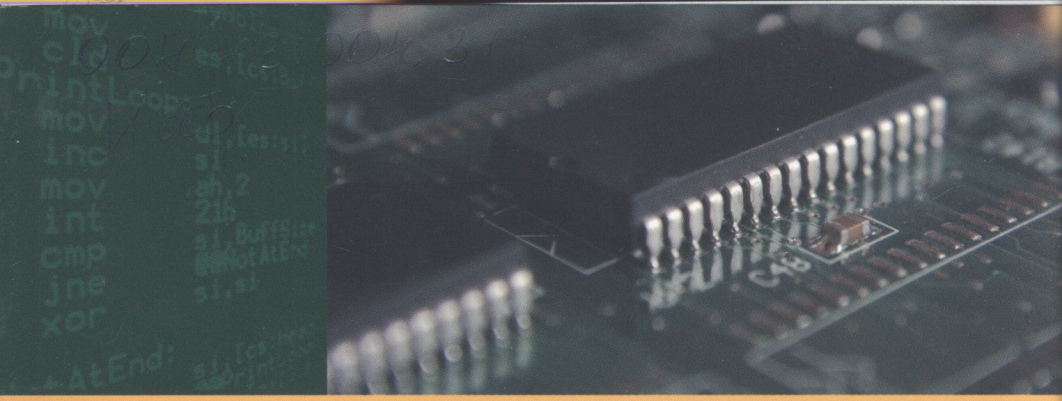


Иванов В. Б.

Программирование микроконтроллеров для начинающих

Визуальное проектирование, язык С, ассемблер

WWW.MK-PRESS.COM

В. Б. Иванов

Программирование микроконтроллеров для начинающих

ВИЗУАЛЬНОЕ ПРОЕКТИРОВАНИЕ, ЯЗЫК С, АССЕМБЛЕР

Киев, "МК-Пресс"
СПб, "КОРОНА-ВЕК"
2016

Иванов В. Б.

И20 Программирование микроконтроллеров для начинающих. Визуальное проектирование, язык С, ассемблер. – К.: "МК-Пресс", СПб.: "КОРОНА-ВЕК", 2016. – 176с., ил.

ISBN 978-5-7931-0559-0 ("КОРОНА-ВЕК")
ISBN 978-966-8806-65-0 ("МК-Пресс")

В этой книге автор проводит читателя по пути освоения программирования микроконтроллеров от простого к сложному. Начав с краткого описания архитектуры и системы команд микроконтроллеров PIC, он переходит к визуальному проектированию в среде Flowcode, которое позволяет получить выполняемый код без каких-либо навыков программирования на ассемблере или языке высокого уровня. В завершающих двух главах представлены примеры программирования микроконтроллеров PIC с помощью языка С в среде MikroC и ассемблера в среде MPLAB.

Для освоения материала книги не требуется каких-либо специальных знаний, что упрощает процесс изучения программирование микроконтроллеров для тех, кто начинает работать в этом направлении "с нуля".

ББК 32.973-04

Всеволод Борисович Иванов

Программирование микроконтроллеров для начинающих

ВИЗУАЛЬНОЕ ПРОЕКТИРОВАНИЕ, ЯЗЫК С, АССЕМБЛЕР

Главный редактор: Ю. А. Шпак

Подписано в печать 21.03.2016. Формат 60 × 84 1/16. Бумага офсетная. Печать офсетная. Усл. печ. л. 10,2. Уч.-изд. л. 9,7. Тираж 1000 экз. Заказ №157.

СПД Савченко Л.А., Украина, г. Киев, тел./факс: (044) 517-73-77; e-mail: info@mk-press.com. Свидетельство о внесении субъекта издательского дела в Государственный реестр издателей, производителей и распространителей издательской продукции: серия ДК №51582 от 28.11.2003г.

Віддруковано з готових діапозитивів ФОП Гордукова І. Є.
(згідно виписки з ЄДРПОУ від 10.06.2015 р.)
м. Кам'янець-Подільський, вул. Привокзальна, 20
тел. 0 38 494 22 50, drukruta@ukr.net

ISBN 978-5-7931-0559-0 ("КОРОНА-ВЕК")
ISBN 978-966-8806-65-0 ("МК-Пресс")

Содержание

Введение

Современный мир совершенно невозможно представить без микроконтроллеров (МК), и здесь нет никакого преувеличения. Они используются в мобильных телефонах и стиральных машинах, в елочных гирляндах и охранной сигнализации, в детских игрушках и банкоматах... И мы перечислили только сферы бытового применения! Не менее широко эти маленькие помощники человека распространены в науке и технике, в армии и на транспорте. Кроме того, стремительное развитие микроэлектронных технологий делает себестоимость подобных приборов все более низкой, их размеры — все меньшими, а возможности — все большими.

