇐direct+1	05	2	2	– – –
INC DPTR	DPTR⇐DPTR+1	A3	1	3	– – –
DEC A	A⇐A–1	18–1F	1	1	– – –
DEC Rn	Rn⇐Rn–1	16, 17	1	1	– – –
DEC @Ri	(Ri)⇐(Ri)–1	14	1	2	– – –
DEC direct	direct⇐direct–1	15	2	2	– – –
MUL AB	B, A⇐A×B	A4	1	9	0 – *
DIV AB	A, B⇐A/B	84	1	9	0 – *
DA A	если A[3:0]>9 или AC=1 то A[3:0]⇐A[3:0]+6 если A[7:4]>9 или CY=1 то A[7:4]⇐A[7:4]+6	D4	1	2	* – –

Команды сложения и вычитания устанавливают флаг CY при переносе из разряда 7, флаг AC — при переносе из разряда 3, флаг OV — при пе

реносе из разряда 6 и отсутствии переноса из разряда 7. Флаг OV устанавливается также командой MUL, если результат больше 255, и командой DIV при делении на 0. При умножении регистр A содержит младшую часть результата, регистр B — старшую. При делении регистр A содержит частное, регистр B — остаток от деления.

Логические команды

К логическим командам относятся: логическое И ANL, логическое ИЛИ ORL, исключающее ИЛИ XRL, очистка CLR, дополнение (инверсия) CPL, циклические сдвиги аккумулятора вправо RR (рисунок 51, а) и влево RL (рисунок 51, б), циклические сдвиги аккумулятора через перенос вправо RRC (рисунок 52, а) и влево RLC (рисунок 52, б) и обмен тетрад аккумулятора SWAP.

Логические команды приведены в таблице 46.

Таблица 46. Логические команды

Мнемокод	***Операция***	***КОП***	***Б***	***Ц***	***C A O***
ANL A,Rn	A⇐A AND Rn	58–5F	1	1	– – –
ANL A, @Ri	A⇐A AND (Ri)	56, 57	1	2	– – –
ANL A, direct	A⇐A AND direct	55	2	2	– – –
ANL A, #data	A⇐A AND #data	54	2	2	– – –
ANL direct, A	direct⇐direct AND A	52	2	2	– – –
ANL direct, #data	direct⇐direct AND #data	53	3	3	– – –
ORL A,Rn	A⇐A OR Rn	48–4F	1	1	– – –
ORL A, @Ri	A⇐A OR (Ri)	46, 47	1	2	– – –
ORL A, direct	A⇐A OR direct	45	2	2	– – –
ORL A, #data	A⇐A OR #data	44	2	2	– – –
ORL direct, A	direct⇐direct OR A	42	2	2	– – –
ORL direct, #data	direct⇐direct OR #data	43	3	3	– – –
XRL A,Rn	A⇐A XOR Rn	68–6F	1	1	– – –
XRL A, @Ri	A⇐A XOR (Ri)	66, 67	1	2	– – –
XRL A, direct	A⇐A XOR direct	65	2	2	– – –
XRL A, #data	A⇐A XOR #data	64	2	2	– – –
XRL direct, A	direct⇐direct XOR A	62	2	2	– – –
XRL direct, #data	direct⇐direct XOR #data	63	3	3	– – –
CLR A	A⇐0	E4	1	1	– – –
CPL A	A⇐NOT A	F4	1	1	– – –
RL A	A[n+1]⇐A[n], A[0]⇐A[7]*	23	1	1	– – –
RR A	A[n]⇐A[n+1], A[7]⇐A[0]*	03	1	1	* – –
RLC A	A[n+1]⇐A[n], A[0]⇐CY, CY⇐A[7]*	33	1	1	* – –
RRC A	A[n]⇐A[n+1], A[7]⇐CY, CY⇐A[0]*	13	1	1	– – –
SWAP A	A[3:0]⇔A[7:4]	C4	1	1	– – –

*) n=0÷6

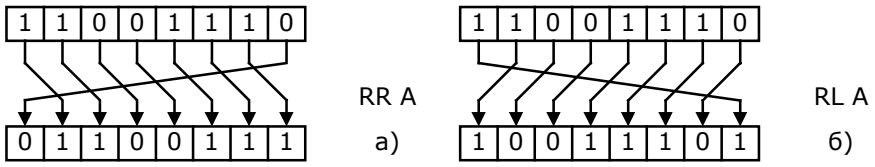


Рисунок 51. Циклические сдвиги аккумулятора вправо (а) и влево (б)

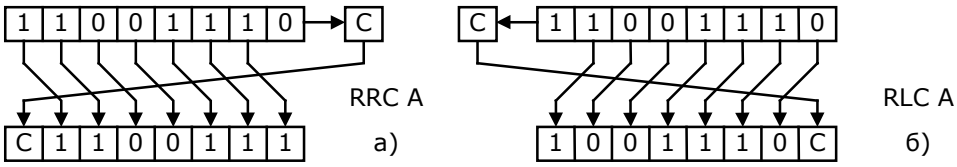


Рисунок 52. Циклические сдвиги аккумулятора через перенос вправо (а) и влево (б)

Команды операций с битами

Над битами могут быть выполнены операции очистки CLR, установки SETB, инвертирования CPL, логического И ANL, логического ИЛИ ORL, пересылки бита в битовый аккумулятор CY или из битового аккумулятора (команда MOV).

В качестве операндов логических команд используются 128 бит битового поля, разряды бит-адресуемых регистров и флаг переноса CY.

Команды операций с битами приведены в таблице 47.

Таблица 47. Команды операций с битами

Мнемокод	*Операция*	*КОП*	*Б*	*Ц*	*С А О*
CLR C	CY⇐0	C3	1	1	0 – –
CLR bit	bit⇐0	C2	2	2	– – –
SETB C	CY⇐1	D3	1	1	1 – –
SETB bit	bit⇐1	D2	2	2	– – –
CPL C	CY⇐NOT CY	B3	1	1	* – –
CPL bit	bit⇐NOT bit	C2	2	2	– – –
ANL C, bit	CY⇐CY AND bit	82	2	2	* – –
ANL C, /bit	CY⇐CY AND (NOT bit)	B0	2	2	* – –
ORL C, bit	CY⇐CY OR bit	72	2	2	* – –
ORL C, /bit	CY⇐CY OR (NOT bit)	A0	2	2	* – –
MOV C, bit	CY⇐bit	A2	2	2	* – –
MOV bit, C	bit⇐CY	92	2	2	– – –

Команды передачи управления

К командам передачи управления относятся: безусловный длинный переход LJMP, абсолютный страничный переход (в пределах 2Кбайт) AJMP, короткий относительный переход SJMP, косвенный относительный переход JMP @A+DPTR, условные переходы по значению аккумулятора JZ и JNZ, условные переходы по значению флага переноса JC и JNC, условные переходы по значению бита JB, JNB и JBC, переходы по неравен

ству аккумулятора и ячейки памяти или регистра CJNE, переходы по результату декремента регистра DJNZ, длинный вызов подпрограммы LCALL, абсолютный вызов подпрограммы (в пределах 2Кбайт) ACALL, возврат из подпрограммы RET, возврат из прерывания RETI и пустая операция NOP. Сводка команд управления приведена в таблице 48. В таблице используется значение счетчика команд после выборки команды.

Таблица 48. Команды управления

Мнемокод	***Операция***	***КОП***	***Б***	***Ц***	***С А О***
LCALL addr16	SP⇐SP+1, (SP)⇐PC[7:0], SP⇐SP+1, (SP)⇐PC[15:8] PC⇐addr16	12	3	4	– – –
LJMP addr16	PC⇐addr16	02	3	4	– – –
SJMP rel	PC⇐PC+rel	80	2	3	– – –
JMP @A+DPTR	PC⇐A+DPTR	73	1	3	– – –
RET	PC[15:8]⇐(SP), SP⇐SP–1, PC[7:0]⇐(SP), SP⇐SP–1	22	1	4	– – –
RETI	то же и разблокировка	32	1	4	– – –
JC rel	PC⇐PC+rel если CY=1	40	2	3	– – –
JNC rel	PC⇐PC+rel если CY=0	50	2	3	– – –
JB bit, rel	PC⇐PC+rel если bit=1	20	3	4	– – –
JNB bit, rel	PC⇐PC+rel если bit=0	30	3	4	– – –
JBC bit, rel	если bit=1, PC⇐PC+rel, bit⇐0	10	3	4	* * *
JZ rel	PC⇐PC+rel если A=0	60	2	3	– – –
JNZ rel	PC⇐PC+rel если A≠0	70	2	3	– – –
CJNE A, direct, rel	PC⇐PC+rel если A≠direct	B5	3	4	* – –
CJNE A, #data, rel	PC⇐PC+rel если A≠#data	B4	3	4	* – –
CJNE Rn, #data, rel	PC⇐PC+rel если Rn≠#data	B8–BF	3	4	* – –
CJNE @Ri, #data, rel	PC⇐PC+rel если (Ri)≠#data	B6, B7	3	4	* – –
DJNZ Rn, rel	Rn⇐Rn-1, PC⇐PC+rel если Rn≠0	D8–DF	2	3	– – –
DJNZ direct, rel	direct⇐direct–1, если direct≠0 то PC⇐PC+rel	D5	3	4	– – –
NOP	холостой машинный цикл	00	1	1	– – –
ACALL addr11	SP⇐SP+1, (SP)⇐PC[7:0], SP⇐SP+1, (SP)⇐PC[15:8] PC⇐PC+addr11	11, 31, 51, 71, 91, B1, D1, F1	2	3	– – –
AJMP addr11	PC⇐PC+addr11	01, 21, 41, 61, 81, A1, C1, E1	2	3	– – –

Команда JBC изменяет бит, указанный в качестве операнда.

Команды условного перехода по неравенству операндов CJNE устанавливают флаг CY, если левый операнд меньше правого (x<y):

```
CJNE    x,y,METKA.
```

3. Основы программирования ядра 8051/8052

В этом разделе приводятся только основные сведения по программированию на ассемблере фирмы *MetaLink Corporation*, необходимые для выполнения практических работ. За более подробным описанием программирования на ассемблере следует обратиться к соответствующим полным описаниям, которые входят в комплект учебного стенда.

Программа на ассемблере состоит из программных строк. Каждая строка, если она не пустая, является командой, директивой кросс-ассемблера, или строкой комментария. Некоторые строки могут также содержать метки.

Команды при компиляции программы преобразуются в машинные инструкции (исполняемый код). Следовательно, не имеет смысла программа, которая не содержит ни одной команды.

Директивы кросс-ассемблера управляют процессом компиляции. Так, например, директива END указывает компилятору на конец программного текста, а директива IF — на выборочную компиляцию. Директивы управляют также распределением памяти. Так, директива DATA связывает байт памяти данных с идентификатором, а директива DSEG указывает на переход к памяти данных.

Метки — это идентификаторы, после которых следует двоеточие. Они предназначены для указания точек (адресов) программы, в которые возможен переход при помощи команд, передающих управление.

Комментарии начинаются со знака «;», могут начинаться в любом месте строки и продолжаются до конца текущей строки.

Идентификаторы

Идентификатор — это символическое имя, описывающее адрес памяти данных или памяти программ. Идентификаторы могут содержать буквы (латинский алфавит), цифры, знак вопроса и знак подчеркивания. Первым символом идентификатора не может быть цифра (с цифр начинаются константы). Регистр букв не имеет значения (кросс-ассемблер все буквы переводит в верхний регистр). Длина идентификатора не может превышать 255 символов, причем только первые 32 знака имеют значение.

В качестве идентификаторов не могут быть использованы имена, используемые самим ассемблером, такие, как мнемоники команд, директивы кросс-ассемблера, операции, выполняемые во время компиляции (EQ, NE, GE, GT, LE, LT, HIGH, LOW, MOD, SHR, SHL, NOT, AND, OR, XOR).

Примеры допустимых идентификаторов:

```
START
PutByteToI2C
current_position
_Index
?_operand
```

Примеры недопустимых идентификаторов:

1stOperand — начинается с цифры
Var#1 — содержит недопустимый знак #
MOV — является мнемоникой команды
DATA — является директивой кросс-ассемблера
LOW — является оператором

Описание констант и переменных

Для описания константы используется директива EQU. С ее помощью можно связать идентификатор с числовым значением и использовать далее идентификатор вместо непосредственного значения. Обычно описания констант располагают в начале программы отдельным блоком.

Примеры описаний констант:

DEC_VAL EQU 65 — описывает константу DEC_VAL со значением 65
HEX_VAL EQU 41h — описывает константу HEX_VAL со значением 65
BIN_VAL EQU 1000001b — описывает константу BIN_VAL со значением 65
A_SYMBOL EQU 'A' — описывает константу A_SYMBOL со значением 65

Описание константы можно использовать также в качестве имени переменной, если задать значением константы адрес переменной, например:

VAR_PO EQU 30h — описывает адрес 30h

Адрес переменной можно также задать директивой DATA:

VAR_PO DATA 30h — описывает байтовый адрес 30h

Числовые значения в описаниях констант и используемые в качестве операндов команд, могут быть заданы в десятичной, двоичной, восьмеричной и шестнадцатеричной системах счисления. Признаком десятичной записи числа является отсутствие суффикса. Признаком записи числа в другой системе счисления является наличие суффикса: B (Binary) для двоичной, O (Octal) или Q для восьмеричной и H (Hexadecimal) для шестнадцатеричной.

Адрес битовой переменной можно задать как при помощи директивы EQU, так и при помощи директивы BIT:

SOMEBIT0 EQU 0h — описывает битовый адрес 0h
SOMEBIT1 BIT 1h — описывает битовый адрес 1h

Текущая позиция в коде

Текущая позиция в коде программы (текущий адрес в программе) обозначается знаком доллара $. Использовать текущую позицию можно для перехода на текущую строку программы, а также для вычисления смещения от текущей точки программы до другой её точки.

Текущая позиция используется для перехода в инструкциях, описывающих условный или безусловный переход. Например, следующая команда безусловного перехода описывает «мертвый» цикл бесконечного перехода на саму команду:

```
SJMP    $
```

Этот фрагмент кода полностью аналогичен следующему фрагменту:

```
DEAD: SJMP   DEAD
```

Аналогичный переход может быть использован при программировании цикла при помощи инструкции DJNZ:

```
DJNZ    R0,$
```

Текущую позицию удобно использовать в цикле ожидания установки или очистки какого-либо бита. В следующем примере текущая позиция используется для организации цикла ожидания переполнения таймера 0 (бит переполнения таймера 0 называется TF0):

```
JNB     TF0,$
```

Программа зацикливается на этой инструкции до тех пор, пока бит TF0 не будет установлен в единичное состояние (если программа содержит логическую ошибку — таймер 0 не включен — программа зациклится на этой инструкции навсегда). Дождаться очистки бита TF0 можно при помощи аналогичной команды:

```
JB      TF0,$
```

Текущая позиция используется для вычисления смещения от текущей точки в программе до другой точки при помощи оператора «минус»:

```
MESSAGE: DB 'This is a message.'
MESSAGE_LEN EQU $-MESSAGE
```

В приведенном примере кода символ MESSAGE_LEN получает значение длины строки данных, объявленной при помощи директивы DB. Если значение длины желательно записать как элемент данных непосредственно в код программы, вместо директивы EQU следует использовать директиву DB:

```
MESSAGE: DB 'This is a message.'
MESSAGE_LEN: DB $-MESSAGE
```

При этом длина строки будет записана в байт, непосредственно следующий за строкой.

Формат команды ассемблера

Машинная инструкция состоит из байта кода операции и, возможно, из одного или двух дополнительных байтов, описывающих операнды и адреса (см. таблицу 42 «Типы команд»).

Для записи машинных инструкций на ассемблере используются:

- мнемоники операций, такие, как MOV, ADD, INC, POP и другие;
- константы непосредственных значений, например «#0FFh»;
- константы адресов, например «0FFh»;
- метки, например «START:»;
- специальные обозначения A (аккумулятор), AB (операция над аккумулятором A и дополнением аккумулятора B), C (флаг CY, битовый аккумулятор), Rn (прямое обращение к регистру), @Ri (косвенное обращение к памяти данных), @DPTR (косвенное обращение к ВПД), @A+DPTR и @A+PC (косвенное обращение к памяти программ).

В целом общий формат команды на ассемблере имеет следующий вид:

```
[МЕТКА:] МНЕМОНИКА [ОПЕРАНД[,ОПЕРАНД[,ОПЕРАНД]]]
```

Здесь квадратные скобки обозначают необязательность присутствия данного элемента команды. Таким образом, простейшая команда состоит из одной мнемоники, более сложные команды содержат дополнительные слова (обозначенные словом «ОПЕРАНД»), разделенные запятой.

Любая команда может начинаться с метки, указывающей на адрес команды. Этот адрес может быть использован для передачи управления на данную команду при помощи команд, передающих управление. При программировании часто бывает удобнее располагать метки на отдельных строчках (перед командами).

Примерами однобайтных команд, не имеющих операндов, являются команды NOP (холостой машинный цикл), RET (возврат из подпрограммы) и RETI (возврат из прерывания).

Наличие дополнительных слов «ОПЕРАНД» не обязательно ведет к появлению дополнительных байт команды. Любая команда, операндом или операндами которой являются специальные обозначения, является однобайтной.

Команда

```
MOV     A,R0
```

пересылает значение регистра R0 в аккумулятор A. Машинная инструкция этой команды состоит только из одного байта — кода операции, равного C8.

Команды пересылки

Команды пересылки данных являются одними из наиболее часто употребляемыми. Можно сказать, что пересылка — это основное действие, из которого состоит программа.

Зачем вообще нужны пересылки? Во-первых, с их помощью можно задать значение регистра SFR. Например, чтобы настроить таймер 0 на работу в режиме 1, в регистр TCON нужно записать значение 1, а для этого используется команда пересылки непосредственного операнда 1

```
MOV     TCON,#1
```

Во-вторых, команда пересылки используется для сохранения значений в памяти для последующего его использования. Например, если микроконтроллер управляет некоторым устройством, и контролируемые сигналы поступают на вход порта 0, то имеет смысл сохранить содержимое порта в памяти данных, например, в ячейке, обозначенной идентификатором VALUE:

```
    MOV     VALUE,P0
```

С помощью команды пересылки можно также получить копию значения из одной ячейки памяти в другую для того, чтобы, например, выполнить преобразование этого значения, при этом сохранить старое значение. Следующая команда копирует значение из ячейки памяти, обозначенной идентификатором VALUE, в аккумулятор A для последующего анализа отдельных битов:

```
    MOV     A,VALUE
```

Команда пересылки имеет формат

```
[МЕТКА:] MOV ПРИЕМНИК,ИСТОЧНИК
```

Здесь операнд ПРИЕМНИК указывает на ячейку памяти или регистр SFR, принимающий информацию, которая содержится в ячейке памяти или регистре SFR, указываемых операндом ИСТОЧНИК.

Отметим пересылки, выполняемые косвенно. Следующий пример показывает, как можно очистить всю память данных при помощи косвенной адресации ячеек памяти:

```
    MOV     R0, 0FFh
    CLR     A
C1: MOV     @R0, A
    DJNZ    R0, C1
```

Здесь ячейка памяти программ адресуется косвенно (знак @ указывает на косвенное обращение) через регистр R0, в котором находится постоянно меняющийся адрес. Начальное значение адреса равно 0FFh, конечное — 0.

Пересылки из памяти программ

Массивы числовых или символьных значений (сообщения) могут располагаться в памяти программ. В этом случае они объявляются как элементы данных при помощи, например, директивы DB. В качестве примера рассмотрим размещение в памяти программ некоторого сообщения, обозначенного меткой MESSAGE:

```
    SJMP    AFTER_MESSAGE
MESSAGE: DB 'This is a message.'
AFTER_MESSAGE:
```

Часть кода программы, содержащая не машинные инструкции, а данные, должна быть расположена таким образом, чтобы предотвратить ис

пользование элементов данных в качестве машинных инструкций. В приведенном выше примере элементы данных «обходятся» при помощи инструкции безусловного перехода SJMP. Более удобным может оказаться расположение элементов данных после таких машинных инструкций, которые безусловно передают управление в другую точку программы — после инструкций SJMP, LJMP, RET. Например, если в программе есть подпрограмма, то имеет смысл данное сообщение разместить в программе после одной из подпрограмм:

```
    ; здесь располагается код подпрограммы
    RET
MESSAGE: DB 'This is a message.'
MESSAGE_LEN EQU $-MESSAGE
SOME_LABEL:
    ; здесь может располагаться код программы или подпрограммы
```

Для того, чтобы во время исполнения программы извлечь символы из указанной строки данных, в некотором месте программы организуется цикл, в каждой итерации которого используется инструкция MOVC пересылки из памяти программ в аккумулятор. Например:

```
    MOV     R0,#MESSAGE_LEN
DLOOP:
    MOV     DPTR,#MESSAGE
    MOV     A,R1
    MOVC    A,@A+DPTR
    ; какие-то действия с элементом данных
    INC     R1
    DJNZ    R0,DLOOP
```

В приведенном фрагменте кода регистр R0 используется в качестве счетчика итераций. Начальное значение этого регистра принимается равным длине строки. Регистр R1 используется как индекс элемента строки данных. Инкремент этого регистра командой INC обеспечивает продвижение по строке.

Заметим, что при помощи инструкции MOVC данные можно только извлечь из памяти программ, но нельзя записать, так как память программ является ПЗУ. Запись в нее во время исполнения программы возможна, но при помощи других методов.

Пересылки в ВПД и обратно

Внешняя память данных в учебном стенде SDK1.1 используется для размещения 512 Кбайт данных в восьми страницах по 64 Кбайт, а также для адресации устройств стенда (см. раздел «Программирование устройств SDK1.1»).

Для пересылки данных в ВПД или обратно используются инструкции MOVX и регистр указателя данных DPTR. В отличие от архитектуры ядра

8051 в микроконвертере ADuC842 указатель данных является 24-х разрядным (в ядре 8051 указатель данных 16-ти разрядный).

Чтобы записать байт данных в ВПД или в регистр устройства, нужно сначала записать три байта адреса данных или устройства в регистры DPTR, затем пересылаемое значение записать в аккумулятор, а затем использовать инструкцию MOVX. В следующем примере непосредственное значение 255 записывается в ячейку ВПД, имеющую абсолютный 24-х разрядный адрес 000000h:

```
MOV     DPP,#0
MOV     DPH,#0
MOV     DPL,#0
MOV     A,#255
MOVX    @DPTR,A
```

Чтобы прочитать содержимое ячейки ВПД, нужно установить ее адрес в регистры DPTR, затем использовать инструкцию MOVX. Получить значение из ВПД можно только в аккумулятор. В примере в аккумулятор записывается значение из ячейки с адресом 20, находящейся в странице 1 ВПД:

```
MOV     DPP,#1
MOV     DPH,#0
MOV     DPL,#20
MOVX    A,@DPTR
```

Для выполнения пересылок блоков данных в ВПД или обратно можно использовать особенность микроконвертера ADuC842 — двойной указатель данных (см. раздел «Двойной указатель данных»).

Если в программе для учебного стенда адресное пространство ВПД используется исключительно для адресации внешних по отношению к микроконвертеру устройств, таких, как клавиатура, дисплей, календарь и других, имеет смысл в начале программы один раз установить значения регистров DPP и DPH, так как для всех регистров устройств они имеют одинаковое значение:

```
MOV     DPP,#8
MOV     DPH,#0
```

Операции с битами

Операции с битами для любой управляющей системы являются очень важными, поскольку как контролируемые, так и управляющие сигналы во многих случаях представляют собой двоичные (битовые) сигналы.

Архитектура ядер 8051/8052 и, соответственно, микроконвертера ADuC842 имеет все необходимые средства для простой работы с битами. Эти средства включают в себя битовое поле, бит-адресуемые регистры SFR, и команды для работы с отдельными битами.

Операции с битами очень просты:

- установка бита (запись в бит значения 1);
- очистка бита (запись в бит значения 0);
- пересылка бита из бит-адресуемой ячейки памяти в битовый аккумулятор C (флаг переноса CY) и обратно;
- логические команды AND и OR;
- инверсию бита CPL;
- команды проверки бита.

Все команды, работающие с битами, предполагают использование битового адреса (обозначенного в таблице 42 «Типы команд» словом *bit*). Битовый адрес ничем не отличается от байтового адреса и может принимать такие же значения — от 0h до 0FFh. Различие проявляется только в используемой команде. Если команда предполагает операцию с битом, то адрес является адресом бита, а если команда предполагает работу с байтом, то адрес в команде — это адрес байта.

Установить бит можно командой SETB, а очистить — командой CLR. Ниже приведено несколько примеров установки и очистки битов, использующих разные приемы задания битового адреса:

```
SETB TF0        — устанавливает поименованный бит регистра SFR
SETB ACC.0      — устанавливает бит 0 в бит-адресуемом аккумуляторе
SETB 0          — устанавливает бит, указанный прямым адресом
CLR C           — очищает битовый аккумулятор
CLR SOMEBIT     — очищает бит с пользовательским именем
```

Пересылки битов возможны только между битовым аккумулятором CY и любым другим битом, например:

```
MOV C,SOMEBIT — поименованный бит пересылается в CY
MOV SOMEBIT,C — бит CY пересылается в поименованный бит
```

Логические операции с битами выполняются над битовым аккумулятором CY и произвольным битом или его инверсией. При помощи следующей команды можно убедиться, что бит CY и бит SOMEBIT одновременно равны единице:

```
ANL     C,SOMEBIT
```

Результат записывается в бит CY.

Если бит CY равен единице, то оба бита до выполнения команды были равны единице. При помощи следующей команды можно убедиться, что бит CY равен единице, а бит SOMEBIT равен нулю:

```
ANL     C,/SOMEBIT
```

Если после выполнения операции бит CY равен единице, то до выполнения операции бит SOMEBIT был равен нулю. Перед выполнением двух последних операций бит CY должен быть установлен в 1.

При помощи операции OR можно убедиться в том, что один из битов команды равен единице:

```
ORL     C,SOMEBIT
```

Если после выполнения этой команды бит CY равен единице, то до выполнения команды один из битов или оба бита CY или SOMEBIT были равны единице.

Команды проверки битов используются для организации алгоритма программы (для ветвления). Следующая команда ожидает установки бита переполнения TF0 таймера 0:

```
JNB     TF0,$
```

Программа зацикливается на этой команде до тех пор, пока бит переполнения не будет установлен. Следующая команда, наоборот, ждет, когда данный бит будет очищен:

```
JB      TF0,$
```

Вместо знака доллара $ в этих командах может стоять также адрес перехода на любое место программы, отстоящее от команды не более чем на 127—128 байт.

Еще одна команда проверяет произвольный бит, и если он установлен, то очищает его и выполняет переход по указанному в команде адресу:

```
JBC     SOMEBIT,SOME_ADDRESS
```

Логические команды

При помощи логических команд можно выполнять разнообразные полезные действия. Операция AND позволяет замаскировать в операнде один или несколько разрядов. Так, если для вас важно проверить младшую тетраду операнда, то старшую тетраду можно замаскировать при помощи маски 0Fh:

```
ANL     A,0Fh
```

При этом следует помнить, что замаскированные разряды очищают в источнике те разряды, в которых в маске находятся нули (иначе говоря, начальное значение операнда будет утеряно во время операции). Еще один пример — прочитав значение года и даты из календаря (см. «Программирование устройств SDK1.1»), вам наверняка потребуется замаскировать, например, биты, относящиеся к году, чтобы прочитать значение, указывающее на дату.

Предположим, что прочитанное из календаря значение находится в аккумуляторе. Чтобы оставить в аккумуляторе только значение даты, нужно замаскировать старшие 2 разряда. Составляем маску, в которой два старших разряда равны нулю, а остальные разряды равны единице, и используем команду ANL:

```
ANL     A,#00111111b
```

Чтобы наоборот, прочитать значение года, нужно замаскировать младшие шесть разрядов. На самом деле проще будет «прокрутить» биты в аккумуляторе так, чтобы два старших разряда стали младшими. Для этого аккумулятор нужно сдвигать сам в себя два раза влево при помощи команды RL, после чего замаскировать старшие шесть разрядов:

```
RL      A
RL      A
ANL     A,#00000011b
```

Возвращаясь к дате, которая закодирована в виде двух десятичных цифр (см. таблицу 62), нужно также выделить каждую цифру даты в отдельное число и записать его, например, в память по некоторым адресам. Можно поступить следующим образом. Предположим, что считанное из календаря значение записано в ячейке, обозначенной символов CAL. Тогда пересылаем значение из ячейки CAL в аккумулятор, маскируем старшие 4 разряда, выделяя в аккумуляторе цифру единиц даты, после чего записываем полученную цифру в ячейку, обозначенную DATE1:

```
MOV     A,CAL
ANL     A,0Fh
MOV     DATE1,A
```

Далее снова берем исходное значение из ячейки CAL, меняем местами младшую и старшую тетрады, маскируем старшие шесть битов и результат записываем в ячейку, обозначенную DATE2:

```
MOV     A,CAL
SWAP    A
ANL     A,03h
MOV     DATE2,A
```

При помощи операции AND можно также удостовериться в наличии установленных битов в произвольном операнде. Так, при чтении регистра клавиатуры вам нужно узнать, если ли вообще какое-нибудь нажатие клавиши в проверяемом столбце, то есть убедиться, что в старших четырех разрядах есть хотя бы один ноль. Это можно проверить, сначала инвертировав аккумулятор, а затем замаскировав младшие 4 разряда:

```
CPL     A
ANL     A,#0F0h
```

После выполнения этих команд аккумулятор равен нулю, если до этого ни один разряд из старших четырех не содержал нуля, и не равен нулю, если хотя бы один разряд содержал ноль.

При помощи операции OR можно установить произвольные биты в произвольном байте. Например, при программировании дисплея вам понадобится установить позицию курсора. Это действие выполняется командой, в которой позиция курсора задается младшими семью битами, а старший бит равен единице. Предполагая, что текущая позиция находится в

ячейке памяти, обозначенной CURSOR_POS, следующие две команды сформируют в аккумуляторе необходимый операнд:

```
MOV     A,CURSOR_POS
ORL     A,#80h
```

Команда ORL устанавливает те биты в аккумуляторе, которые в маске заданы единицами.

Операция XOR (команда XRL) может быть использована для инверсии тех битов в источнике, которые в маске заданы единицами. Например, если аккумулятор A содержит значение 00001111b, а маска равна 00111100b, результатом операции будет значение 00110011b. Операцию XOR можно также использовать для проверки равенства двух значений. Если два значения равны, операция XOR между ними дает нулевой результат. В следующем примере сравниваются значение в аккумуляторе A и регистре R0. Результат проверяется при помощи команды проверки нулевого значения аккумулятора:

```
XRL     A,R0
JZ      EQUAL
```

Команды сдвигов аккумулятора используются не только для перемещения битов в подходящую для анализа позицию, но и для побитового анализа аккумулятора. При этом используются команды сдвига через флаг CY. Например, в следующем цикле аккумулятор сдвигается через бит CY вправо, позволяя проанализировать в каждой итерации цикла один из битов, начиная с младшего:

```
    MOV     R0,#8
LOOP:
    RRC     A
    ; анализ бита C, выдвинутого из аккумулятора
    DJNZ    R0,LOOP
```

Другие примеры использования команд циклического вращения аккумулятора через бит CY можно найти в разделе «Календарь» главы «Программирование устройств SDK1.1».

Арифметические команды

Арифметические команды используются для вычислений арифметических выражений. Они представлены командами сложения, вычитания, умножения, деления, десятичной коррекции, инкремента и декремента.

Самые простые команды инкремента INC и декремента DEC увеличивают или уменьшают значение операнда на единицу. Особого пояснения они не требуют.

Команда сложения выполняется над аккумулятором и операндом, результат записывается в аккумулятор. Есть две разновидности этой коман

ды — простое сложение операндов ADD, и сложение операндов с переносом ADDC.

Простое сложение формирует результат сложения следующим образом. Если двоичная сумма операндов не превышает 256, то сумма является результатом. Если сумма равна или выше 256, то результат меньше суммы на 256.

Например, если значение в аккумуляторе равно 127, то результатом следующей команды будет значение 254 (0FEh):

```
ADD     A,#127
```

При тех же условиях результатом следующей команды будет значение, равное нулю:

```
ADD     A,#129
```

Результат сложения оценивается при помощи флагов CY и OV. Флаг CY устанавливается в единичное значение, если при сложении произошел перенос из разряда 7. Установка этого флага означает, что результат сложения превысил значение 256.

Флаг OV устанавливается в единичное значение, если произошел перенос из разряда 6, но не из разряда 7. Установка флага OV означает, что результат сложения находится в диапазоне 128—255.

Интерпретация флагов CY и OV зависит от того, какими вы считаете складываемые числа — знаковыми или беззнаковыми. Если складываемые числа считаются беззнаковыми, установка флага CY указывает на неверный результат. Если складываемые числа считаются знаковыми, то установка любого флага, CY или OV, указывает на неверный результат.

Например, если складываются два беззнаковых числа 127 и 129, устанавливается флаг CY, что означает, что результат, равный 0, неверен. Если же складываются два знаковых числа 127 и 1, то будет установлен флаг OV, указывающий что сумма превысила допустимый диапазон положительного числа (результат сложения 128 является отрицательным числом 128). Если складываются два отрицательных числа 128 (равное —128) и 255 (равное —1), при сложении будет установлен флаг CY, указывающий на неверный результат 127.

Сложение с переносом ADDC выполняется аналогично простому сложению, но к результату прибавляется значение бита CY. Эта операция нужна для организации алгоритма сложения двух чисел, записываемых в виде двух и более байт. При этом сначала складываются младшие байты чисел при помощи простого сложения, а затем старшие байты чисел при помощи сложения с переносом. Например, если одно число находятся в регистрах R2 (младшая часть) и R3 (старшая часть), а второе — в регистрах R4 (младшая часть) и R5 (старшая часть), то получить их сумму в регистры R6 (младшая часть) и R7 (старшая часть) можно при помощи следующих команд:

```
    MOV     A,R2
    ADD     A,R4
    MOV     R6,A
    MOV     A,R3
    ADDC    A,R5
    MOV     R7,A
```

Команда вычитания с заёмом SUBB вычитает из аккумулятора указанный операнд и значение бита переноса CY. Команды простого вычитания нет, поэтому, чтобы выполнить простое вычитание, предварительно нужно очистить бит переноса CY. Результат вычитания оценивается по значениям флагов CY и OV так же, как и при сложении, и зависит от интерпретации чисел.

К системе команд нет команды сравнения двух чисел, поэтому при необходимости сравнение заменяется вычитанием. При этом необходимо помнить, что при вычитании теряется одно из чисел, и его необходимо запомнить. В качестве примера использования команды вычитания для сравнения двух чисел приведем подпрограмму TOHEXLA, которая преобразует младшую тетраду аккумулятора в соответствующую шестнадцатеричную цифру:

```
TOHEXLA:
    ANL     A,0Fh
    CLR     C
    SUBB    A,#10
    JC      TOHEX1
    ADD     A,#7
TOHEX1:
    ADD     A,#58
    RET
```

Здесь из тетрады вычитается число 10. При этом если тетрада имела значение меньше десяти, то при вычитании устанавливается флаг переноса CY, и подпрограмма переходит к метке TOHEX1, и к полученному при вычитании числу прибавляется 58 (в котором 48 — это код цифры 0, а 10 — число, которое было вычтено из тетрады). Если же тетрада имела значение большее или равное 10, то флаг CY не устанавливается, и тогда к результату вычитания дополнительно прибавляется число 7. Например, если тетрада была равна 10, то после сложения нуля с 7 и с 58 будет получено число 65, равное коду буквы A.

Команда умножения MUL AB умножает значение в аккумуляторе A на значение в дополнении аккумулятора B (регистр B является регистром SFR, так же, как и аккумулятор). Старший байт результата умножения записывается в регистр B, а младшая часть — в аккумулятор A. Если при умножении установлен флаг OV, результат операции превысил 255.

Команда деления DIV AB делит значение в аккумулятора A на значение в дополнении аккумулятора B. Частное записывается в аккумулятор A, остаток от деления — в регистр B. Если при делении установлен флаг OV, произошло деление на ноль.

Команда десятичной коррекции DA A используется для коррекции результата при сложении или вычитании двух двоично-кодированных десятичных цифр. Двоично-кодированная десятичная цифра записывается своим значением в одной из тетрад байта. В одном байте таким образом можно записать две десятичные цифры.