Хотя размеры МК порой не превышают спичечной головки, они представляют собой весьма сложные устройства: по сути, миниатюрные компьютеры. "Сердцем" МК является микропроцессор, выполняющий обработку и преобразование цифровой информации. По этой причине, как и для "настоящего" компьютера, для работы МК требуется программирование. Независимо от размеров устройства, программирование может оказаться весьма трудоемким процессом — особенно, если к создаваемым программам предъявляются высокие требования в отношении экономного расходования ресурсов памяти и быстродействия. Для МК экономия ресурсов зачастую является первоочередной задачей, поскольку в сравнении с тем же персональным компьютером они у МК довольно ограничены.

Автор попытается провести читателя по пути освоения программирования МК от простого к сложному, и путь этот довольно сильно отличается от традиционного подхода, применяемого в большинстве монографий и учебных пособий, посвященных микроконтроллерам. Дело в том, что многие авторы сами продвигались от "железа" к программам. Зачастую они начинали с использования МК в радиолюбительских или бытовых конструкциях и сразу же переходили к "высшему пилотажу" программирования: ассемблеру, на освоение которого уходит немало сил и времени. На наш взгляд, существуют системы программирования МК, гораздо более подходящие для первого знакомства. С них мы и начнем. Для многих задач, которые поручается выполнять МК, этих систем вполне достаточно. Хотя, безусловно, ассемблер остается наиболее мощным инструментом программирования, и обойден вниманием также не будет.

Данная книга рассчитана в первую очередь на начинающих, которые, впрочем, также могут быть разного уровня подготовки. Ознакомиться с основами проектирования с применением МК можно, даже не

владея никакими языками программирования. Для этого служат средства так называемого визуального проектирования. Тот, у кого есть хотя бы начальное представление о языках программирования высокого уровня, может освоить программирование для МК на языке С, а затем с помощью данной книги испробовать свои силы и в разработке программ на ассемблере.

Прочитавшие эту книгу не станут профессиональными программистами микроконтроллеров. Автор и не ставил перед собой такую задачу, тем боле, что и себя не считает профессионалом в этой сфере. В соответствии с названием книги, здесь предлагается сделать лишь несколько шагов в данном направлении, однако они, будем надеяться, составят хорошую базу для дальнейшего самостоятельного продвижения к профессионализму. В то же время, приобретенных знаний и навыков хватит, чтобы программировать микроконтроллерные системы достаточно высокой сложности и для самых разнообразных приложений. Другой вопрос — насколько рационально такие системы будут сделаны? Но, это уже — дело опыта и практики.

Разработка микроконтроллерных систем состоит из четырех этапов: проектирования, программирования, макетирования и реализации готового изделия. Данная книга посвящена почти исключительно второму этапу. Впрочем, использование описываемой в книге платы разработки можно рассматривать как некоторую форму макетирования. Проектирование — это просто грамотная, продуманная постановка задачи и подбор возможных и доступных средств ее решения. Учить здесь, собственно говоря, нечему. Что касается реализации, то это — чисто технологическая сторона разработки системы, и потому она здесь не рассматривается.

Теперь о содержании книги... Глава 1 посвящена описанию главных действующих персонажей книги: микроконтроллеров и подключаемых к ним внешних устройств. В настоящее время существует несколько семейств МК, широко используемых на практике, разработанных крупными компаниями-производителями. В данной книге речь идет об устройствах компании Microchip под общим названием PIC. Дается описание структуры и состава микроконтроллеров этого класса. Хотя данное описание далеко не полное, оно дано в таком объеме, который необходим для понимания материала, излагаемого в последующих главах о программировании. То же самое относится и к представленному в главе 1 описанию команд процессора МК. Здесь приведены только самые необходимые справочные данные по командам, а их использование поясняется при разработке конкретных программ в последней главе.

Для успешного изучения материала необходима практика реализации разрабатываемых проектов на каком-то инструментарии (в "железе"). Одного микроконтроллера самого по себе, конечно же, недостаточно. В качестве такого инструментария автор предлагает использовать отладочную плату от компании Mikroelektronika (Сербия) под названием EasyPIC5. В последнем разделе главы 1 дается краткое описание этой платы в базовом комплекте и с некоторыми дополнительными модулями: алфавитно-цифровым жидкокристаллическим дисплеем, графическим ЖК-дисплеем, сенсорным экраном, термодатчиком, PS/2-клавиатурой, адаптером карты памяти Compact Flash. Программирование работы МК со всеми этими аксессуарами рассматривается в следующих главах.