Например, если для записи десятичной цифры используется только младшая тетрада, то значение байта совпадает со значением цифры. Предположим, что аккумулятор содержит двоично-кодированную десятичную цифру 9 (то есть значение 9), к которой мы хотим прибавить двоично-кодированную цифру 6 (то есть значение 6). Результатом сложения 9 и 6 является число 15, которое умещается в тетраду, однако при сложении десятичных цифр тетрада должна содержать число 5 и при этом должен возникать перенос в старшую цифру. Десятичная коррекция решает эту проблему:

```
MOV     A,#9
ADD     A,#6
DA      A
```

После выполнения этого кода аккумулятор содержит число 15h, которое соответствует двоично-кодированному числу 15. Если же команду десятичной коррекции опустить, результатом сложения будет число 0Fh, что не соответствует никакому двоично-кодированному десятичному числу.

В следующем примере кода складываются два числа 19 и 16 из двух двоично-кодированных десятичных цифр. При этом цифры 1 записаны в старших тетрадах, а цифры 9 и 6 — в младших. Результатом сложения после десятичной коррекции является правильное двоично-кодированное десятичное число 35 (после выполнения операции аккумулятор содержит число 35h):

```
MOV     A,#19h
ADD     A,#16h
DA      A
```

После выполнения команды сложения устанавливается также бит дополнительного переноса AC, указывающий на перенос из разряда 3, который возникает, например, при сложении двоично-кодированных десятичных чисел 9 и 9 (результат сложения не умещается в тетраду).

Команда десятичной коррекции устанавливает флаг переноса CY, если двоично-кодированный результат аккумулятора превышает значение 99. Например, при сложении двух двоично-кодированных десятичных чисел 90 и 16 результат десятичной коррекции дает в аккумуляторе значение 6 и перенос:

```
MOV     A,#90h
ADD     A,#16h
DA      A
```

Флаг дополнительного переноса AC следует проверять тогда, когда байтовое значение используется для представления одной двоично-кодированной десятичной цифры. Флаг переноса CY следует проверять в случае, когда байтовое значение используется для представления двух двоично-кодированных десятичных цифр.

Команды управления потоком

При программировании на ассемблере в распоряжении программиста нет операторов управления потоком вычислений, таких, как условный оператор или оператор цикла. Вместо этого в программе на ассемблере используются условные и безусловные передачи управления.

Команда безусловной передачи управления заставляет процессор перейти из текущей точки программы в произвольное другое место. Есть две команды безусловного перехода — короткий переход SJMP и длинный переход LJMP. Обе команды в качестве операнда используют метку, которая указывает на точку перехода.

Команда короткого перехода SJMP позволяет выполнить переход из текущей точки на расстояние не больше, чем знаковое значение байта, то есть вперед по коду на максимальное расстояние 127 байт и назад по коду на максимальное расстояние 128 байт. При попытке выполнить более далекий переход компилятор фиксирует ошибку.

Команда длинного перехода LJMP позволяет выполнить переход в любое место программы. Существует также команда перехода AJMP в пределах 2 Кбайт, однако использовать её не рекомендуется.

Команды условной передачи управления выполняют переход при наличии определенного состояния.

Самая простая команда условного перехода выполняет переход, проверяя значение в аккумуляторе. Если значение в аккумуляторе равно нулю, команда JZ выполняет переход, а если значение в аккумуляторе не равно нулю, то команда JNZ выполняет переход.

При программировании конструкций, управляющих потоками вычислений, следует пользоваться понятием ветки. Ветка — это последовательность ассемблерных команд, которая исполняется или не исполняется в зависимости от некоторых условий. При этом ветки в памяти программ расположены одна за другой. Чтобы обеспечить выполнение одной из двух или более ветвей, нужно использовать операторы условного или безусловного перехода, позволяющих обходить ненужные в конкретном случае ветки.

Рассмотрим, как при помощи команд условного и безусловного переходов программируется аналог условного оператора. Предположим, нужно запрограммировать алгоритм, включающий в себя условный оператор:

```
ЕСЛИ A=0 ТО
    R0=1
ИНАЧЕ
    R0=2
ВСЕ
```

В алгоритме есть две ветки, которые соответствуют разным состояниям аккумулятора. Если значение в аккумуляторе равно нулю, то выполняется ветка «ИСТИНА»

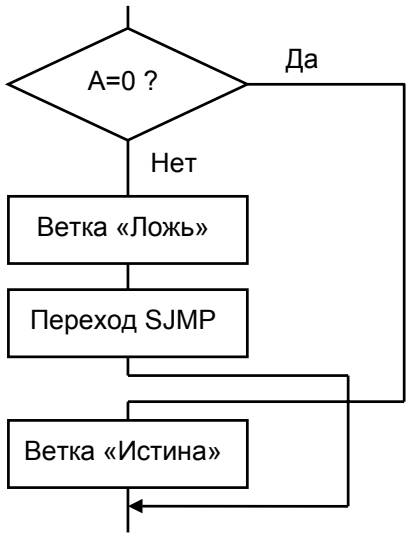
```
    R0=1
```

а если значение в аккумуляторе не равно нулю, то выполняется ветка «ЛОЖЬ»

```
    R0=2
```

Запрограммировать данный алгоритм можно двумя способами, в зависимости от того, какую ветку расположить в памяти первой. Если для проверки условия использовать команду JZ, то первой в памяти программ будет расположена ветка «ЛОЖЬ», а за ней — ветка «ИСТИНА»:

```
    MOV     A,#0
    JZ      TRUE_BRANCH
    MOV     R0,#2
    SJMP    END_IF
TRUE_BRANCH:
    MOV     R0,#1
END_IF:
```



На рисунке справа показано размещение в памяти веток данного алгоритма, а также линии, показывающие обход веток. Например, при помощи команды SJMP после выполнения ветки «ЛОЖЬ» обходится ветка «ИСТИНА».

Пусть теперь требуется запрограммировать более простой алгоритм с одной только веткой «ИСТИНА»:

```
ЕСЛИ A=0 ТО
    R0=1
ВСЕ
```

В этом случае выбор команды условного перехода JZ или JNZ не произволен, использовать можно только команду JNZ:

```
    MOV     A,#0
    JNZ     END_IF
    MOV     R0,#1
END_IF:
```

В качестве условия, вызывающего ветвление программы, может быть использован также флаг переноса CY. Есть две команды перехода по состоянию этого бита — команда JC вызывает переход, если флаг CY установлен (имеет значение 1), команда JNC вызывает переход, если флаг CY не установлен (имеет значение 0). Флаг переноса устанавливается, например, при выполнении арифметических операций, что позволяет скорректировать результат вычислений, выполняя ту или иную ветку алгоритма.

Часто бит переноса используется для индикации успешности или неуспешности выполнения какой-либо подпрограммы. При этом программист явно устанавливает в подпрограмме этот флаг, если результат ее выполнения следует считать ошибочным, и очищает флаг, если результат выполнения подпрограммы следует считать нормальным. После команды вызова подпрограммы в этом случае следует команда проверки флага CY, и ветвление алгоритма на соответствующие ветки, например:

```
    LCALL   SOME_SUBROUTINE
    JC      ERROR_HANDLER
    ; нормальное завершение подпрограммы
    SJMP    SOME_POINT
ERROR_HANDLER:
    ; обработка ошибки выполнения подпрограммы
SOME_POINT:
```

Другими условиями, с помощью которых можно организовать ветвление, являются команды проверки состояния произвольного бита JB, JNB и JBC. Поскольку их применение описывалось ранее, здесь они не рассматриваются. См., например, разделы «Текущая позиция в коде» и «Битовые операции».

Еще одна команда позволяет сравнить два операнда, и выполнить переход, если операнды не совпадают. Это команда CJNE (*compare and jump if not equal* — сравнить и перейти, если не равны). В качестве операндов в ней могут быть использованы аккумулятор, регистры и произвольные ячейки памяти.

В программах часто используются циклические конструкции (циклы). Цикл имеет только одну ветку, называемую *итерацией*. Итерация может быть исполнена произвольное число раз, в том числе ни разу.

В зависимости от расположения конструкции, которая проверяет условие, различают циклы с предусловием и циклы с постусловием.

В цикле с предусловием проверка условия производится перед началом выполнения итераций, поэтому эти циклы используют тогда, когда итерации могут не исполняться ни разу. В языках высокого уровня этим циклам соответствуют циклы while (рисунок 53).

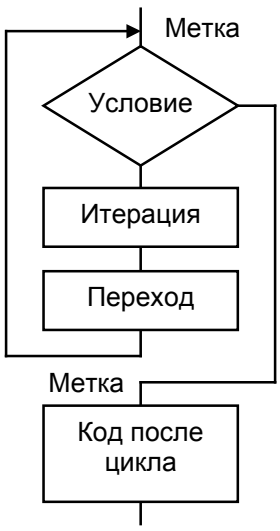


Рисунок 53. Организация цикла с предусловием

В цикле с постусловием проверка условия производится после выполнения итерации, поэтому в этих циклах итерация исполняется минимум один раз (рисунок 54).

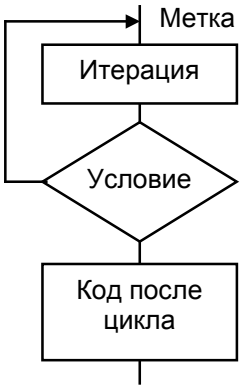


Рисунок 54. Организация цикла с постусловием

На рисунках 53 и 54 не указано, в каком случае при проверке условия производится переход, так как это зависит от применяемой для проверки условия команды, и решается в каждом конкретном случае индивидуально.

В качестве примера приведем организацию цикла с предусловием, использующим команду сравнения операндов CJNE:

```
LOOP_START:
    CJNE    A,R0,LOOP_EXIT
    ; команды итерации
    SJMP    LOOP_START
LOOP_EXIT:
    ; код после цикла
```

Здесь в качестве условия завершения цикла используется неравенство содержимого аккумулятора A и регистра R0. Если эти регистры содержат одинаковое значение, выполняется итерация.

Следующий пример показывает организацию цикла с постусловием:

```
LOOP_START:
    ; команды итерации
    CJNE    A,R0,LOOP_START
    ; код после цикла
```

Здесь равенство содержимого аккумулятора A и регистра R0 является условием продолжения цикла, а неравенство — условием завершения.

Для организации цикла со счетчиком итераций в ядре 8051 предусмотрена команда DJNZ (*decrement and jump if not zero* — декремент и переход, если не равно нулю). В качестве счетчика в этой команде может быть использован регистр R0—R7 или прямо адресуемая ячейка памяти (в том числе и регистр SFR).

С помощью команды DJNZ можно организовать как цикл с постусловием, так и цикл с предусловием. Пример организации цикла с постусловием:

```
    MOV     R0,#NUMBER_OF_ITERATIONS
LOOP_START:
    ; команды итерации
    DJNZ    R0,LOOP_START
```

Здесь условием продолжения цикла является неравенство нулю содержимого регистра R0, который выполняет роль обратного счетчика итераций. Перед выполнением цикла в этот регистр необходимо занести число, равное требуемому количеству итераций.

Подпрограммы

Подпрограмма — часть общего кода программы, выполнение которой требуется неоднократно. Подпрограммы похожи на процедуры и функции в языках программирования высокого уровня (на самом деле процедуры и функции в языках высокого уровня реализуются на ассемблере с помощью подпрограмм). Подпрограммы используются также для обеспечения читабельности кода основной программы. В этом случае они выполняют какую-то конкретную задачу, при этом не обязательно многократно.

Каждая подпрограмма обозначается своим уникальным именем, в качестве которого используется метка той части программы, с которой начинается выполнение подпрограммы.

Выполнение подпрограммы вызывается при помощи команды вызова LCALL, при этом в стек заносится адрес команды, следующей непосредственно за командой вызова, а подпрограмма по завершении своего алгоритма использует команду возврата RET, которая извлекает из стека адрес, записанный при вызове, в результате чего управление возвращается к команде, следующей за командой вызова. Таким образом, очень важно соблюдать равенство вызовов LCALL и возвратов RET.

Подпрограммы являются обособленными частями программы, поэтому, если они используют общие для всей программы регистры Rn, которые

в основной программе хранят какие-то значения, то подпрограммы должны сохранять значения используемых ими регистров в стеке перед началом выполнения своего алгоритма при помощи команд PUSH, а перед завершением восстанавливать значения регистров при помощи команд POP.

Учитывая вышеизложенное, общий вид подпрограммы следующий:

```
МЕТКА_ПОДПРОГРАММЫ:
    ; сохранение используемых регистров в стеке
    ; код подпрограммы
    ; восстановление использованных регистров из стека
    RET
```

Для правильной работы подпрограмм также важно, чтобы количество команд PUSH в начале подпрограммы совпадало с количеством команд POP, которые были выполнены до команды RET. Поскольку подпрограммы могут содержать произвольное количество команд RET, перед каждой из них требуется выполнить команды восстановления регистров, например:

```
МЕТКА_ПОДПРОГРАММЫ:
    PUSH    ACC
    ; какой-то код
    JZ      МЕТКА1
    POP     ACC
    RET
МЕТКА1:
    ; какой-то код
    POP     ACC
    RET
```

Ту же самую логику подпрограммы можно организовать при помощи одной команды возврата RET:

```
МЕТКА_ПОДПРОГРАММЫ:
    PUSH    ACC
    ; какой-то код
    JZ      МЕТКА1
    SJMP    МЕТКА2
МЕТКА1:
    ; какой-то код
МЕТКА2:
    POP     ACC
    RET
```

Одни подпрограммы могут вызывать другие подпрограммы, организуя вложенные вызовы. Поэтому следует помнить о стеке.

Для программы, в которой много подпрограмм, вызывающих другие подпрограммы, необходимо установить в начале работы программы (обычно в самом начале) указатель стека SP в значение 7Fh.

Если подпрограмме нужны аргументы, то для их передачи обычно используют аккумулятор A или регистры Rn. Примеры использования аккумулятора в качестве аргумента подпрограммы можно, например, найти в разделе «Программирование устройств стенда SDK1.1». Главное, о чем нужно не забыть при разработке подпрограмм — какие регистры Rn или другие ячейки памяти используются и не вызовет ли их использование порчу данных в вызывающем участке кода.

Часто совершаемые ошибки при написании подпрограмм:

1. Переход на метку подпрограммы при помощи инструкции условного или безусловного перехода. Поскольку подпрограммы завершаются инструкцией возврата, извлекающей адрес следующей команды из стека, такой переход однозначно ведет к неправильной работе программы в целом.

2. Переход из одной подпрограммы в другую при помощи команды условного или безусловного перехода. Хотя такие переходы в принципе разрешены, не следует этого делать, чтобы избежать запутанности программы. Кроме того, при этом, возможно, будет нарушено равенство количеств вызовов и возвратов, что ведет к неправильной работе программы.

При разработке программы возникает вопрос — в каком месте программы должны быть размещены подпрограммы? Ответ — в любом, в котором вам удобно. Наиболее часто подпрограммы размещают в конце программы или в ее начале. Пример организации области подпрограмм в конце программы:

```
    ; команды программы
    SJMP    $   ; мертвый цикл
    ; область подпрограмм
```

Пример организации программы с подпрограммами, расположенными в ее начале:

```
    LJMP    START
    ; область подпрограмм
START:
    ; команды программы
    SJMP    $   ; мертвый цикл
```

Есть простое правило, определяющее место, в котором может быть размещена подпрограмма — после любой команды безусловного перехода, а также после команд RET и RETI, например:

```
    SJMP    SOME_POINT
    ; удачное место для размещения какой-нибудь подпрограммы
```

Программирование задержек

Программирование задержек при организации алгоритмов управляющих программ для микроконтроллеров встречается часто. Это связано с

необходимостью, например, выдерживать временные интервалы при передаче информации по протоколам тех или иных интерфейсов.

Есть два принципиально различных подхода к программированию задержек. Первый заключается в выполнении бесполезных с точки зрения алгоритма машинных инструкций, с целью потратить на их исполнение заданное время. Это наиболее простой способ получить задержку, который применяется довольно широко, особенно в простых системах.

В системах, которые отвечают требованиям систем реального времени, задержки формируют с использованием таймеров и прерываний так, чтобы время задержки отсчитывалось во время выполнения какой-нибудь полезной работы.

Здесь мы рассмотрим формирование задержки с помощью бесполезного исполнения каких-нибудь инструкций.

Для выполнения задержки обычно пишется подпрограмма. Подпрограмм задержки может быть несколько, если для работы программы требуются разные временные задержки. Одни подпрограммы задержки могут вызывать другие подпрограммы задержки.

При разработке подпрограммы задержки нужно выбрать способ формирования потока бесполезно выполняемых инструкций. Важным фактором здесь является время задержки.

Если требуется задержка, исчисляемая микросекундами, то самый простой способ организовать её — вставить бесполезно исполняемые инструкции непосредственно в код программы. В качестве бесполезно выполняемой инструкции можно использовать команду NOP, которая для нормального режима работы учебного стенда выполняется за время 0,477 мкс, то есть задерживает исполнение следующей команды на это время. Можно вставить несколько команд NOP, чтобы получить время задержки в несколько микросекунд. Поделите время задержки в мкс на 0,477 мкс и получите количество команд NOP.

На самом деле нужно действительно поделить время задержки в микросекундах на 0,477 мкс для того, чтобы получить требуемое количество N бесполезно исполняемых машинных циклов. Далее, если задержка организуется на основе цикла, то нужно поделить полученное число N на количество машинных циклов, требуемых для исполнения той команды, которая будет использоваться в цикле. В результате будет получено число итераций.

Пример. Разработаем подпрограмму для задержки на 40 мс (40000 мкс). Делим 40000 на 0,477 и получаем примерно 83857 машинных циклов. Если в цикле исполнять команду DJNZ Rn, которая исполняется за 3 машинных цикла (таблица 48), потребуется 83857/3=27952 раза выполнить эту команду. С помощью команды DJNZ можно организовать цикл, в котором команда будет исполнена не более, чем 256 раз. Два вложенных друг в друга цикла дадут 256×256=65536 итераций. Следовательно, нужно

организовать два вложенных друг в друга цикла. Теперь можно написать следующую подпрограмму:

```
D40MS:
    MOV     R6,#0
D40MS1:
    MOV     R7,#K
    DJNZ    R7,$
    DJNZ    R6,D40MS1
    RET
```

Осталось определить число K итераций внутреннего цикла. Построим формулу для вычисления общей задержки. Во внутренний цикл входят K раз команда DJNZ, плюс выполнение команды DJNZ внешнего цикла и команды MOV R7,#K (2 машинных цикла): K×3+3+2. Это количество машинных циклов исполняется 256 раз плюс выполнение команды MOV R6,#0 (2 машинных цикла), плюс выполнение команды LCALL (4 машинных цикла), плюс выполнение команды RET (4 машинных цикла). Получили формулу количества машинных циклов для данной подпрограммы:

M = 10+256×(K×3+5).