В качестве первого шага к программированию МК в главе 2 представлена система визуального проектирования Flowcode. Мы называем ее средством проектирования, а не программирования, по той причине, что для разработки программ в Flowcode практически не требуется знать языков программирования. Программа собирается как бы из "кубиков" путем размещения на специальной диаграмме готовых компонентов с помощью мыши. В собранных таким образом компонентах пишутся весьма простые и понятные коды. В результате, программа собирается быстро и с минимальными трудозатратами. Готовый проект может быть испытан средствами симуляции прямо в пакете Flowcode. В симуляцию можно включить компьютерные модели различных внешних устройств (например, дисплея или подключенных к МК светодиодов), однако конечным продуктом пакета Flowcode может быть и вполне работоспособная реальная программа для МК: так называемая "прошивка". При наличии средств записи исполняемой программы в память МК (на плате EasyPIC5 присутствуют) мы создаем работающий проект. Конечно, простота проектирования имеет свои негативные стороны. Конечный продукт выполнен не очень рационально с точки зрения расходования ресурсов микроконтроллера, и в первую очередь — памяти. Не лучшим образом он себя проявляет и в плане быстродействия. Впрочем, рациональность и оперативность нужны далеко не всегда. И кроме того, это ведь — только первый шаг!

Второй шаг (глава 3) — программирование МК на языке высокого уровня С. Это уже — вполне профессиональный подход к разработке микропроцессорных систем, поскольку с помощью языка С создают реальные коммерческие продукты и изделия. Конечно, здесь также имеет место проигрыш по производительности и потреблению ресурсов по сравнению с низкоуровневым программированием, однако по сравнению с использованием Flowcode он уже существенно меньше. Если

к системе не предъявляются очень высокие требований по производительности и ресурсам, то использование С вполне оправдано.

К настоящему существует несколько компиляторов для разработки С-программ для МК. Мы в данной книге предлагаем работать с системой MikroC. Во-первых, это — разработка той же компании Mikroelektronika, что, само собой, дает максимальную совместимость программного и аппаратного обеспечения. Во-вторых, система представляет собой не просто компилятор, а полностью интегрированную среду подготовки проектов, включая средства отладки и прошивки программы в МК. Наконец, немаловажный фактор — наличие для MikroC хорошо подготовленной системы подсказок и справки на русском языке.

Сам язык С мы в этой книге изучать не будем. Возможно, глубоких знаний в данной области читателю на первых порах и не понадобится. На достаточно большом ряде примеров, представленных в главе 3, мы разберемся с большинством возможностей, предоставляемых С-программированием для МК.

Последняя глава книги (третий шаг) посвящена программированию на ассемблере. Сразу отметим, что на глубину изложения она не претендует. Будут рассмотрены только основы. В качестве инструмента создания проектов мы воспользуемся наиболее популярной среды разработки для микроконтроллеров PIC: системой MPLAB. Опять-таки, мы не ставим цель освоить все возможности этого пакета, однако основные навыки работы в нем читатель получит. Как и в предыдущих разделах, процесс обучения построен на ряде примеров, освоив которые можно продвигаться в программировании на ассемблере самостоятельно.

К книге прилагается компакт-диск с используемым в книге программным обеспечением (в ряде случаев — демо-версии) и исходные тексты и проектные файлы всех рассматриваемых примеров.

И последнее... На кого ориентирована эта книга? На самые широкие круги читателей от школьников старших классов до вполне сформировавшихся специалистов, у которых возникла необходимость "с нуля" освоить программирование МК. Никаких специальных знаний для освоения материала читателю не понадобится. Действительно, все здесь можно начинать "с нуля", даже не имея по началу под рукой самого микроконтроллера и его окружения. Единственное пожелание читателям: двигайтесь по книге с самого начала и до конца, ничего не пропуская.

Глава 1

Микроконтроллер и его окружение

Микроконтроллер (МК) является составной частью микропроцессорной системы. Его функции заключаются в получении информации извне, ее обработке и выдаче информации во внешний мир. Под такое определение подпадают, конечно, все микропроцессорные устройства, в том числе, например, и персональный компьютер (ПК). Разница между ними заключается только в смещении акцентов. В персональном компьютере ключевую роль играют средства ввода-вывода и отображения информации: монитор, клавиатура, мышь, звуковая карта и т.д. В микроконтроллере же основная часть работы выпадает на логическую и арифметическую обработку данных. Для ПК наиболее важна организация взаимодействия с человеком, в то время как МК занимается, прежде всего, взаимодействием с внешними устройствами.

Микроконтроллер всегда работает в некой "среде обитания", состоящей из внешних устройств. Сам МК и его окружение составляют аппаратную часть микропроцессорной системы (*hardware*), а комплекс программных средств — системный сегмент (*software*). В данной главе мы познакомимся с наиболее важными объектами аппаратного обеспечения МК и его окружения.

Состав и структура микроконтроллера

Состав МК принято разделять на две части: процессорное ядро и изменяемый функциональный блок. Первая часть обязательна для всех микроконтроллеров и включает в себя центральный процессор, схему управления и схему синхронизации. Вторая часть содержит набор модулей, состав которого зависит от типа МК.

В отличие от фон-неймановской архитектуры персонального компьютера, современные микроконтроллеры построены по так называемой гарвардской архитектуре (рис. 1.1).