Выразив теперь K через M, получим:

K = (M—1380)/768.

Подставляя M=83857, получаем примерно 107 внутренних итераций.

Выполнить задержку можно не только при помощи циклов, но и при помощи таймера. 16-ти разрядный таймер подсчитывает максимум 65536 машинных циклов, то есть максимальное время задержки, которое можно получить при помощи одного 16-ти разрядного таймера, равно 65536×0,477≈31261 мкс. Получить полное время работы одного таймера можно при помощи следующей подпрограммы:

```
D32MS:
    MOV     TCON,#1
    MOV     TL0,#0
    MOV     TH0,#0
    SETB    TR0
    JNB     TF0,$
    CLR     TF0
    RET
```

Первая команда задает режим работы таймера. Две следующих команды очищают счетчик таймера. Команда SETB запускает таймер, а команда JNB ожидает, когда таймер выставит флаг переполнения TF0.

Если при помощи таймера необходимо организовать меньшую задержку, чем полное время его счета, то нужно рассчитать число, которое следует записать в регистр счетчика. В качестве примера рассчитаем число для задержки в 10 мс.

Делим 10 мс на время исполнения машинного цикла 0,477 мкс и получаем 20964 машинных цикла. Вычитаем это число из максимального числа счета таймера 65536: 65536—20964=44572. Именно это число нужно записать в регистр счетчика таймера. Поскольку регистр счетчика представлен двумя регистрами SFR, регистром младшей части TL0 и регистром старшей части TH0, то нужно перевести число 44572 в шестнадцатеричную систему счисления: AE1Ch. Теперь понятно, что в регистр TL0 нужно записать значение 1Ch, а в регистр TH0 — значение AEh.

Программирование прерываний

Прерывание — это некоторое событие в системе, вызывающее временную приостановку выполнения программы (в любом ее месте) и переход на точку программной памяти, которая зависит от вектора прерывания. Все эти точки расположены в начале памяти программ по адресам, начиная с адреса 3 и далее каждая следующая точка через 8 байт.

Прерывания связаны с векторами — адресами перехода. Векторы прерываний по разным источникам прерываний (по разным событиям) указаны на рисунке 42 «Схема системы прерываний».

На самом деле при возникновении сигнала прерывания формируется команда LCALL с адресом, равным вектору прерывания. Таким образом, прерывание очень похоже на обычный вызов подпрограммы. Различие заключается в том, что моменты прерываний непредсказуемы, при прерывании используется блокировка, которая выключается командой возврата из прерывания RETI.

Если в системе используются какие-то прерывания, то они должны быть запрограммированы. Это означает, что по адресам векторов тех прерываний, которые в системе могут возникнуть, должны быть расположены подпрограммы обработки прерываний.

Прерывания могут возникать при сочетании множества условий.

Во-первых, прерывания вообще должны быть разрешены установкой бита EA в регистре IE. Если этот бит не установлен (а по умолчанию он не установлен), никакие прерывания ни при каких условиях не возникают.

Во-вторых, должно быть разрешено прерывание конкретного типа, соответствующего конкретному устройству или сигналу и вектору прерывания. Например, могут быть разрешены прерывания от таймера T0 при помощи установки бита ET0 в регистре IE. При этом могут возникать прерывания по вектору 0Bh.

В третьих, источник прерывания должен выставить сигнал прерывания, который является запросом на обслуживание прерывания.

В четвертых, запрос на прерывание не должен быть заблокирован обработкой другого прерывания (см. раздел «Система прерываний»).

Если все условия соблюдены, то прерывание возникает, и управление переходит по адресу вектора прерываний. При этом, как было сказано, по этому адресу должна находиться подпрограмма обработки прерывания.

Например, если вы предполагаете использовать прерывания от таймера T0, то по адресу 0Bh вы должны расположить код подпрограммы обработки прерывания. Поскольку на этот код в указанном месте отводится всего 8 байт, то инструкции обработчика прерывания могут быть очень немногочисленны, например:

```
    LJMP    START
    ORG     0Bh
    PUSH    PSW       ; команды
    LCALL   HANDLER   ; обработчика
    POP     PSW       ; прерывания
    RETI              ; вектора 0Bh
    ORG     80h
START:
    ; инструкции программы
    SJMP    $    ; мертвый цикл
HANDLER:
    ; инструкции обработчика прерывания
    RET
```

Здесь приведен примерный код программы, в которой используется прерывание от таймера T0. Директива ORG 0Bh указывает на начало обработчика прерывания, который сохраняет флаги, вызывает подпрограмму HANDLER, которая, собственно, и является обработчиком прерывания, после чего восстанавливает флаги и возвращается из прерывания при помощи команды RETI. Указанные инструкции занимают ровно 8 байт.

Другой вариант кода в этой части памяти программ представляет собой простейшее действие, например, подсчет числа прерываний:

```
    ORG     0Bh
    DEC     COUNT
    RETI
```

Основная программа при этом, находясь в бесконечном цикле, проверяет счетчик, а при достижении конца счета восстанавливает его начальное значение и выполняет требуемые действия.

Наконец, еще один способ реализации прерываний — поставить заглушку, чтобы прерывание было разблокировано:

```
    ORG     0Bh
    RETI
```

Естественно, в этом случае прерывание никак не обрабатывается (но и не приводит к неприятным последствиям, таким, как отказ программы).

Обработчик прерывания HANDLER обязан сохранить в стеке все используемые им регистры Rn в начале при помощи команд PUSH и восста

новить их в конце при помощи команд POP. Заметим, что порядок восстановления регистров должен быть обратным порядку сохранения.

В общем можно сказать, что все, что было сказано относительно подпрограмм, справедливо и для обработчиков прерываний. Важное отличие только в том, что момент прерывания непредсказуем, как было сказано, а поэтому вы не можете делать никаких умозаключительных предположений о том, в каком состоянии находится программа.

Что может делать обработчик прерывания? Это зависит от источника прерывания. Если, например, вы ожидаете прерывания, которое пришло от устройства по выводу микросхемы INT0, то, скорее всего, это означает на необходимость получить какие-то данные через порт, или начать прием информации через тот или иной интерфейс. Если вы получили прерывание от АЦП, то вам нужно записать значение из АЦП в память. Если вы получили прерывание от последовательного порта, значит по последовательному порту передан или принят очередной байт. Все эти действия выполняются во время обработки прерывания.

Обычно говорят, что подпрограмма обработки прерывания должна завершать свою работу как можно скорее. Это верно, когда вы программируете в системе, в которой исполняются параллельные процессы, или когда ваша система должна отвечать требованиям к системам реального времени. В обычном случае, когда, например, используется прерывание всего по одному вектору, обработчик прерывания может быть достаточно длинным.

В каждом конкретном случае нужно анализировать время работы обработчика прерывания и сопоставлять его с периодом возникновения прерываний не только по данному вектору, но и по другим векторам.

В качестве примера рассмотрим план организации программы с прерываниями, которая отслеживает нажатие клавиш.

Прежде всего нужно понять, какие действия следует выполнять в подпрограмме обработки прерывания, а какие — в основной программе. Программирование клавиатуры связано с необходимостью устранения дребезга. Для этого состояние клавиатуры проверяется каждые 20—50 мс. Учитывая, что полный цикл работы 16-ти разрядного таймера обеспечивает задержку примерно 32 мс, можно использовать полный цикл таймера для генерирования прерываний.

Далее нужно решить, какие действия будет выполнять подпрограмма обработки прерывания. Поскольку состояние клавиатуры может меняться каждые 32 мс, то подпрограмма обработки прерывания должна фиксировать факт наличия изменений, и устанавливать при этом какой-нибудь признак, сообщая таким образом основной программе о необходимости проверить клавиатуру.

Таким образом, вся сложность определения нажатия или отпускания конкретной клавиши ложится на подпрограммы основной программы, и на

обработчик прерывания возлагается более простая задача по определению текущего состояния клавиатуры и сравнению его с предыдущим.

Из всего этого следует, что требуется переменная для индикации изменения состояния клавиатуры и переменная для хранения предыдущего состояния клавиатуры, которые записываются обработчиком прерывания. Основная программа при этом отслеживает флаг изменения, и, обнаружив его установку, очищает его и вызывает подпрограмму, которая определяет нажатие или отпускание клавиши и выполняет связанные с этим действия.

Это не единственно возможный план. Другой вариант — обработчик прерывания берет на себя всю работу по отслеживанию нажатий и отпусканий клавиш, генерируя так же, как и в предыдущем случае, флаг изменения состояния клавиатуры. При этом нужно обеспечить время работы обработчика прерываний, меньшее времени между прерываниями.

При этом необходимо помнить также о том, что параллельно с прерываниями, обслуживающими обработку клавиатуры, в системе могут быть прерывания, обслуживающие другие события, и обработка прерывания событий клавиатуры не должна задерживать обслуживание других прерываний.

Например, если одновременно с обслуживанием клавиатуры происходит обработка прерываний АЦП, то имеет смысл назначить прерываниям АЦП высокий приоритет, чтобы прерывание АЦП прерывало обслуживание клавиатуры, так как обработка АЦП должна выполняться немедленно, а клавиатура может подождать.

Кроме того, следует учитывать, что один и тот же таймер может использоваться одновременно для разных целей, например для отслеживания состояния клавиатуры и для генерирования звукового сигнала. Задача в этом случае становится сложнее, так как придется разбираться с разными счетчиками.

В качестве второго примера рассмотрим план программы с прерываниями, обслуживающими генерацию звука.

Генерация звука в простейшем случае заключается в периодической смене напряжения на генераторе звука (зуммере). Следовательно, прерывание по таймеру должно изменять это напряжение. Однако, если генерируется какой-то периодически изменяющийся звук, то понадобится либо счетчик, либо дополнительный таймер со счетчиком для того, чтобы определять моменты, в которые звук генерируется, а в которые нет.

Таким образом, для управления звуковым генератором необходима переменная для хранения текущего значения напряжения на генераторе, переменная, которая подсчитывает время звучания, переменная, которая указывает на текущее состояние генератора (включен—выключен) и одну или две процедуры обслуживания прерываний от таймеров. Изменение этих переменных и генерацию звука берут на себя обработчики прерываний.

Программирование ПЗУ данных Flash/EEPROM

ПЗУ данных типа Flash/EEPROM предназначена для сохранения оперативных данных управляющей программы в перерывах между отключениями микроконтроллера. При использовании ПЗУ данных для записи следует придерживаться определенной последовательности действий:

1. записать адрес страницы в регистры EADRL и EADRH;
2. записать данные в регистры данных EDATA1—EDATA4;
3. записать в регистр ECON команду.

Перед записью информации в ПЗУ данных предварительно следует стереть имеющуюся информацию. При этом в байты записываются значения FFh. Так как стирание возможно только для страницы целиком, запись одного байта требует предварительного запоминания страницы, например, в регистрах данных EDATA1—EDATA4. Ниже приведен пример записи байта 1 в страницу 2:

```
MOV     EADRL, #2h    ; установка страницы
MOV     EADRH, #0
MOV     ECON, #1      ; чтение страницы
MOV     EDATA1, #33h  ; изменение байта 1
MOV     ECON, #5      ; стирание страницы
MOV     ECON, #2      ; запись страницы
MOV     ECON, #4      ; проверка записи
MOV     A, ECON
JNZ     WRITE_ERROR
```

Программирование Flash памяти занимает некоторое время, несовместимое со временем выполнения машинного цикла. Это следует учитывать при проектировании управляющей программы. В таблице 49 приведены временные интервалы, необходимые для операций с Flash памятью данных.

Таблица 49. Время чтения, записи и стирания Flash памяти данных

Операция	***Временной интервал***
Чтение страницы	22 машинных цикла
Запись страницы	380 мкс (~760 машинных циклов)
Проверка страницы	22 машинных цикла
Стирание страницы	2 мс (~4000 машинных циклов)
Стирание всей памяти	2 мс
Чтение прямоадресуемого байта	9 машинных циклов
Запись прямоадресуемого байта	200 мкс (~400 машинных циклов)

При выполнении операции с Flash памятью процессор приостанавливает выполнение последующих команд до окончания операции, однако таймеры и интерфейсы продолжают работать параллельно.

Следует также иметь ввиду, что первые два байта этой памяти хранят адрес старта текущей рабочей программы, поэтому первую страницу переписывать нельзя.

4. Программирование устройств SDK1.1

Учебный стенд SDK1.1 оснащен 16-ти клавишной клавиатурой, линейкой из восьми светодиодов, двухстрочным матричным дисплеем (ЖКИ), зуммером (устройством для воспроизведения звука), а также микросхемой календаря и микросхемой дополнительной памяти типа EEPROM, подключенными к микроконвертеру через интерфейс I^2C.

Перечисленных устройств достаточно, чтобы изучить основы программирования учебного стенда и программирования микроконтроллеров в общем. Кроме этого, при помощи дополнительных внешних схем учебный стенд может быть подключен к источникам двоичных и аналоговых сигналов, и служить в качестве, например, измерительного устройства, или устройства управления. Использование учебного стенда во взаимодействии с внешними (по отношению к микроконтроллеру-прибору) устройствами выходит за рамки данного пособия.

Все устройства учебного стенда представлены в архитектуре микроконтроллера своими регистрами (за исключением микросхем календаря и дополнительной памяти, которые представлены своими адресами в адресном пространстве интерфейса I^2C). Эти регистры располагаются в адресном пространстве внешней памяти данных (ВПД).

Микроконтроллер оснащен 512 Кбайт ОЗУ, которое используется как внешняя память данных. Обращение к ВПД происходит косвенно через регистр-указатель данных DPTR. Как было сказано ранее, этот регистр фактически состоит из трех регистров DPL, DPH и DPP.

При помощи двух регистров, DPL и DPH, можно адресовать 64 Кбайт памяти. Регистр DPP используется как переключатель страниц ВПД. 512 Кбайт составляют 8 страниц по 64 Кбайт, с номерами от 0 до 7. Страница с номером 8 используется для адресации регистров устройств учебного стенда. Адреса и назначение регистров приведены в таблице 50.

Таблица 50. Адреса устройств учебного стенда

Адрес	***Регистр***	***Доступ***	***Назначение***
08 00 00h	KB	R/W	Регистр клавиатуры
08 00 01h	DATA_IND	R/W	Регистр шины данных ЖКИ
08 00 02h	EXT_LO	R/W	Регистр параллельного порта (разряды 0—7)
08 00 03h	EXT_HI	R/W	Регистр параллельного порта (разряды 8—15)
08 00 04h	ENA	W	Регистр управления портами в/в, звуком, INT0
08 00 06h	C_IND	W	Регистр управления ЖКИ
08 00 07h	SV	W	Регистр управления светодиодами

В столбце «Адрес» приведен полный шестнадцатеричный адрес устройства в ВПД. В столбце «Регистр» приведено название регистра, данное ему изготовителем стенда. В столбце «Доступ» указано, какие операции можно производить с данным регистром: R обозначает возможность чтения регистра, W обозначает возможность записи.

Для обращения к конкретному регистру в регистры DPTR нужно записать адрес регистра: младшие две цифры в регистр DPL, средние две шестнадцатеричных цифры — в регистр DPH, и число 8 в регистр DPP (адрес страницы внешних устройств микроконвертера). Например, для обращения к регистру светодиодов нужно записать DPL=07, PDH=0, DPP=08.

Линейка светодиодов

Линейка светодиодов представляет собой восемь расположенных в ряд светодиодов (светящихся индикаторов), предназначенных для начального обучения программированию микроконтроллера, а также для индикации различных состояний устройства (управляющей программы). Это самое простое устройство, которое можно запрограммировать в учебном стенде.

В таблице 51 приведено назначение битов регистра светодиодов.

Таблица 51. Регистр светодиодов CV (08 00 07h)

Разряд	***7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***
Доступ	W	W	W	W	W	W	W	W
Назначение	Управление свечением одного из светодиодов линейки							

Каждый разряд этого регистра отвечает за один светодиод линейки светодиодов. При записи в некоторый разряд значения «1» светодиод, связанный с этим разрядом, включается (начинает светиться), а при записи значения «0» — выключается.

Следует иметь ввиду, что с помощью этого регистра нельзя управлять состоянием только одного светодиода. При записи в регистр изменяется состояние сразу всех светодиодов в соответствии с записанным в регистр значением. Нельзя также узнать состояние светодиодов, поскольку регистр можно только записывать. Поэтому при программировании линейки светодиодов необходимо хранить текущее состояние всех светодиодов в какой-нибудь переменной или в одном из регистров.

В качестве примера рассмотрим, как можно непосредственно включить, а затем через некоторое время выключить светодиод, связанный с разрядом 0. В примере переменная для хранения состояния линейки не используется.

Прежде всего нужно записать в регистр указателя данных DPTR адрес регистра светодиодов:

```
MOV     DPL,#7
MOV     DPH,#0
MOV     DPP,#8
```

Далее в аккумулятор A записывается значение 01h (00000001b), которое обозначает выключение всех светодиодов, кроме первого:

```
MOV     A,#1
```

Затем значение из аккумулятора A записывается в регистр светодиодов при помощи команды косвенной пересылки через указатель данных DPTR:

```
MOVX    @DPTR,A
```

Сразу после выполнения этой команды все светодиоды, кроме первого, будут погашены, а первый «загорится».

Далее вызывается подпрограмма задержки, названная в примере DELAY. Она выдерживает некоторый интервал времени, в течение которого светодиод будет включен:

```
LCALL   DELAY
```

Наконец, записываем в аккумулятор значение 0 (очищаем аккумулятор) и пересылаем его в регистр светодиодов, чтобы погасить всю линейку:

```
CLR     A
MOVX    @DPTR,A
```

Клавиатура

В таблице 52 приведено назначение битов регистра клавиатуры KB.