Центральный процессор, содержащий устройство управления (УУ) и арифметико-логическое устройство (АЛУ), выполняет команды, извлекаемые из памяти программ. Команды производят операции над операндами (данными), выбираемыми либо также из памяти программ, либо из памяти данных. В отличие от ПК, где программы хранятся на маг-

нитном диске и для исполнения загружаются в оперативную память (ОЗУ), в МК используется одна единственная программа, записанная в постоянную (возможно, перепрограммируемую) память (ПЗУ). Еще один вид памяти: стек — используется при работе с подпрограммами.

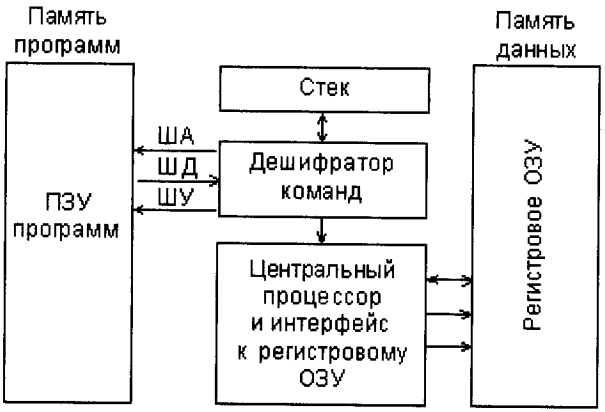


Рис. 1.1. Гарвардская архитектура

Выборка очередной команды из ПЗУ осуществляется по адресу, который передается по шине адреса (ША), возможно, при участии шины управления (ШУ). Данные передаются по шине данных (ШД). Разрядность шины данных как раз и определяет разрядность микроконтроллера. В данной книге мы говорим исключительно о восьмиразрядных устройствах. Это означает, что по ШД одновременно (параллельно) передается восемь двоичных разрядов, т.е. один байт. Разрядность шины адреса определяет максимально возможный объем памяти программ. В большинстве МК рассматриваемого нами класса разрядность ША составляет 14.

В настоящее время существует несколько вариантов реализации физической памяти микроконтроллера. Память программ всегда энергонезависима, т.е. ее содержимое сохраняется при выключении питания. В микроконтроллерах в составе конечных устройств, не подлежащих модификации, для программ используется ПЗУ масочного типа. Память программируется один раз при ее изготовлении. В универсальных МК предусмотрена возможность многократного перепрограммирования ПЗУ. В устаревших технологиях для этого использовались средства ультрафиолетового стирания и записи данных. Сегодня в большинстве случаев применяются технологии чисто электрического стирания и записи. Две основные разновидности такой памяти: EEPROM и Flash-

ROM. Часто для памяти программ используется Flash-ROM, а EEPROM применяется для энергонезависимого хранения констант. Необходимо отметить, что количество перезаписей памяти Flash на порядки выше числа перезаписей памяти EEPROM.

Поскольку программы и часть данных в МК хранится в ПЗУ, потребность в оперативной памяти существенно меньше в сравнении, например, с персональным компьютером. ОЗУ в микроконтроллерах организовано в виде так называемых регистров. В данном случае регистр представляет собой восьмиразрядный блок памяти. При этом в программах возможно обращение как ко всему байту данных в регистре, так и к его отдельным битам. Пространство регистровой памяти разделено на две части: регистры специальных функций (SFR) и регистры общего назначения (GPR). Подробнее об этом речь пойдет чуть позже.

До недавнего времени работа с большими объемами данных, не помещающимися в память МК, была нетипичным случаем, однако потребность в этом возникает все чаще. Выходом из положения является подключение к МК через стандартные средства ввода/вывода (так называемые порты) адаптеров для носителей больших объемов данных. В частности, к МК можно подключать карты памяти типа Compact Flash или мультимедийные карты MMC. Таким образом, реализуется возможность работы с массивами данных объемом в гигабайты и более.

Вся работа микропроцессора через схему синхронизации управляется тактовыми импульсами, поступающими или от внешнего источника или от собственного, встроенного в МК генератора. Тактовая частота генератора определяет производительность системы. Большинство команд выполняется за один машинный цикл, состоящий из четырех тактов генератора. Небольшое число команд выполняется за два цикла. В свою очередь, тактовая частота задается осциллятором (резонатором). Осциллятор может быть как внутренним (редко), так и внешним. В последнем варианте он подключается к специальным входам микроконтроллера (рис. 1.2).

Осциллятор может быть кварцевым или керамическим. Подключение для такого случая показано на рис. 1.2*а*. Кварцевый осциллятор наиболее стабилен. Флуктуации частоты в нем обычно не превышают сотых долей процента. Для керамического резонатора стабильность частоты составляет порядка десятых долей процента. В сравнении с кварцем он более дешев и механически более стоек. Если к временным интервалам работы системы не предъявляются высокие требования, то в качестве резонатора используется RC-цепочка со схемой подключения, показанной на рис. 1.2*б*.