Таблица 52. Регистр клавиатуры KB (08 00 00h)

Разряд	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Доступ	R	R	R	R	W	W	W	W
Назначение	ROW4	ROW3	ROW2	ROW1	COL4	COL3	COL2	COL1
Поле	ROW				COL			

Клавиатура представляет собой матрицу замыкающих контактов размером 4×4 (рисунок 55).

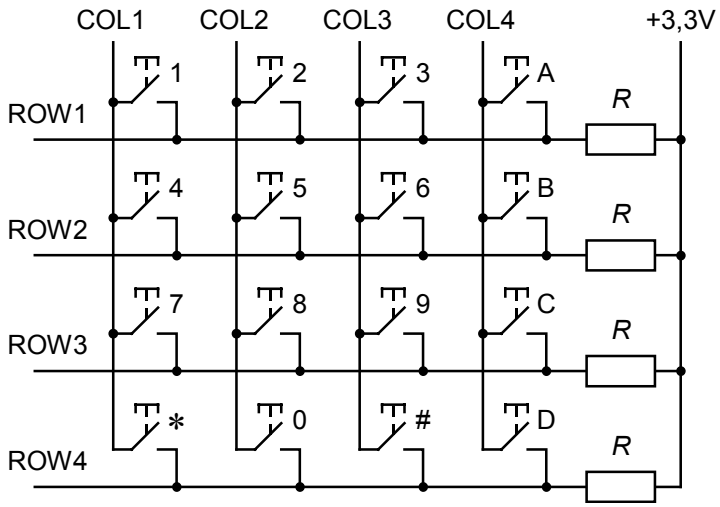


Рисунок 55. Матрица клавиатуры

Каждая клавиша подключена к одному из столбцов матрицы, обозначенных COLn, и к одному из рядов, обозначенных ROWn, 1≤n≤4.

Линии рядов ROWn через резисторы R подключены к напряжению логической единицы, поэтому при чтении регистра KB, когда не нажата ни одна из клавиш, в поле ROW будут считаны единицы в разрядах 4—7.

Чтобы определить, нажата или нет определенная клавиша, в поле COL регистра клавиатуры нужно записать нулевое значение в тот разряд, который соответствует столбцу клавиши, и единичные значения в другие разряды.

Сразу после этого регистр клавиатуры нужно прочитать и протестировать поле ROW. Если разряд, соответствующий строке клавиши, содержит нулевое значение, клавиша нажата, иначе отпущена. Другие разряды при этом будут указывать на состояние остальных клавиш тестируемого столбца. Таким образом, за два обращения к регистру KB можно проверить нажатие 4-х клавиш.

Рассмотрим пример кода, в котором производится запись нулевого значения в разряд столбца COL1, затем чтение поля ROW и определение нажатия клавиши, обозначенной «1», по наличию нуля в разряде ROW1 регистра KB.

Предварительно нужно установить адрес регистра клавиатуры в регистры указателя данных DPTR:

```
MOV     DPL,#0
MOV     DPH,#0
MOV     DPP,#8
```

Далее записываем значение 0FEh (11111110b) сначала в аккумулятор:

```
MOV     A,#0FEh
```

затем аккумулятор косвенно через регистр DPTR записываем в регистр клавиатуры:

```
MOVX    @DPTR,A
```

Сразу после этого считываем регистр клавиатуры в аккумулятор:

```
MOVX    A,@DPTR
```

после чего проверяем наличие нулевого значения в разряде 4, соответствующему строке ROW1:

```
ANL     A,#010h
```

Операция ANL (AND) с маской 010h (00010000b) маскирует все разряды, кроме разряда 4. Таким образом, если после этой операции аккумулятор равен нулю, то разряд 4 также равен нулю и клавиша «1» в момент тестирования была нажата, иначе нет.

Для того, чтобы определить нажатое состояние произвольной клавиши, необходим какой-нибудь метод. Например, в памяти программ можно организовать массив из 16-ти элементов данных, каждый из которых соответствует нажатому состоянию одной из клавиш. Если расположить массив в начале программы, то его нужно обойти при помощи инструкции SJMP. В примере метка START обозначает начало программного кода:

```
    SJMP    START
KEYS:
    DB 07Dh,0EEh,0EDH,0EBh,0DEh,0DDh,0DBh,0BEh
    DB 0BDh,0BBh,07Eh,07Bh,0E7h,0D7h,0B7h,077h
START:
```

Элементы массива сформированы таким образом, чтобы обеспечить последовательность, в которой первые 10 состояний соответствуют нажатым клавишам «0—9», затем идут состояния для клавиш «*» и «#», затем для клавиш, обозначенных «A—D». Элементы массива содержат нули в тех разрядах регистра клавиатуры, которые соответствуют столбцу и ряду некоторой клавиши.

Для определения нажатия клавиши нужно выделить либо ячейку памяти, либо регистр для хранения текущего индекса в массиве. В примере используется регистр R4. Изначально в него записывается значение 0, которое соответствует нулевому (первому) элементу массива и клавише «0»:

```
    MOV     R4,#0
```

Поскольку вычисления происходят в цикле, выполняем предварительные действия для организации цикла. Записываем в регистр R3, используемый для подсчета итераций, значение 16 (равное количеству клавиш):

```
    MOV     R3,#16
```

Далее считываем элемент массива, на который указывает регистр R4, в аккумулятор. Сначала в регистр DPTR записываем значение метки KEYS (адрес массива). При помощи инструкции MOVC A,@A+DPTR считываем элемент массива из памяти программ в аккумулятор. Адрес элемента массива складывается из адреса массива в DPTR и индекса элемента массива в A. Запоминаем элемент массива в регистре R2. Перед этой секцией кода находится метка цикла KLOOP:

```
KLOOP:
    MOV     A,R4
    MOV     DPTR,#KEYS
    MOVC    A,@A+DPTR
    MOV     R2,A
```

Далее устанавливаем адрес регистра клавиатуры в регистр указателя данных DPTR, записываем аккумулятор в регистр клавиатуры, и сразу считываем регистр клавиатуры в аккумулятор:

```
    MOV     DPL,#0
    MOV     DPH,#0
    MOV     DPP,#8
    MOVX    @DPTR,A
    MOVX    A,@DPTR
```

Теперь осталось сравнить аккумулятор с элементом массива, который находится в регистре R2. Выполним над ними операцию XRL (XOR):

```
XRL     A,R2
```

Если в результате этой операции аккумулятор содержит нулевое значение, значит перед операцией аккумулятор A и регистр R2 содержали одинаковые значения и, соответственно, клавиша, на которую указывает регистр R4, была в момент тестирования нажата. Используем инструкцию JZ для выхода из цикла и перехода на метку PRESSED, объявленную в программе ниже:

```
JZ      PRESSED
```

После этой инструкции следует инструкция цикла, за которой может быть расположена метка PRESSED:

```
DJNZ    R3,KLOOP
    ; никакая клавиша не нажата
PRESSED:
    ; нажата какая-то клавиша
```

При программировании клавиатуры необходимо иметь ввиду важную особенность всех клавиатур, называемую «дребезг». Он проявляется в том, что в момент нажатия или отпускания клавиши клавиатуры происходят многократные переключения из нажатого состояния в не нажатого и наоборот. Условно процесс нажатия и отпускания клавиши изображен на рисунке 56.

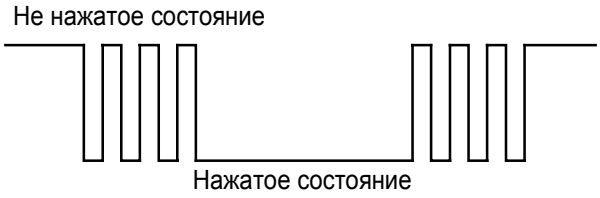


Рисунок 56. Дребезг клавиш при нажатии и отпускании

Из-за дребезга клавиш определение состояния клавиатуры становится нетривиальной задачей. Длительность «дребезга» различна для разных типов клавиатур и зависит также от текущего состояния клавиатуры. Ориентировочно длительность можно принять равной 20—50 мс. Учитывая, что 16-ти разрядный таймер 0 или 1 микроконвертера позволяет отсчитать максимальный интервал примерно 32 мс, полный цикл работы таймера можно принять в качестве интервала, в течение которого следует проверять состояние клавиши.

При этом, если в какой-то момент времени обнаружено нажатое состояние клавиши, которая до этого не была нажата, то если по истечение заданного интервала τ клавиша нажата, то следует принять, что она нажата. Аналогично, если в какой-то момент времени обнаружено, что клавиша не нажата, при этом до этого было зафиксировано ее нажатое состояние, то если по истечении заданного интервала τ клавиши не нажата, то следует принять, что клавишу отпустили (рисунок 57).

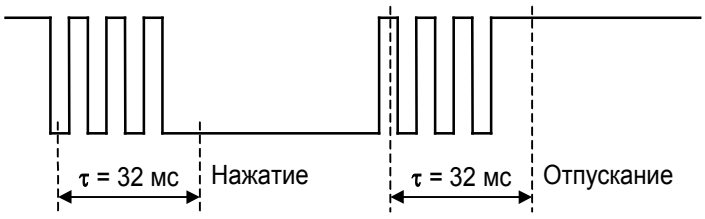


Рисунок 57. Отслеживание дребезга клавиши

Для полного сканирования клавиатуры, вероятно, понадобится массив значений, указывающих на нажатое или не нажатое состояние каждой контролируемой клавиши, а также признак нажатия любой клавиши, а опрос состояния клавиатуры производить по прерыванию от таймера 0 или 1.

Матричный дисплей (ЖКИ)

Матричный дисплей (в дальнейшем просто ЖКИ, жидкокристаллический индикатор) представлен в адресном пространстве ВПД двумя регистрами: регистром данных DATA_IND и регистром управления C_IND. Регистр данных предназначен для записи информационных байтов, предназначенных для отображения и для установки режимов дисплея, а регистр управления — для организации процесса записи информационных байтов непосредственно в ЖКИ.

Регистр данных DATA_IND предназначен как для записи информации, так и для её чтения. Поскольку ЖКИ содержит в себе память для хранения информации, эту память можно использовать не только для отображения, но и для обычного временного хранения информации.

Полное понимание принципов программирования ЖКИ возможно только после детального ознакомления с документацией на ЖКИ, которая прилагается к учебному стенду. Сам ЖКИ представляет собой микроконтроллер, имеющий в своем составе специализированный микропроцессор, память, рабочие регистры и схемы управления. ЖКИ подключается к внешнему устройству (в качестве которого в данном случае выступает учебный стенд) при помощи шины данных и шины управления. Регистры DATA_IND и C_IND являются отображением этого интерфейса ЖКИ. В пособии кратко рассматриваются только общие принципы программирования этого устройства, необходимые для выполнения практических работ.

Для отображения информации на ЖКИ необходимо сначала задать режим его работы, а затем записывать в память, называемую DDRAM, информационные байты для отображения. Мы рассматриваем работу ЖКИ только в двухстрочном режиме с 8-ми разрядным интерфейсом, с использованием зашитого в ПЗУ ЖКИ знакогенератора.

Знакогенератор ЖКИ позволяет отображать цифры, знаки, латиницу, отдельные прописные буквы кириллицы, греческие символы. Недостаток встроенного знакогенератора — часть символов кириллицы нужно кодировать символами латиницы.

В таблице 53 приведены коды символов, которые встроены непосредственно в ЖКИ (выборочно, т.к. некоторые символы не имеют аналогов в ASCII и в редакторе Microsoft Word). Коды символов шестнадцатеричные.

Таблица 53. Таблица символов ЖКИ

Код	Знак	Код	Знак	Код	Знак	Код	Знак	Код	Знак	Код	Знак
10	►	30	0	50	P	70	p	90	α	B0	°
11	◄	31	1	51	Q	71	q	91	♪	B1	±
12	“	32	2	52	R	72	r	92	Г	B2	²
13	”	33	3	53	S	73	s	93	π	B3	³
14		34	4	54	T	74	t	94	Σ	B4	
15		35	5	55	U	75	u	95	σ	B5	µ
16	•	36	6	56	V	76	v	96	♫	B6	¶
17	↵	37	7	57	W	77	w	97	т	B7	
18	↑	38	8	58	X	78	x	98		B8	
19	↓	39	9	59	Y	79	y	99	θ	B9	¹
1A	→	3A	:	5A	Z	7A	z	9A	Ω	BA	
1B	←	3B	;	5B	[	7B	{	9B	δ	BB	»
1C	≤	3C	<	5C	\	7C	|	9C		BC	¼
1D	≥	3D	=	5D	]	7D	}	9D		BD	½
1E	▲	3E	>	5E	^	7E	~	9E	ε	BE	¾
1F	▼	3F	?	5F	_	7F	⌂	9F	∩	BF	¿
20	space	40	@	60	`	80	Б	A0		D7	×
21	!	41	A	61	a	81	А	A1		F7	÷
22	@	42	B	62	b	82	Ж	A2			
23	#	43	C	63	c	83	З	A3			
24	$	44	D	64	d	84	И	A4	¤		
25	%	45	E	65	e	85	Й	A5	¥		
26	&	46	F	66	f	86	Л	A6			
27	’	47	G	67	g	87	П	A7	§		
28	(	48	H	68	h	88	У	A8	*ƒ*		
29	)	49	I	69	i	89	Ц	A9			
2A	*	4A	J	6A	j	8A	Ч	AA			
2B	+	4B	K	6B	k	8B	Ш	AB	«		
2C	,	4C	L	6C	l	8C	Щ	AC	Ю		
2D	–	4D	M	6D	m	8D	Ъ	AD	Я		
2E	.	4E	N	6E	n	8E	Ы	AE			
2F	/	4F	O	6F	o	8F	Э	AF	‘		

В целом можно сказать, что знакогенератор позволяет отображать достаточно много символов, чтобы организовать интерфейс пользователя для разного рода задач.

Другие особенности ЖКИ включают в себя возможность отображения курсора, сдвиг информации в строке вправо и влево, установка текущей позиции (курсора) в начало дисплея, очистка всего дисплея и другие.

В двухстрочном режиме информация на дисплее выводится в две строки по 16 знаков. Для отображения какой-либо информации нужно записать информационные байты во внутреннюю видеопамять ЖКИ под названием DDRAM. Эта память в двухстрочном режиме содержит 80 знакомест, имеющие адреса от 0 до 39 (от 0h до 27h) для первой отображаемой строки, и от 64 до 103 (от 40h до 67h) для второй отображаемой строки.

Соответствие между адресом памяти и позицией знака на дисплее показано в таблице 54. В первой строке таблицы указан номер позиции в строке, в последующих двух строчках указаны шестнадцатеричные адреса видеопамяти, соответствующие этим позициям.

Таблица 54. Соответствие адреса видеопамяти и знакоместа дисплея

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Операция с ЖКИ представляет собой команду. Перечень команд приведен в таблице 55. Обозначения таблицы 55 расшифрованы в таблице 56.

Таблица 55. Команды ЖКИ

Команда			Код команды								Время исп.
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
Очистка дисплея	0	0	0	0	0	0	0	0	0	1	1,53 мс
Курсор в начало дисплея	0	0	0	0	0	0	0	0	1	1	1,53 мс
Направление движения, движение курсора или дисплея	0	0	0	0	0	0	0	1	I/D	SH	39 мкс
Вкл./выкл. дисплей (D), курсор (C) или мерцание курсора (B)	0	0	0	0	0	0	1	D	C	B	39 мкс
Сдвиг курсора или дисплея и направление сдвига	0	0	0	0	0	1	S/C	R/L	1	1	39 мкс
Режим дисплея	0	0	0	0	1	DL	N	F	1	1	39 мкс
Адрес знакогенератора CGRAM	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	39 мкс
Адрес видеопамяти DDRAM	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	39 мкс
Флаг занятости и адрес DDRAM	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	0 мкс
Запись данных в DDRAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	43 мкс
Чтение данных из DDRAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	43 мкс

Таблица 56. Обозначения таблицы 55

	0	***1***
I/D	инкремент позиции при записи	декремент позиции при записи
SH	при записи знака движется курсор	при записи знака движется дисплей
S/C	сдвиг курсора	сдвиг дисплей
R/L	сдвиг дисплея или курсора влево	сдвиг дисплея или курсора вправо
DL	интерфейс 4 разряда	интерфейс 8 разрядов
N	одна строка	две строки
F	знакоместо 5×8 точек	знакоместо 5×10 точек
BF	ЖКИ свободен	ЖКИ выполняет операцию

Элементы команд, расположенные в таблице 54 в столбцах, обозначенных DB0—DB7, записываются или считываются через регистр данных DATA_IND. Для выполнения операции используется регистр управления C_IND. Состав этого регистра приведен в таблице 57.

Таблица 57. Регистр управления ЖКИ (08 00 06h)

Разряд	***7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***
Доступ	—	—	—	—	W	W	W	W
Назначение	—	—	—	—	1	RS	RW	E

Переход значения разряда E в регистре C_IND из единичного в нулевое фиксирует данные в регистре данных DATA_IND на шине данных ЖКИ. К этому моменту данные (команда или код символа) должны быть записаны в регистр DATA_IND.

Значение разряда RW, равное «0», указывает на запись информации из регистра данных DATA_IND в ЖКИ, а значение, равное «1» — чтение информации из ЖКИ в регистр DATA_IND.

Если значение разряда RS равно «0», то данные в регистре DATA_IND являются командой управления, а если значение равно «1» — то данные в регистре DATA_IND являются адресом видеопамяти или кодом знака в текущей позиции.

Для выполнения команды управления сначала команда записывается в регистр DATA_IND, затем в регистр C_IND сначала записываются значения разрядов RS и RW при единичном значении разряда E, и после этого те же значения RS и RW записываются при нулевом значении разряда E. Последовательная запись в регистр C_IND двух значений, отличающихся только разрядом E, создает необходимый переход разряда E, по которому данные поступают в ЖКИ и вызывают выполнение им затребованных действий.

Для записи знака в видеопамять сначала нужно установить текущую позицию (address counter, AC) в видеопамяти при помощи команды 001AAAAAAA (A обозначает одну двоичную цифру семиразрядного адреса), затем записать код знака при помощи команды 00CCCCCCCC (C обозначает одну двоичную цифру восьмиразрядного кода знака). Сразу после этого, если адрес знака находится в видимой части дисплея, знак должен быть отображен в заданной позиции. Двукратная запись в регистр C_IND здесь также необходима, как и для записи команды управления.