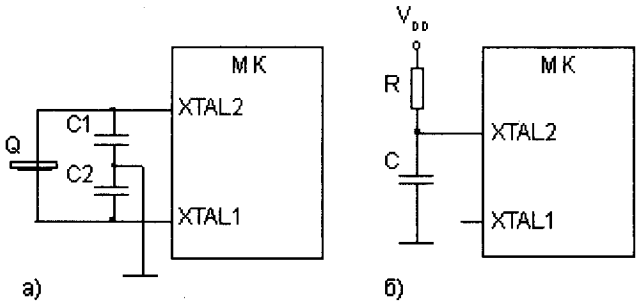


Рис. 1.2. Подключение внешнего осциллятора

Практически все микроконтроллеры имеют в своем составе один или несколько модулей таймеров. Таймер представляет собой устройство, работающее параллельно выполняемой программе (независимо от нее). По сути, таймер — это буфер памяти, содержимое которого инкрементируется (увеличивается на единицу) на каждом машинном цикле. Когда инкрементируемое значение превышает максимально возможную для буфера величину, возникает переполнение таймера. В этот момент программа может выполнить некоторые действия по обработке произошедшего события.

После переполнения содержимое буфера обнуляется, и процесс инкрементирования повторяется. Обычно в микроконтроллерах предусмотрено переключение таймера в так называемый режим счетчика. При этом инкрементирование происходит не по сигналам тактового генератора, а по сигналам на специальном входе. Тогда таймер просто подсчитывает число таких сигналов. В настройках МК можно предусмотреть деление частоты тактового генератора, чтобы таймер инкрементировался, например, не каждым циклом, а через один, два и т.д. вплоть до 256 циклов. При этом работает так называемый предделитель частоты на входе таймера.

В состав МК обычно входит модуль таймера специального назначения: сторожевого таймера (WDT — Watch Dog Timer). Его функция заключается в разрешении ситуации с аварийным зацикливанием программы. Работает сторожевой таймер так же, как и обычный, однако реакцией на переполнение является полный перезапуск МК (если, конечно, это разрешено). Для того чтобы предотвратить переполнение сторожевого таймера, используют команду сброса буфера WDT, размещенную в правильной точке программы. В результате, если программа работает штатно, не зацикливаясь, то переполнение не возникает. Разумеется, сторожевой таймер может быть включен или выключен с помо-

щью соответствующих настроек. Для него также предусмотрена возможность работы в режиме предделителя частоты.

Широкие возможности для управления работой МК предоставляет модуль прерываний. Он отслеживает (параллельно работе основной программы) наступления следующих событий: переполнение счетчиков/таймеров, поступление сигнала на специальный вход микроконтроллера, поступления сигнала о готовности памяти EEPROM после ее записи или чтения. В результате возникновения прерываний управление передается той или иной подпрограмме, где реализуется некоторый алгоритм реакции на соответствующее событие. С помощью соответствующих настроек МК можно разрешить или запретить прерывания любого типа или всех одновременно.

Типичной задачей для МК является обработка информации, поступающей от внешних устройств в аналоговой форме, в виде изменяющегося во времени напряжения, подаваемого на вход МК. В этих случаях в первую очередь требуется преобразование аналоговой информации в цифровую, чем и занимается модуль аналого-цифрового преобразователя (АЦП). Обычно аналоговую информацию вводят параллельно по нескольким каналам. Качество аналого-цифрового преобразования определяется разрядностью АЦП. Типичны 8-, 10- и 12-разрядные модули. Так, при 10-разрядном преобразовании диапазон входных значений сигнала разбивается на $2^{10} = 1\ 024$ градаций.

Модули цифро-аналогового преобразования в универсальных микроконтроллерах встречаются редко, однако с помощью несложных внешних приспособлений такое преобразование можно реализовать с помощью так называемых модулей широтно-импульсной модуляции (ШИМ), обычно присутствующих в МК. Модуль ШИМ генерирует последовательность прямоугольных импульсов с программно регулируемой скважностью (отношением длительности импульса к длительности межимпульсной паузы). Если сигнал от такого модуля подать на усредняющую цепочку, то на выходе будет получен аналоговый сигнал с величиной, пропорциональной скважности.

Важнейшими для МК являются аппаратные средства организации взаимодействия с внешним миром. Наиболее широко для этих целей используется модуль универсального последовательного асинхронного приемопередатчика USART. Его назначение заключается в преобразовании байтов выходной информации во временную последовательность битов и наоборот: входной последовательности битов — в байты, заносимые в память МК. Основной технической характеристикой здесь является скорость передачи/приема данных (например, 9 600 бод). Как

правило, эта скорость в USART может программно меняться в весьма широких пределах.

Последовательный код с выхода модуля может передаваться внешним устройствам с помощью различных последовательных интерфейсов. В частности, широко используется интерфейс RS232, позволяющий установить передачу данных между МК и последовательным портом (COM) персонального компьютера. Интерфейс RS485 позволяет подключить к одной линии связи сразу несколько внешних устройств. Отметим, что, кроме USART, в МК обычно реализованы и другие коммуникационные интерфейсы (CAN, I^2C и пр.).

Мы рассмотрели обычный набор модулей, включаемых в состав микроконтроллеров. В зависимости от типа МК этот набор может быть сокращен или наоборот расширен. Именно состав модулей определяет большое ценовое и функциональное разнообразие микроконтроллеров на рынке.

Организация памяти данных микроконтроллера

Как уже было сказано, память данных представляет собой совокупность регистров, которые подразделяются на регистры общего назначения и регистры специальных функций. В микроконтроллерах PIC вся область регистровой памяти разделена на два или четыре раздела, называемых банками памяти, с нумерацией от 0 до 3. При разбиении на два банка структура ОЗУ может выглядеть примерно так, как показано на рис. 1.3.

Младшие адреса отданы под регистры специальных функций, для которых в программах рекомендуется использовать стандартные имена типа STATUS, INTCON и пр. Разработчики программного обеспечения стараются поддерживать рекомендуемые имена, хотя встречаются и исключения. Об этом надо помнить во избежание недоразумений при рассмотрении конкретных программ.

Как видно из рис. 1.3, некоторые регистры специальных функций продублированы в обоих банках. Остальные присутствуют только в одном из банков. Для работы, например, с регистром TRISB необходимо перейти в банк 1 (исходное состояние МК предполагает нахождение в банке 0). Как это сделать, будет показано чуть позже.

Старшие адреса ОЗУ отданы под регистры общего назначения, т.е. под хранение оперативных данных. Количество предоставляемой под такие данные памяти зависит от типа МК и обычно составляет сотни байт. Следует отметить, что регистры специальных функций (точнее, часть из них) столь же доступны для записи и чтения из программы, как

и регистры общего назначения, но, разумеется, использование их для записи промежуточных данных чревато непредсказуемыми последствиями.

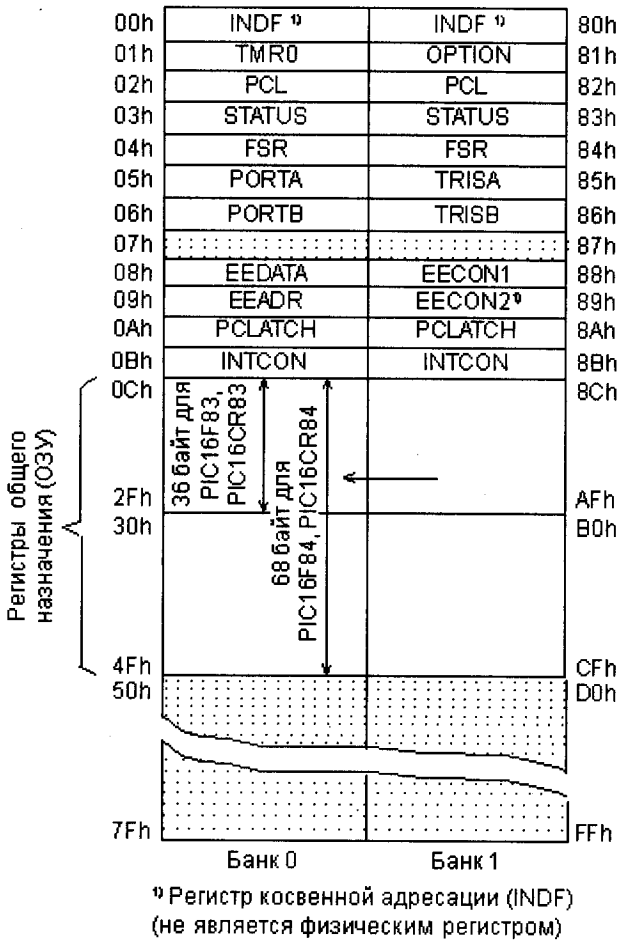


Рис. 1.3. Структура ОЗУ для двух банков памяти

Полное описание регистров специальных функций заняло бы немало страниц, поэтому мы рассмотрим в качестве примера только несколько из них. Справочную информацию по каждому из SFR можно найти в официальных технических описаниях каждого типа микроконтроллера (так называемых, *datasheet*).

Регистр статуса (STATUS) содержит признаки операции (арифметические флаги) АЛУ процессора, а также разряды, позволяющие определить состояние МК после включения или перезапуска и выбрать банк памяти данных. Назначение разрядов регистра STATUS представлено на рис. 1.4. Здесь и далее в первой строке указана доступность разрядов для записи (W) и чтения (R), а также их исходное (после включения или перезапуска) значение. Во второй строке дано символьное обозначение разряда, а в третьей — его номер.

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	/TO	/PD	Z	DC	C
7	6	5	4	3	2	1	0

Разряд 7 (IRP) — выбор страницы банка данных (при косвенной адресации):
0 = банк 0, 1 (00h – FFh);
1 = банк 2, 3 (100h – 1FFh);
Разряд IRP не используется в МК подгруппы PIC16F8X.