После включения питания учебного стенда дисплей автоматически инициализируется в однострочный режим при 8-ми разрядном интерфейсе. Кроме того, находящееся в стенде базовое программное обеспечение («зашитый» по адресу 0E000H загрузчик HEX02-03) инициализирует дисплей в двухстрочный режим. Поэтому при программировании дисплея нет необходимости выполнять его инициализацию — она возникает только в случае смены базового программного обеспечения. В этом случае следует обратиться к документации.

В качестве примера программирования ЖКИ разберем вывод на дисплей фразы «Go to OTI MEPHI to learn more about MC. » (фраза содержит ровно 40 знаков).

Для организации вывода потребуется разработать подпрограмму для вывода одного знака. Название подпрограммы DISPA, выводимый знак записывается в аккумулятор перед вызовом подпрограммы. Прежде всего подпрограмма устанавливает в DPTR адрес регистра данных DATA_IND:

```
DISPA: ; вывод символа
    MOV     DPP,#8
    MOV     DPH,#0
    MOV     DPL,#1
```

Затем код знака, находящийся в аккумуляторе A, записывается в регистр данных DATA_IND и вызывается подпрограмма задержки 40 мкс, чтобы выдержать время, необходимое ЖКИ для обработки информации:

```
    MOVX    @DPTR,A
    LCALL   D40MCS
```

Чтобы символ появился на дисплее, нужно обеспечить смену состояния разряда E регистра C_IND из единичного в нулевое, поэтому далее необходимо изменить адрес ВПД на адрес регистра управления C_IND и записать в него код, содержащий установленные разряды RS (запись данных) и E (единичное состояние). После записи также вызывается подпрограмма задержки на 40 мкс:

```
    MOV     DPL,#6
    MOV     A,#0Dh
    MOVX    @DPTR,A
    LCALL   D40MCS
```

Далее нужно записать те же данные в регистр управления C_IND, но с разрядом E, равным нулю, после чего подпрограмма завершается:

```
    MOV     A,#0Ch
    MOVX    @DPTR,A
    LCALL   D40MCS
    RET
```

Кроме подпрограммы для записи знака потребуется также подпрограмма для записи команды. Название подпрограммы CDISP, команда для записи также передается через аккумулятор A.

Подпрограмма для записи кода аналогична подпрограмме для записи знака. Отличия заключаются в использовании задержки в 2 мс (а не 40 мкс) после записи команды в регистр данных DATA_IND, а также в кодах, которые записываются в регистр управления C_IND. Поскольку в ЖКИ записывается команда, разряд RS принимается равным нулю, и управляющие коды получаются равными 9 и 8:

```
CDISP: ; запись команды
```

```
    MOV     DPP,#8
    MOV     DPH,#0
    MOV     DPL,#1
    MOVX    @DPTR,A
    LCALL   D2MS
    MOV     DPL,#6
    MOV     A,#9
    MOVX    @DPTR,A
    LCALL   D40MCS
    MOV     A,#8
    MOVX    @DPTR,A
    LCALL   D40MCS
    RET
```

Начало программы содержит фразу как элемент данных. Для того, чтобы обойти этот элемент, первая инструкция программы — короткий переход на метку START:

```
    SJMP    START
DB  'Go to OTI MEPHI to learn more about MC. '
START:
```

Далее устанавливаем текущую позицию АС для записи в дисплей и инициализируем считывание данных из памяти программ и цикл из 40 итераций:

```
    MOV     A,#080h     ; команда адрес памяти 0
    LCALL   CDISP
    MOV     R1,#0       ; адрес знака в строке
    MOV     R0,#40      ; цикл 40 раз
```

Последующий цикл считывает по одному знаки из строки данных в аккумулятор А и выводит их на дисплей при помощи подпрограммы DISPA:

```
DLOOP:
    MOV     DPTR,#MESSAGE
    MOV     A,R1
    MOVC    A,@A+DPTR
    LCALL   DISPA
    INC     R1
    DJNZ    R0,DLOOP
```

Программа завершается «мертвым» циклом:

```
DEAD: ; мертвый цикл
    SJMP    DEAD
```

Заметим, что, поскольку на дисплее может быть отображено всего 16 знаков в строке, мы увидим только начало фразы.

Чтобы увидеть фразу целиком, ее можно «прокрутить», сдвигая дисплей 40 раз, например, влево. Для этого после цикла записи строки данных

в дисплей и перед мертвым циклом нужно организовать цикл сдвига дисплея и использовать внутри него команду 1Bh (0000011011b). Чтобы строка прокручивалась достаточно медленно, после каждого сдвига дисплея нужно вызывать подпрограмму задержки на время 200—250 мс:

```
    MOV     R0,#40          ; цикл 40 раз
SLOOP:
    MOV     A,#1Bh          ; команда сдвиг дисплея влево
    LCALL   CDISP
    LCALL   D250MS
    DJNZ    R0,SLOOP
```

Чтобы отобразить курсор, его нужно включить. Следующий код можно расположить после цикла прокручивания строки:

```
    MOV     A,#0Eh          ; команда включить дисплей и курсор
    LCALL   CDISP
```

При этом курсор появится во второй строке, поскольку первая строка была полностью заполнена. Чтобы переместить курсор в начало дисплея, нужно далее записать команду 3 (0000000011b):

```
    MOV     A,#3            ; команда курсор в начало
    LCALL   CDISP
```

Далее при помощи еще одного цикла можно заставить перемещаться курсор вправо по длине строки при помощи команды 17h (0000010111b):

```
    MOV     R0,#40          ; цикл 40 раз
CLOOP:
    MOV     A,#17h          ; команда сдвиг курсора вправо
    LCALL   CDISP
    LCALL   D250MS
    DJNZ    R0,CLOOP
```

После выполнения всего цикла курсор появится во второй строке дисплея.

Заметим, что ЖКИ требуется время для выполнения любой команды. Это время приведено в таблице 54, и должно обязательно выдерживаться, иначе команда будет проигнорирована.

В те моменты времени, когда дисплей занят выполнением очередной команды, он выставляет флаг BF (busy flag). Этот флаг можно использовать для того, чтобы определять момент, когда дисплей готов принять очередную команду.

Для проверки флага BF нужно внести изменения в подпрограмму DISPA, заменив вызов подпрограммы задержки D40MCS вызовом подпрограммы BWAIT проверки состояния дисплея и ожидания, когда дисплей освободится.

Подпрограмма ожидания готовности дисплея сначала создает переход из единичного состояния разряда E в нулевое при установленном разряде

RW при помощи двух команд записи в регистр управления C_IND, затем считывает в аккумулятор A состояние дисплея из регистра данных DATA_IND и проверяет значение старшего разряда аккумулятора, маскируя другие разряды при помощи операции ANL (AND). Чтение флага BF не требует задержек:

```
BWAIT: ; ожидание готовности дисплея
    MOV     DPP,#8
    MOV     DPH,#0
BWAIT1:
    MOV     DPL,#6
    MOV     A,0Bh
    MOVX    @DPTR,A
    MOV     A,0Ah
    MOVX    @DPTR,A
    MOV     DPL,#1
    MOVX    A,@DPTR
    ANL     A,#80h
    JNZ     BWAIT1
    RET
```

Календарь

Микросхема календаря также представляет собой достаточно сложное устройство, обладающее большим набором функций. Полное описание этой микросхемы приведено в документации, прилагаемой к учебному стенду. В пособии рассматриваются только те элементы микросхемы, которые относятся к отсчету времени.

В микросхеме календаря находятся 256 8-ми разрядных ячеек памяти. Первые 8 ячеек содержат текущее состояние микросхемы и данные, описывающие время и дату, следующие 8 ячеек используются для сигнализации, оставшиеся 240 ячеек используются произвольным образом. В таблице 58 приведено назначение первых восьми ячеек памяти.

Таблица 58. Назначение ячеек памяти микросхемы календаря

Адрес	*Назначение*
0	регистр управления/состояния
1	доли секунды
2	секунды
3	минуты
4	часы
5	год/дата
6	день недели/месяц
7	таймер

Регистр управления состояния (адрес 0) предназначен для задания режима работы микросхемы и проверки её состояния. Поскольку этот ре

гистр не требуется записывать или считывать, здесь значения его разрядов не приводятся.

Время и дата записываются в регистры

Он имеет формат, приведенный в таблице 59.

Таблица 59. Регистр состояния/управления микросхемы календаря

Разряд	***Назначение***
7	Флаг остановки счета
6	Флаг сохранения последнего считывания
5	Режим: 0 — часы
4	Режим: 0 — часы
3	Маскирование дня недели и года при чтении даты
2	Разрешение сигнала
1	Флаг сигнала
0	Флаг таймера

В этом регистре единичное значение бита 3 позволяет при считывании даты и месяца замаскировать хранящиеся в тех же регистрах год и день недели.

Время записывается в регистры с адресами 1—4 в двоично-кодированном десятичном формате. Формат регистров сотых долей секунды, секунд, минут и часов приведен в таблице 60.

Таблица 60. Регистр сотых долей секунды

Разряд	***Назначение***
4—7	Старшая десятичная цифра
0—3	Младшая десятичная цифра

Число часов, минут, секунд и сотых долей записывается в виде десятичного числа, младшая цифра числа кодируется в младшей тетраде, а старшая — в старшей.

Регистр с адресом 5 хранит год и число месяца в формате, приведенном в таблице 61.

Таблица 61. Регистр года и даты

Разряд	***Назначение***
6—7	Год (0—3)
4—5	Старшая цифра даты (0—3)
0—3	Младшая цифра даты (0—9)

Год представляет собой число от 0 до 3, поэтому максимальный отсчет времени — 4 года. Программист должен прибавить к числу, хранимому в микросхеме календаря некоторое число для того, чтобы получить год, соответствующий реальному, и выводить полученное число. При вводе года, наоборот, число нужно вычитать. Следует учитывать, что год, обозначаемый нулем, должен быть високосным.

Регистр с адресом 6 хранит день недели и месяц. В таблице 62 приведен формат этого регистра.

Таблица 62. Регистр дня недели и месяца

Разряд	*Назначение*
5—7	День недели (0—6)
4	Старшая цифра месяца (0—1)
0—3	Младшая цифра месяца (0—9)

Программирование микросхемы календаря заключается в установке и считывании времени и даты (а также в выполнении других функций, которые здесь не рассматриваются). Микросхема подключена к микроконвертеру при помощи интерфейса I^2C, поэтому программирование микросхемы является достаточно сложной и интересной задачей. Описание интерфейса I^2C приведено выше в соответствующем разделе. Здесь мы рассмотрим, как практически можно прочитать значение сотых долей секунды и вывести их на дисплей.

Управление интерфейсом микроконвертера I^2C в режиме мастера (а микроконвертер при работе с микросхемой календаря выступает в качестве мастера) осуществляется при помощи битов MDE, MDI, MDO и MCO регистра I2CCON. Бит MCO отвечает за сигналы синхронизации SCL.

Прежде всего нужно подготовить несколько вспомогательных подпрограмм, которые будут выполнять отдельные простые действия по управлению шинами интерфейса. Первая подпрограмма предназначена для формирования состояния интерфейса «старт». Она называется I2CSTART:

```
I2CSTART: ; последовательность "старт"
    SETB    MDO             ; SDA = 1
    LCALL   D6MCS
    SETB    MCO             ; SCL = 1
    LCALL   D6MCS
    CLR     MDO             ; SDA = 0
    LCALL   D6MCS
    CLR     MCO             ; SCL = 0
    LCALL   D6MCS
    RET
```

Сначала подпрограмма формирует высокий уровень линии данных, а затем высокий уровень линии синхронизации, после чего на линию данных записывается низкий уровень, затем понижается и уровень синхросигнала. В результате данная последовательность команд формирует сигналы, показанные на рисунке 58 слева.

После каждой смены состояния линий в подпрограмме I2CSTART вызывается подпрограмма задержки на 6 мкс, требуемая для обеспечения временных задержек интерфейса I^2C (см. рисунок 50). На рисунке 58 временная задержка обозначена символом τ.

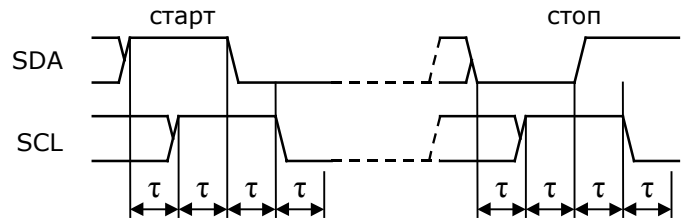


Рисунок 58. Формирование состояний «Старт» и «Стоп»

Аналогичным образом формируется последовательность «стоп», показанная на рисунке 58 справа. Подпрограмма формирования этой последовательности в примере называется I2CSTOP:

```
I2CSTOP: ; последовательность "стоп"
    CLR     MDO             ; SDA = 0
    LCALL   D6MCS
    SETB    MCO             ; SCL = 1
    LCALL   D6MCS
    SETB    MDO             ; SDA = 1
    LCALL   D6MCS
    CLR     MCO             ; SCL = 0
    LCALL   D6MCS
    RET
```

Для записи на линии интерфейса байта информации предназначена следующая подпрограмма, названная I2CPUTA. Записываемый байт находится в аккумуляторе:

```
I2CPUTA: ; запись аккумулятора в устройство
    MOV     R6,#8
I2CPUTA1:
    RLC     A               ; сдвигаем старший бит в CY
    MOV     MDO,C           ; SDA = передаваемый бит
    LCALL   D6MCS
    SETB    MCO             ; SCL = 1
    LCALL   D6MCS
    CLR     MCO             ; SCL = 0
    LCALL   D6MCS
    DJNZ    R6,I2CPUTA1
```

Подпрограмма представляет собой цикл, в каждой итерации которого на линию записывается один бит. Сначала в регистр R6 записывается количество передаваемых битов. Затем в ходе очередной итерации старший бит аккумулятора выдвигается во флаг переноса CY, а флаг переноса выставляется на линию данных MDO, после чего следует временная задержка.

Далее формируется синхроимпульс, который должен быть уже, чем сигнал данных. На рисунке 59 показан момент записи одного бита данных. Следующий записываемый бит примыкает к записанному.

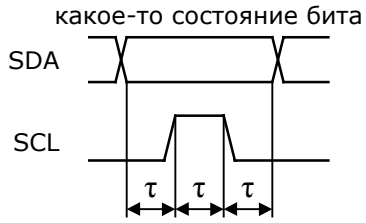


Рисунок 59. Запись одного бита данных на линии I^2C

После записи байта данных согласно протоколу передатчик обязан принять девятый бит подтверждения acknowledge. Поэтому подпрограмма после завершения цикла переводит линию данных в режим чтения, очищая бит MDE, формирует синхроимпульс и в течение синхроимпульса считывает бит с шины SDA интерфейса в бит CY. Чтобы не потерять бит подтверждения, подпрограмма записывает его в некоторый бит битового поля, названный I2CACK. По завершении импульса синхронизации устанавливается бит MDE, возвращая интерфейс в режим записи:

```
; получим подтверждение
CLR     MDE
LCALL   D6MCS
SETB    MCO             ; SCL = 1
LCALL   D6MCS
MOV     C,MDI           ; чтение SDA
MOV     I2CACK,C
LCALL   D6MCS
CLR     MCO             ; SCL = 0
LCALL   D6MCS
SETB    MDE
MOV     C,I2CACK
RET
```

Перед выходом из подпрограммы I2CPUTA сохраненный бит подтверждения записывается во флаг переноса CY для последующего анализа основной программой или подпрограммой. На рисунке 60 приведена диаграмма приема бита подтверждения.

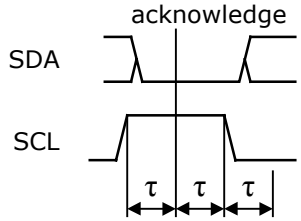


Рисунок 60. Прием бита подтверждения acknowledge

Следующая подпрограмма читает передаваемый устройством байт с линий интерфейса. Название подпрограммы I2CGETA. В ней нет ничего нового по отношению к тому, что уже было сказано:

```
I2CGETA:
```

```
    MOV     R6,#8
    CLR     MDE
I2CGETA1:
    SETB    MCO             ; SCL = 1
    LCALL   D6MCS
    MOV     C,MDI           ; чтение SDA
    RLC     A               ; задвигаем CY в ACC
    CLR     MCO             ; SCL = 0
    LCALL   D6MCS
    DJNZ    R6,I2CGETA1
    SETB    MDE
    RET
```

Кроме приведенных потребуются также подпрограммы для подтверждения приема байта данных мастером. При записи байта на линию данных интерфейса бит подтверждения выставляет приемник данных, а при чтении байта с линии данных выставлять бит является обязанностью мастера, причем при приеме последнего байта данных мастер вместо нулевого бита должен выставить единичный. Поэтому подпрограмм подтверждения приема две: одна выставляет нулевой бит подтверждения (I2CSETK0), вторая единичный (I2CSETK1).

Нулевой бит подтверждения реализуется фиксацией нулевого уровня на линии данных SDA и формированием синхроимпульса:

```
I2CSETK0: ; подтверждение мастера 0
    CLR     MDO             ; SDA = 0
    LCALL   D6MCS
    SETB    MCO             ; SCL = 1
    LCALL   D6MCS
    CLR     MCO             ; SCL = 0
    LCALL   D6MCS
    RET
```

Единичный бит подтверждения реализуется аналогично, только на линии данных фиксируется единичный уровень:

```
I2CSETK1: ; подтверждение мастера 1
    SETB    MDO             ; SDA = 1
    LCALL   D6MCS
    SETB    MCO             ; SCL = 1
    LCALL   D6MCS
    CLR     MCO             ; SCL = 0
    LCALL   D6MCS
    RET
```

С помощью приведенных подпрограмм достаточно просто можно прочитать, например, текущее значение числа сотых долей секунды. Сначала необходимо выполнить цикл инициализации, приведенный на рисунке 49. В этом цикле на линии интерфейса записывается адрес устройства и

режим записи, после чего формируется состояние «стоп». Цикл позволяет убедиться, что устройство с указанным адресом присутствует. Адрес микросхемы календаря равен 50h. Поскольку он записывается в старшие семь бит байта, его нужно умножить на два, получим A0h. Именно этот байт передается в цикле инициализации, поскольку признак записи равен нулю.