Разряды 6, 5 (RP1:RP0) — выбор страницы банка данных (при прямой адресации):
00 = банк 0 (00h – 7Fh);
01 = банк 1 (80h – FFh);
10 = банк 2 (100h – 17Fh);
11 = банк 3 (180h – 1FFh);
В МК подгруппы PIC16F8X используется только разряд RP0.

Разряд 4 (/TO) — момент срабатывания сторожевого таймера:
1 = после включения питания, а также по командам `CLRWDT` и `SLEEP`;
0 = по завершении выдержки сторожевого таймера.

Разряд 3 (/PD) — переход в режим пониженного энергопотребления:
1 = после включения питания, а также по команде `CLRWDT`;
0 = по команде `SLEEP`.

Разряд 2 (Z) — флаг нулевого результата:
1 = результат арифметической или логической операции нулевой;
0 = результат арифметической или логической операции ненулевой.

Разряд 1 (DC) — флаг десятичного переноса/заема (для команд `ADDWF` и `ADDLW`):
1 = присутствует перенос из четвертого разряда;
0 = перенос из четвертого разряда отсутствует.

Разряд 0 (C) — флаг переноса/заема (для команд `ADDWF` и `ADDLW`):
1 = присутствует перенос из старшего разряда;
0 = перенос из старшего разряда отсутствует.

Рис. 1.4. Назначение разрядов регистра STATUS (адрес 03h, 83h)

Регистр конфигурации (OPTION) содержит управляющие разряды для конфигурирования предварительного делителя (предделителя), внешних прерываний, таймера и подтягивающих (*pull-up*) резисторов на выводах порта B (будет пояснено далее). Назначение разрядов этого регистра представлено на рис. 1.5.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
7	6	5	4	3	2	1	0

Разряд 7 (/RBPU) — установка подтягивающих резисторов на выводах порта B:
0 = подтягивающие резисторы подключены;
1 = подтягивающие резисторы отключены.

Разряд 6 (INTEDG) — выбор активного фронта для сигнала прерывания:
0 = прерывание по спаду сигнала на выводе RB0/INT;
1 = прерывание по нарастающему фронту сигнала на выводе RB0/INT.

Разряд 5 (T0CS) — выбор источника сигнала таймера TMR0:
0 = внутренний тактовый сигнал (CLKOUT);
1 = переход на выводе RA4/T0CKI.

Разряд 4 (T0SE) — выбор активного фронта для источника сигнала для TMR0:
0 = приращение по нарастающему фронту сигнала на выводе RA4/T0CKI;
1 = приращение по спаду сигнала на выводе RA4/T0CKI.

Разряд 3 (PSA) — назначение предделителя:
0 = предделитель подключен к TMR0;
1 = пределитель подключен к сторожевому таймеру.

Разряды 2–0 (PS2:PS0) — выбора коэффициента деления для предделителя:

Значения разрядов	Скорость TMR0	Скорость WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Рис. 1.5. Назначение разрядов регистра OPTION (адрес 81h)

В том случае, когда prelделитель обслуживает сторожевой таймер, таймеру TMR0 назначается коэффициент предварительного деления 1:1.

Регистр условий прерывания (INTCON) содержит разряды доступа для всех источников прерываний (рис. 1.6).

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
7	6	5	4	3	2	1	0

Разряд 7 (GIE) — разрешение всех прерываний:
0 = все прерывания запрещены;
1 = разрешены все незамаскированные прерывания.

Разряд 6 (EEIE) — разрешение прерывания записи в EEPROM;
0 = прерывания записи в EEPROM запрещены;
1 = прерывания записи в EEPROM разрешены.

Разряд 5 (T0IE) — разрешение прерывания по переполнению TMR0:
0 = прерывания от TMR0 запрещены;
1 = прерывания от TMR0 разрешены.

Разряд 4 (INTE) — разрешение прерываний по входу RB0/INT:
0 = прерывания по входу RB0/INT запрещены;
1 = прерывания по входу RB0/INT разрешены.

Разряд 3 (RBIE) — разрешение прерываний по изменению PORTB:
0 = прерывания по изменению PORTB запрещены;
1 = прерывания по изменению PORTB разрешены.

Разряд 2 (T0IF) — запрос прерывания по переполнению TMR0:
0 = прерывание по переполнению TMR0 отсутствует;
1 = прерывание по переполнению TMR0 присутствует.

Разряд 1 (INTF) — запрос прерывания по входу RB0/INT:
0 = прерывание по входу RB0/INT отсутствует;
1 = прерывание по входу RB0/INT присутствует.

Разряд 0 (RBTF) — запрос прерывания по изменению PORTB:
0 = ни на одном из входов RB7:RB4 состояние не изменилось;
1 = хотя бы на одном из входов RB7:RB4 изменилось состояние.

Рис. 1.6. Назначение разрядов регистра INTCON (адрес 0Bh, 8Bh)

Разряд разрешения всех прерываний GIE сбрасывается автоматически по:

- включению питания;
- внешнему сигналу /MCLR при нормальной работе;
- внешнему сигналу /MCLR в "спящем" режиме;
- истечении времени выдержки сторожевого таймера при нормальной работе;

- истечении времени выдержки сторожевого таймера в "спящем" режиме.