В программе используются следующие константы:

```
CALWR       EQU 0A0h     ; байт адреса устройства при записи
CALRD       EQU 0A1h     ; байт адреса устройства при чтении
I2CACK      EQU 0        ; временный хранитель бита acknowledge
```

Прежде, чем начать работу с линиями интерфейса, нужно установить их начальное состояние при помощи инструкции (см. таблицу 40):

```
MOV     I2CCON,#0C8h ; мастер-режим, SCL=0, SDA=1
```

Чтобы сформировать необходимую последовательность сигналов интерфейса, нужно вызвать подпрограммы в следующем порядке: сначала формируем состояние «старт», затем записываем адрес устройства при записи, затем проверяем прием подтверждения, затем формируем состояние «стоп»:

```
LCALL   D6MCS
LCALL   I2CSTART
MOV     A,#CALWR     ; запись в устройство
LCALL   I2CPUTA
JC      ERR
LCALL   I2CSTOP
```

В примере используется некая метка ERR, на которую программа переходит в случае, если бит подтверждения не получен. Далее программа может выполнить цикл чтения после записи, приведенный на рисунке 47:

```
LCALL   I2CSTART
MOV     A,#CALWR     ; запись в устройство
LCALL   I2CPUTA
JC      ERR
MOV     A,#1         ; адрес слова
LCALL   I2CPUTA
JC      ERR
LCALL   I2CSTART     ; рестарт
MOV     A,#CALRD     ; чтение устройства
LCALL   I2CPUTA
JC      ERR
LCALL   I2CGETA
LCALL   I2CSETK1
LCALL   I2CSTOP
```

В этом цикле читается первый регистр микросхемы календаря, в котором хранятся сотые доли секунды. Сначала в цикле записи мы устанавливаем адрес устройства, затем делаем рестарт (повторное состояние «старт») а затем формируем повторный цикл чтения, в котором считываем

один байт, выставляем единичный бит подтверждения и формируем состояние «стоп»:

Зуммер

Зуммер представляет из себя пьезоэлектрический генератор, управляемый напряжением. Чтобы получить звук, на зуммер нужно подавать напряжение звуковой частоты. Лучше всего (громче всего) зуммер звучит на частотах 1 КГц, 1,5 КГц и 3,5 КГц.

Звук генерируется переключением напряжения на зуммере: напряжение 0 сменяется полным напряжением и наоборот. Для программирования зуммера нужно рассчитать время половины периода. Если f — частота звука в КГц, то время половины периода в миллисекундах $\tau = 1/(2 \times f)$.

Подача напряжения на зуммер производится через регистр ENA, имеющий формат, приведенный в таблице 63.

Таблица 63. Регистр ENA (08 00 04h)

Разряд	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Доступ	—	—	W	W	W	W	W	W
Назначение			INT0	SND2	SND1	SND0	EN_HI	EN_LO

Для генерации звука в этом регистре используются биты 2—4, обозначенные SNDn. Для генерации звука одной частоты нужно каждые τ миллисекунд менять значение битов 2—4 регистра ENA с нулевых на единичные.

Следующий простой пример генерирует звук частотой 1 КГц, τ = 0,5 мс.

```
    MOV     TCON,#01h
    MOV     DPP,#8
    MOV     DPH,#0
    MOV     DPL,#4
    SETB    TR0
LOOP:
    LCALL   WAITT
    MOV     A,#10h
    MOVX    @DPTR,A
    LCALL   WAITT
    MOV     A,#0h
    MOVX    @DPTR,A
    SJMP    LOOP
```

Подпрограмма задержки WAITT использует таймер 0 в режиме 1:

```
WAITT:
    JNB     TF0,$
    CLR     TF0
    MOV     TL0,#0E8h
    MOV     TH0,#0FBh
```

```
RET
```

Константы, загружаемые в регистры TL0 и TH0, рассчитываются следующим образом. Время задержки τ делится на время машинного цикла 0,477 мкс: 500 мкс/0,477 мкс = 1048 машинных циклов задержки.

Вычитаем из полного числа циклов 16-ти разрядного таймера полученное число: 65536—1048=64488.

Переводим число 64488 в шестнадцатеричную форму: 0FBE8h. Младшую часть числа 0E8h загружаем в регистр TL0, старшую часть числа 0FBh загружаем в регистр TH0.

5. Организация работы

Цикл создания управляющей программы для микроконтроллера представляет из себя обычный цикл написания программ, с той лишь разницей, что отладка программы значительно затруднена, если нет соответствующего программно-аппаратного обеспечения.

Сначала в обычном текстовом редакторе (или в специализированном редакторе, если программа создается на языке высокого уровня, и существует специализированный редактор) пишется текст программы. Далее с помощью программы-компилятора текст компилируется с получением объектного файла в так называемом hex-формате Intel и файла листинга.

Полученный объектный файл для отладки загружается в симулятор микроконтроллера, если таковой существует, или непосредственно в микроконтроллер. В первом случае мы имеем возможность пошагово исполнять инструкции программы и проверять её правильность. Во втором случае пошаговая отладка возможна только с помощью специализированного аппаратно-программного комплекса (эмулятора-отладчика). При этом используется программное обеспечение самой микросхемы микроконтроллера, зашитое в её ПЗУ изготовителем.

Подготовка и компиляция программы на ассемблере

Текст программы

Текст программы на ассемблере обычно создается в простом текстовом редакторе, например, с помощью программы «Блокнот», или с помощью текстового редактора программы FAR Manager. При этом нужно следовать следующим правилам.

1. Текст программы начинается с включения файла определений символов данной микросхемы микроконтроллера. Для учебного стенда таким файлом является файл mod812, который должен быть расположен в том же каталоге, что и файл текста программы. Включение файла производится при помощи символа $ (доллар):

```
$mod812
```

Если пропустить эту строчку кода, то ни один из символов, используемых в описаниях, например, регистров SFR, не будет доступен, и придется использовать явные адреса в виде целых чисел. Например, чтобы указать регистр управления таймерами 0 и 1 TCON, вместо TCON в программе нужно будет указывать его адрес 88h.

2. Программный текст должен завершаться инструкцией END, после которой не допускается размещение никаких символов.

3. Поскольку управляющая программа является единственной программой в микроконтроллере, она не может завершаться неопределенным образом, так как, в отличие от программы, работающей в среде операци

онной системы, по завершении основного алгоритма программы нет возможности вернуться к исполнению какой-то еще программы.

Следовательно, если управляющая программа не представляет из себя замкнутый бесконечный цикл, то после выполнения алгоритма такой цикл, называемый здесь «мертвым», должен быть организован, например, так:

```
SJMP $   ; мертвый цикл
```

4. Поскольку таблица векторов прерываний для данной архитектуры располагается в начальной части памяти (начиная с адреса 3), программа, использующая прерывания, не может начинаться с нулевого адреса.

Вместо этого по нулевому адресу располагается инструкция перехода на начало программы (три первых байта памяти зарезервированы как раз для этой цели). Начало программы при этом размещается после таблицы векторов прерываний. В микроконвертере ADuC842 последний адрес прерывания равен 5Bh (91), для размещения инструкций обработчика прерываний отводится 8 байт, поэтому минимальный адрес начала программы составляет 63h (99), приемлемый адрес начала программы равен 80h.

Чтобы организовать правильное размещение кода программы, используется инструкция ORG АДРЕС, например:

```
    LJMP    START
    ; таблица векторов прерываний располагается здесь
    ORG     80h
START:
    ; код программы располагается здесь
DEAD: ; мертвый цикл
    SJMP DEAD
END
```

5. Объявление переменных (ячеек памяти) производится в начале программы (перед первой исполняемой инструкцией), если текст программы не содержит инструкций распределения памяти DSEG (определяющей сегмент данных) и CSEG (определяющей сегмент кода).

Для объявления одного байта используется инструкция DATA:

```
ИМЯ DATA АДРЕС
```

Для объявления одного бита используется инструкция BIT:

```
ИМЯ BIT АДРЕС
```

Контроль за правильным размещением переменных в памяти данных возлагается на программиста. Следует помнить, что свободная область байтов имеет начальный адрес 30h (48) и конечный адрес 7Fh (127). Таким образом, программисту доступно 80 байт памяти.

6. Если программа интенсивно использует стек (много подпрограмм и одни подпрограммы вызывают другие подпрограммы, а также если используется обработка прерываний), то следует позаботиться об определении области стека.

Удачным способом размещения стека является использование 128 ячеек памяти, размещаемых по тем же адресам, что и SFR (80h—0FFh). Поскольку регистры SFR доступны при помощи прямого метода адресации, а ячейки памяти по тем же адресам — при помощи косвенного метода адресации, эти два пространства не пересекаются. Чтобы определить область стека таким образом, при старте программы нужно задать начальное значение указателя стека, равное 7Fh. Поскольку стек растет вверх, первой ячейкой стека будет ячейка с адресом 80h.

Компиляция

Для компиляции готового текста программы используется командная строка. Следует либо открыть приложение «Командная строка», либо использовать какой-либо файловый менеджер, например FAR Manager.

Предполагается, что все необходимые файлы находятся в одном каталоге, который следует сделать текущим. Если, например, вы работаете на диске D в каталоге SDK11, то сделать этот каталог текущим из командной строки можно при помощи двух команд. Первая команда делает текущим диск D:

```
D:<Enter>
```

Вторая команда делает текущим каталог SDK11:

```
CD\SDK11<Enter>
```

Необходимыми файлами для компиляции программы являются файл текста программы с расширением .asm, файл определений символов микроконтроллера mod812, кросс-ассемблер asm51.exe. Длина имени файла программы не должна превышать 8 символов.

Для компиляции программы в командной строке нужно ввести команду:

```
ams51 имя_файла.asm<Enter>
```

Например, если вы назвали файл программы test.asm, то следует ввести команду:

```
ams51 test.asm<Enter>
```

Результатом успешной компиляции являются:

1. Сообщение кросс-ассемблера в командной строке, имеющее примерно следующий вид:

```
8051 Cross-Assembler, Version 1.2h
     (c) Copyright 1984, 1985, 1986, 1987, 1988, 1989, 1990
         by MetaLink Corporation
     First pass
     Second pass
     ASSEMBLY COMPLETE, 0 ERRORS FOUND
```

Слова **0 ERRORS FOUND** указывают на отсутствие ошибок компиляции.

2. Объектный файл программы в hex-формате (файл с наименованием имя_файла.hex). Это текстовый файл, в котором машинные инструкции записаны шестнадцатеричными цифрами в нескольких строк. Каждая строка начинается с заголовка, который описывает количество инструкций, адрес размещения в памяти и контрольную сумму.

3. Файл листинга (файл с наименованием имя_файла.lst), при помощи которого можно посмотреть, в какие машинные инструкции и по каким адресам кросс-ассемблер оттранслировал исходную программу.

Если программа содержит синтаксические ошибки, результатом трансляции будет сообщение кросс-ассемблера, указывающее на количество обнаруженных ошибок, например:

```
8051 Cross-Assembler, Version 1.2h
     (c) Copyright 1984, 1985, 1986, 1987, 1988, 1989, 1990
         by MetaLink Corporation
     First pass
     Second pass
     ASSEMBLY COMPLETE, 2 ERRORS FOUND
```

Кроме того, кросс-ассемблер формирует файл листинга, в котором места ошибок указаны стрелками. Например, если скомпилировать следующий программный текст:

```
$mod812
    MOV     DPL,#7
    MOV     DPH,#0
    MOV     DPP,#8
    MOV     A,#255
    MOV     @DPTR,A
    SJMP    $
END
```

кросс-ассемблер обнаружит два места ошибок (приведен листинг):

```
                    1    $mod812
0000 758207         2        MOV     DPL,#7
0003 758300         3        MOV     DPH,#0
0006 758408         4        MOV     DPP,#8
0009 74FF           5        MOV     A,#255
000B E8             6        MOV     @DPTR,A
****-----------------------------------^----^
****ERROR #20: Illegal operand
****ERROR #20: Illegal operand
000C 80FE           7        SJMP $
                    8    END
```

На самом деле в этой программе всего одна ошибка — вместо инструкции MOV в строке 6 должна быть инструкция MOVX.

Доставка программы в микроконтроллер

Процесс записи скомпилированной управляющей программы в программную память микроконтроллера называется *доставкой*.

Программа загружается в ПЗУ микроконвертера при помощи внутреннего программного обеспечения микроконвертера — загрузчика HEX02-03, расположенного по адресу 0E000h. Микроконвертер начинает свою работу с указанного адреса. Внутренний загрузчик HEX02-03 сначала производит инициализацию и проверку устройств учебного стенда, а затем в течение некоторого времени ожидает передачи по последовательному порту строк загружаемой программы в hex-формате. При этом загрузчик посылает через последовательный порт слово CHIPID, затем знак «=», затем идентификатор микросхемы A6, затем символы «.». Если в течение передачи десяти точек никакой передачи по последовательному порту не следует, загрузчик передает управление на начало программной памяти.

Микроконтроллер и компьютер должны быть соединены через COM порт компьютера и соответствующий порт учебного стенда при помощи прилагаемого к учебному стенду кабеля, микроконтроллер должен быть подключен к блоку сетевого питания, блок питания подключен к сетевой розетке (рисунок 61).

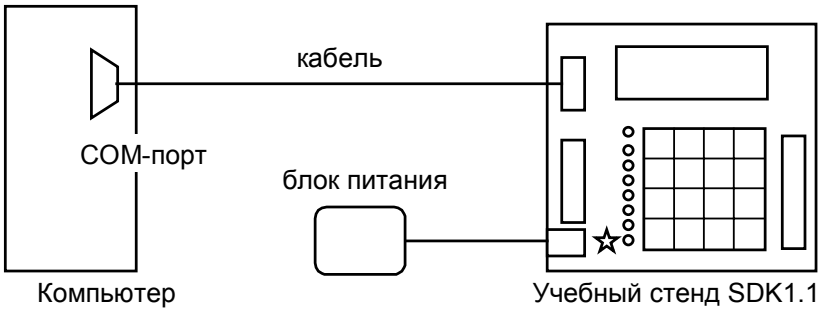


Рисунок 61. Подключение учебного стенда к компьютеру

Если по последовательному порту передается загружаемая программа, загрузчик стирает программную память (первые 56 Кбайт), построчно принимает программу, проверяет контрольную сумму каждой строки, и записывает инструкции по указанному в заголовке строки адресу.

После загрузки всех программных строк управление передается на начальный адрес загруженной программы, который указывается в последней строке hex-файла.

Для передачи управляющей программы по последовательному порту из компьютера в учебный стенд используется специальное программное обеспечение. Программа, разработанная автором пособия, и предназначенная для доставки hex-файла в учебный стенд, имеет имя SDK11LDR.exe.

После запуска программы SDK11LDR необходимо щелкнуть кнопку «Open» (рисунок 62) и при помощи появившегося диалога для поиска файлов найти и выбрать hex-файл.

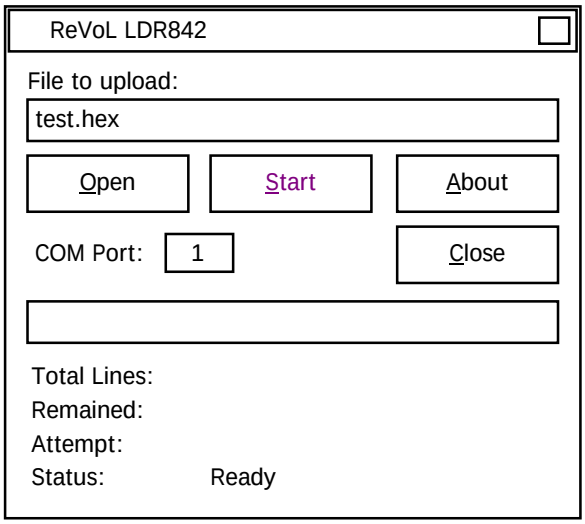


Рисунок 62. Окно программы SDK11LDR

После того, как файл будет выбран, программа SDK11LDR проверяет, указан ли в hex-файле стартовый адрес загружаемой программы, и если не указан, предлагает ввести его в диалоге «Start address» (рисунок 63).

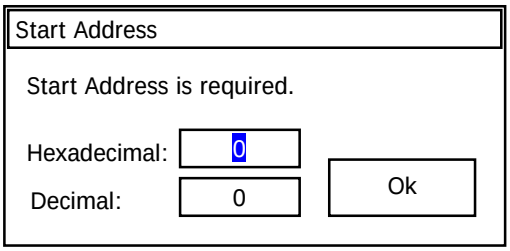


Рисунок 63. Диалог для задания стартового адреса

Адрес задается в шестнадцатеричном формате в поле «Hexadecimal» или в десятичном формате в поле «Decimal». В большинстве случаев можно принять адрес «0», предлагаемый по умолчанию.

Для начала передачи файла в учебный стенд нужно:

1. Нажать кнопку «Start» программы SDK11LDR;
2. Нажать кнопку сброса (reset) на учебном стенде.

Кнопка сброса в учебном стенде расположена внизу слева от линейки светодиодов (на рисунке 61 кнопка указана звездочкой).

Длительность процесса доставки зависит от размера передаваемой программы и может занимать несколько минут. Сразу после загрузки в окне программы SDK11LDR появится надпись Success, а микроконтроллер начнет выполнение загруженной программы. Чтобы начать выполнение загруженной программы заново со стартового адреса, нужно нажать кнопку сброса.

Следует иметь ввиду, что программа SDK11LDR работает только с учебным стендом SDK1.1 с микроконвертером ADuC842.

6. Симулятор SIMSDK11

Разработанный автором симулятор SIMSDK11 предназначен для отладки программ для учебного стенда SDK1.1 (рисунок 64).

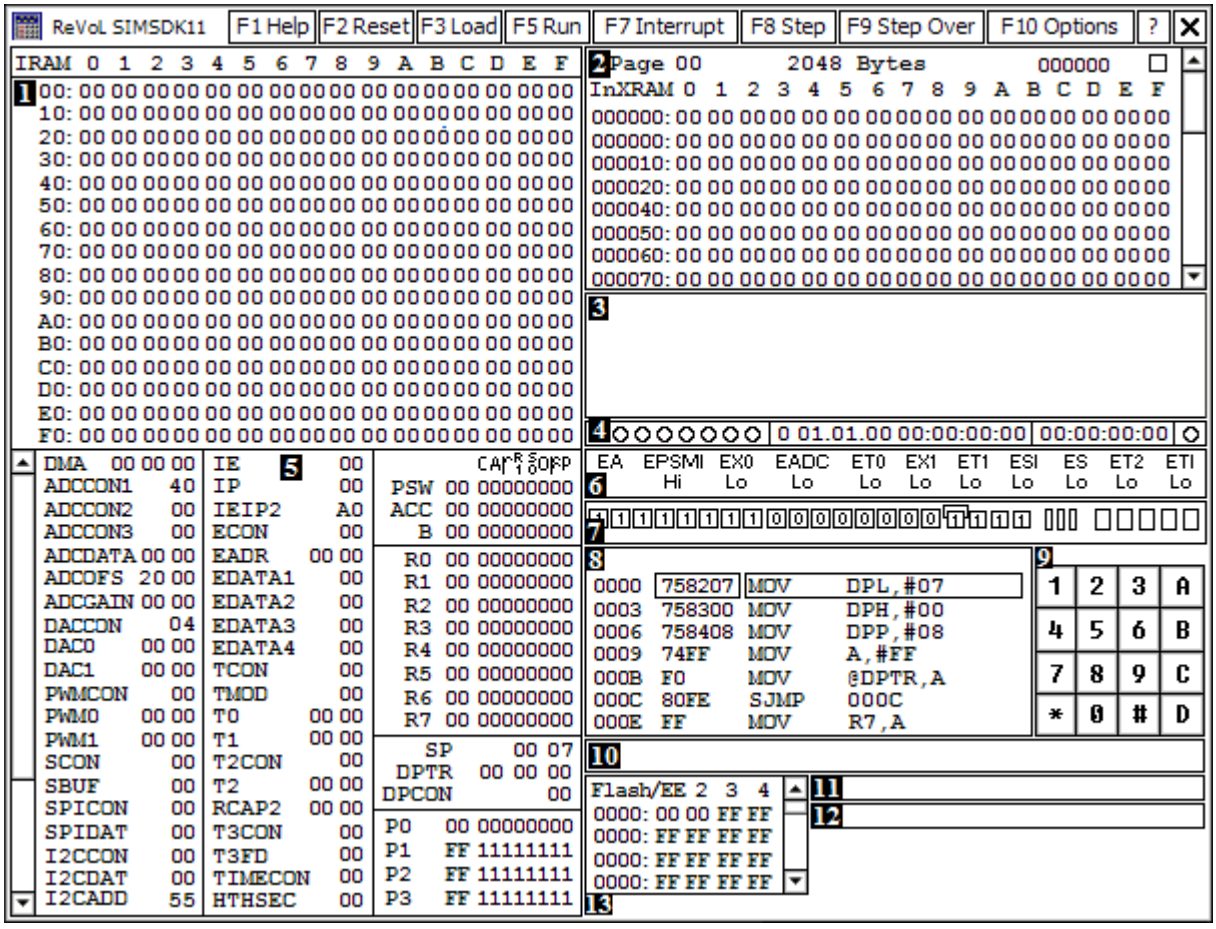


Рисунок 64. Окно симулятора SIMSDK11

Основное окно программы отображает все устройства учебного стенда, кроме дополнительной Flash памяти.

В левой верхней части отображается вся внутренняя память данных IRAM ❶, в правой верхней части отображается внутренняя внешняя память данных InXRAM и внешняя память данных XRAM ❷ (для переключения между InXRAM и XRAM нужно щелкнуть квадратик в правой верхней части области XRAM). Под окном внешней памяти расположен дисплей ❸, под дисплеем расположены линейка светодиодов, часы, отсчет времени работы симулятора и кнопка-индикатор режима работы в строке, обозначенной ❹.

В нижней левой части окна расположено поле регистров SFR ❺, поделенное на три колонки. В правой колонке отображаются самые важные регистры — флаги, аккумулятор A, вспомогательный аккумулятор B, регистры R0—R7, указатель стека SP, указатель данных DPTR, порты P0—P3.

В правой части под линейкой светодиодов расположено поле состояния регистров IE, IP, IEIP2 ❻, где буквами указаны биты регистров, слова

ми «Hi» и «Lo» — уровень приоритета. В этом же поле под словами «Hi» и «Lo» при установке флагов прерываний загораются цветные точки.

Ниже расположено поле 16-ти разрядного порта вывода, показывающее состояние его разрядов ❼. В правой части этого поля расположены индикаторы частот 1КГц, 100Гц, 10 Гц и 1Гц, а левее индикаторов частоты — три индикатора сигналов на зуммере.

Ниже поля порта вывода и индикаторов расположено поле дизассемблера ❽, отображающее семь текущих инструкций памяти программ. Текущая инструкция выделена. Правее поля дизассемблера расположена клавиатура ❾. Над рядами клавиш при обращении к регистру клавиатуры загораются цветные точки.

Ниже поля дизассемблера расположен индикатор диаграммы интерфейса I^2C ❿. Здесь при работе интерфейса рисуется состояние его линий.

Указателями 11 и 12 на рисунке 64 обозначены индикаторы состояния выхода и входа последовательного порта. Указателем 13 обозначено поле, показывающее состояние встроенной Flash памяти данных.

При использовании данной программы следует помнить, что она реализует ограниченное количество функций реального микроконтроллера, обусловленное необходимостью выполнить практические работы.

Внешний вид программы может незначительно отличаться от приведенного на рисунке 64 в связи с ее совершенствованием.

Загрузка программы

Чтобы начать отладку программы, её нужно загрузить в симулятор. Есть два способа сделать это. Первый — соединить порты компьютера COM1 и COM2 нуль-модемным кабелем и использовать ту же процедуру доставки программ в микроконтроллер, что и для учебного стенда.

Второй способ — нажать клавишу F3 и при помощи диалога для поиска файлов найти и выбрать hex-файл отлаживаемой программы.

При этом память симулятора очищается, программа размещается в памяти программ, и программный счетчик PC устанавливается на начало программы, если оно указано в hex-файле, или на начало памяти.

В окне дизассемблера отображается часть загруженной программы на ассемблере, начиная с начального адреса. Инструкции в дизассемблере не обязательно будут иметь вид, который вы им придали в тексте программы.

Исполнение программы

Для исполнения программы целиком или отдельных ее частей есть несколько клавиш. При помощи клавиши F5 программа запускается на исполнение в непрерывном режиме. Эту клавишу следует использовать, когда есть уверенность, что программа не содержит ошибок и правильный результат ее работы является ожидаемым.

Чаще придется использовать клавиши F8 и F9. Клавиша F8 заставляет симулятор выполнить одну (текущую) инструкцию. При этом можно сразу увидеть и проанализировать результат ее исполнения. Этому способствует также то обстоятельство, что при записи в ячейки памяти или в регистры SFR они отмечаются в симуляторе красным цветом.

Клавиша F9 заставляет симулятор выполнить все инструкции для того, чтобы перейти к следующей инструкции в окне дизассемблера. Это позволяет выполнять вызовы подпрограмм, не заходя в них, а также полностью выполнять циклы, не отслеживая выполнение каждой итерации. В случае, если текущая инструкция не является вызовом подпрограммы или инструкцией цикла, клавиша F9 действует как клавиша F8.

При помощи клавиши F7 программа запускается на непрерывное исполнение до тех пор, пока не возникнет прерывание. Этот режим используется для отладки прерываний.

Кроме того, при помощи сочетания клавиш «Alt+*n*», где *n* — цифра от 0 до 7, можно заставить симулятор работать в непрерывном режиме до тех пор, пока содержимое регистра R*n* не станет равным нулю.

Режимы отладки

Микроконвертер начинает свою работу с адреса 0E000h, по которому должно располагаться базовое программное обеспечение. При отладке программ в симуляторе исполнение этой части кода лучше отключить. Для этого нужно нажать клавишу F10 (вызвать диалог «Options») и включить флажок «Quick Start».

Скорость непрерывного исполнения программы можно изменить при помощи флажка «Slow Mode» (медленный режим), а также выбрав время обновления состояния памяти в списке «Update interval». При выборе режима обновления «Continuous» (непрерывно) становится доступным флажок «Show Disassembly», который управляет отображением исполняемых команд в окне дизассемблирования.

В диалоге «Options» можно также установить адрес остановки программы в окне «Break Address».

Отладка программ для клавиатуры

При отладке программ, отслеживающих нажатия клавиш, в режиме пошаговой отладки нужно включить режим залипания клавиш клавиатуры. Для этого нужно вызвать диалог «Options» (клавиша F10) и выключить флажок «Button Keypad». В этом режиме при нажатии клавиши на клавиатуре симулятора она останется нажатой до повторного нажатия.

Кроме того, при обращении к столбцу клавиатуры над ним на непродолжительное время загорается маленькая красная точка, а при чтении ряда, в котором расположена нажатая клавиша, там же загорается зеленая точка.

Отладка программ для I²C

При отладке программ, использующих интерфейс I²C, в диалоге «Options» должен быть включен флажок «Diagram On/Off» в рамке «I2C». При этом каждое изменение состояния линий интерфейса вызывает отрисовку диаграммы в окне ❿ (рисунок 64).

Диаграмма содержит три линии (рисунок 65). Верхняя линия показывает состояние линии данных SDA. Нижняя линия соответствует линии синхронизации SCL. Между ними рисуется вспомогательная цветная линия, обозначающая начало сеанса, конец сеанса и рестарт.

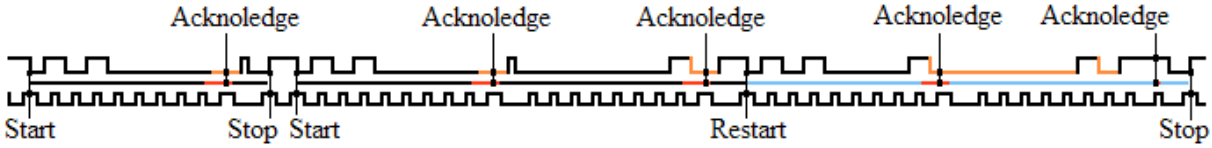


Рисунок 65. Диаграмма I²C цикла запись—чтение

Начало сеанса отмечается началом средней линии оранжевого цвета. Если во время сеанса произошел рестарт, средняя линия диаграммы меняет цвет на синий. Как только на линиях интерфейса окажется состояние «Стоп», средняя линия заканчивается. Таким образом, при помощи средней линии вы можете отслеживать правильность формирования сеансов.

На верхней линии (линии данных), которая имеет зеленый цвет, появляются участки с оранжевым оттенком. Они соответствуют понижению линии принимающим устройством при передаче сигнала acknoledge.

Розовые участки на верхней линии соответствуют понижению линии данных ведомым устройством для передачи нулевых сигналов.

На средней линии красным цветом отмечаются участки, соответствующие ожиданию сигнала подтверждения acknoledge.

Чтобы изменить вид диаграммы (растянуть ее), можно открыть диалог «Options» при помощи клавиши F10 и изменять значения полей «Stroke length» (длина штриха) и «Shift cycles» (количество сдвигов на изменение состояния).

Кроме того, растянуть тот или иной участок диаграммы можно программно, просто дублируя несколько раз состояние той или иной линии интерфейса, поскольку диаграмма дорисовывается каждый раз при выполнении команд, изменяющих линии интерфейса. Например, изменив состояние линии синхронизации при помощи команды CLR MCO, повторите команду еще раз, и диаграмма расширится в этом месте.

Остановка непрерывного исполнения

Чтобы остановить непрерывное исполнение программы, нужно щелкнуть кнопку «Закрыть» (крестик), либо нажать любую из клавиш управления программой, либо любую кнопку в окне программы, либо кнопку старт/стоп (в правой части линейки, обозначенной ❹ на рисунке 64).

Примерные темы практических заданий

Задания предпочтительно должны выполняться в указанном порядке. Программы должны накапливать подпрограммы, разработанные в ходе выполнения предыдущих заданий. Звездочкой «*» отмечены задания повышенной сложности.

1. Зажечь все светодиоды линейки.

2. Зажечь светодиоды в линейке через один.

3. Мигание линейки светодиодов: 0,25 с все светодиоды горят, затем 0,25 с все светодиоды погашены (использовать подпрограмму задержки Delay250 с помощью вложенных циклов DJNZ).

4. Бегущий огонь на линейке светодиодов. Один из светодиодов горит 0,125 с, после чего он гаснет и зажигается следующий; по достижении конца линейки процесс повторяется с первого светодиода.

5. Формирование бегущей волны на линейке светодиодов. Светодиоды последовательно зажигаются по одному каждые 0,125 с, начиная с первого. После того, как все светодиоды зажгутся, они так же последовательно гасятся, начиная с первого.

6. Как только будет обнаружено нажатие клавиши «1», зажечь все светодиоды линейки. Дребезг не отслеживается.

7. Отслеживание нажатия и отпускания клавиши. При нажатии клавиши «1» зажигается вся линейка светодиодов, и горит до тех пор, пока клавиша «1» не будет отпущена. Отслеживать дребезг с помощью таймера.

8*. Отслеживание нажатия всех клавиш. При нажатии клавиш «1—8» зажигается соответствующий светодиод. При нажатии на любую другую клавишу все светодиоды гасятся. Дребезг отслеживать.

9*. Отслеживание нажатия и отпускания всех клавиш. При нажатии клавиш «1—8» зажигается соответствующий светодиод, который гасится при отпускании клавиши. Дребезг отслеживать.

10**. Драйвер клавиатуры. Драйвер непрерывно сканирует клавиатуру, используя прерывания таймера и отслеживание дребезга. При определении нажатия клавиши драйвер записывает в буфер клавиатуры скэн-код клавиши (её порядковый номер от единицы) при помощи подпрограммы PutKey. При определении отпускания клавиши драйвер записывает в буфер клавиатуры её скэн-код с установленным старшим битом (при помощи подпрограммы PutKey). Основная программа извлекает скэн-коды клавиш из буфера клавиатуры при помощи подпрограммы GetKey и выполняет действия, описанные в задании 9. Наличие кода 0 в буфере клавиатуры — признак его пустоты.

11. Формирование на зуммере звукового сигнала частотой 1,5 КГц (сигнал на зуммере изменяется каждые 333 мкс).

12. Формирование на зуммере звукового сигнала «Бип—Бип» (0,25—0,5 с издается звуковой сигнал, 0,25—0,5 с зуммер молчит).

13*. То же, что и 12, но с использованием прерываний по таймерам 0 и 1. Прерывание по таймеру 0 вызывает смену напряжения на зуммере, заставляя его издавать сигнал. Прерывание по таймеру 1 включает и выключает звучание зуммера.

14. Вывод на дисплей символа из аккумулятора (процедура DispAC).

15. Вывод на дисплей символа, соответствующего нажатой клавише. Используется драйвер клавиатуры и таблица перекодировки скэн-кода в код символа, а также процедура DispAC.

16. Вывод на дисплей произвольного сообщения на русском языке.

17. Прокрутка сообщения на дисплее.

18. Перемещение курсора в строке дисплея (по кругу).

19. Перемещение мигающего курсора в строке дисплея (по кругу).

20*. Перемещения курсора в строке дисплея клавишами «4» (влево) и «6» (вправо). Клавиша «5» устанавливает курсор в начало дисплея.

21. Вывод на дисплей шестнадцатеричного значения байта аккумулятора (подпрограмма вывода значения аккумулятора на дисплей DispAH).

22. Вывод на дисплей двоично-кодированного десятичного числа, содержащего две цифры в одном байте.

23. Запись и чтение из памяти данных Flash/EEPROM, расположенной на кристалле микроконвертера. Записывается содержимое аккумулятора в строку 2, столбец 3.

24. Чтение регистра сотых долей секунды календаря и вывод считанного значения на дисплей (регистр сотых долей содержит две двоично-кодированные десятичные цифры).

25*. Чтение из календаря и вывод на дисплей часов, минут и секунд в формате «ЧЧ:ММ:СС».

26*. Запись в календарь времени 12 часов, 33 минуты, 11 секунд, затем чтение и вывод времени на дисплей, как в предыдущем задании.

27. Разработка подпрограммы BCDAdd, складывающей два двоично-кодированных десятичных числа (две цифры в одном байте), и выводящей результат на дисплей. Числа задаются непосредственно кодом программы.

28. Разработка подпрограммы BCDSub, вычитающей два двоично-кодированных десятичных числа (две цифры в одном байте), и выводящей результат на дисплей. Числа задаются непосредственно кодом программы.

29*. Ввод одно или двухзначного целого числа с клавиатуры и запись значения в память по нажатию клавиши «#».

30*. То же, что и 28, но клавиша «*» стирает неверно введенный знак.

31*. Разработка калькулятора, который принимает одно число, затем знак операции, затем второе число, выполняет операцию и результат выводит на дисплей. Ввод первого числа завершается нажатием знака операции. Ввод второго числа завершается нажатием клавиши «#». Знак операции вводится клавишами «A» (обозначающей сложение) и «B» (обозначающей вычитание). Число состоит из одной или двух десятичных цифр.

Литература

1. 8051 Cross Assembler. User's manual. Электронный документ «ASM51.pdf».

2. HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver). Электронный документ «hd44780.pdf».

3. MicroConverter 12-Bit ADCs and DACs with Embedded High Speed 62-kB Flash MCU. Электронный документ «ADuC841_2_3_0.pdf».

4. PCF8583 Clock/calendar with 240 x 8-bit RAM. Электронный документ «pcf8583.pdf».

5. Piezo Sound Generator HPA17A. Электронный документ «HPA17A-R9.pdf».

6. The I2C-bus and how to use it (including specifications). Электронный документ «I2C.pdf».

7. Отличия в программировании SDK-1.1 с ADuC842, ADuC831 и ADuC812. Электронный документ «SDK11_APPNOTE1.pdf».

8. Принципиальная схема учебного стенда SDK1.1. Электронный документ «sdk1.1r4sch.pdf».

9. Учебный стенд SDK-1.1. Руководство пользователя. Электронный документ «sdk11_userm_v1_0_8.pdf».

10. Электронный документ «842qref0.pdf» — экспресс справка по микроконвертеру ADuC842.

Владимир Вадимович Пономарев
Организация и программирование
учебного стенда SDK1.1

Отпечатано с готового оригинал-макета

Издательство ОТИ МИФИ
Тираж 50 экз.