Прерывание по входу INT может вывести процессор из "спящего" режима, если перед входом в этот режим разряд INTE был установлен в единицу. Состояние разряда GIE также определяет, будет ли процессор переходить на подпрограмму обработки прерывания после выхода из "спящего" режима.

Сброс разрядов, определяющих запрос на прерывание, должен осуществляться соответствующей программой.

Общение с внешним миром в микроконтроллерах осуществляется через так называемые порты ввода-вывода. В различных типах МК может быть реализовано от одного до пяти портов. На порт извне может быть подан восьмиразрядный двоичный код (байт данных). В этом случае порт работает на ввод, принимает данные от внешнего устройства и передает их на обработку процессору. При работе порта на вывод данные из процессора, наоборот, передаются в форме байта на порт и далее могут быть считаны внешним устройством.

С каждым портом связан свой регистр специальных функций. Например, для порта А таким регистром будет PORTA. Весь регистр порта или его отдельные разряды должны быть программно настроены на ввод или вывод путем записи соответствующего значения в регистр, управляющий портом. Для порта А таким управляющим регистром будет TRISA.

Если порт (или какой-то его вывод) настроен на ввод, то может возникнуть необходимость в его электрическом согласовании с внешним устройством. Для этого к выводу может быть подключен так называемый подтягивающий резистор. Если вторым концом резистор подключен к "земле", то он называется согласующим (*pull-down*), если же он подключен к линии питающего напряжения, то его называют подтягивающим (*pull-up*).

Микроконтроллер PIC16877A и его система команд

Микроконтроллер PIC16877A от компании Microchip является современным универсальным МК, позволяющим решать самые разнообразные задачи. В следующих главах подавляющая часть примеров реализована именно на этом устройстве, поэтому рассмотрим его более подробно.

Внешний вид МК, размещенного в так называемом DIP-корпусе, показан на рис. 1.7.

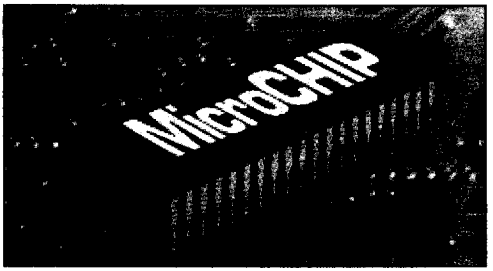


Рис. 1.7. Внешний вид микроконтроллера PIC в DIP-корпусе

Устройство имеет 40 выводов, расположенных вдоль левого и правого борта конструкции. Схема размещения выводов микроконтроллера PIC16877A показана рис. 1.8.

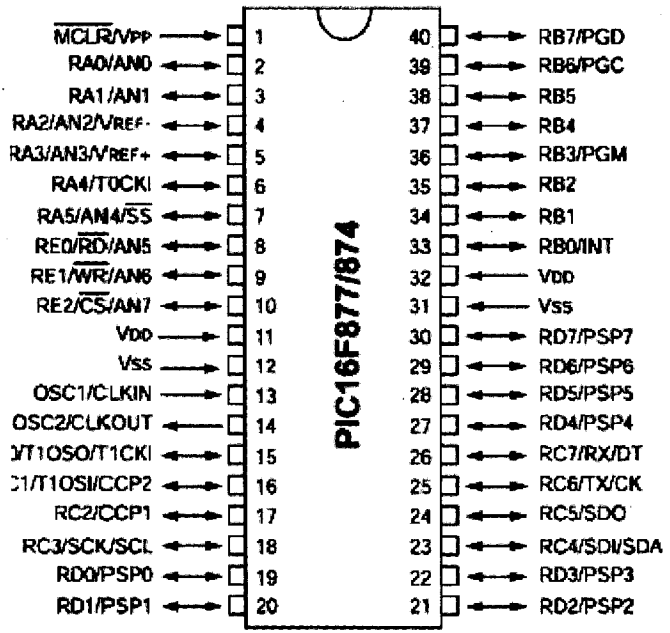


Рис. 1.8. Схема размещения выводов микроконтроллера PIC16877A

Часть выводов имеет единственное назначение. Так, вывод 37 подключен к разряду 4 порта B, а на вывод 31 может подаваться напряжение питания. Другие выводы мультиплексированы, т.е. имеют двойное

или тройное назначение. Например, вывод 2 может быть либо цифровым входом/выходом 0 порта A, либо одним из каналов аналогового входа AN0.

Основные технические характеристики микроконтроллера PIC16F877A в сравнении с параметрами других МК этого же семейства представлены в табл. 1.1.

Таблица 1.1. Технические характеристики микроконтроллера PIC16F877A

Параметр	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Тактовая частота	DC – 20 МГц	DC – 20 МГц	DC – 20 МГц	DC – 20 МГц
Сброс (задержка сброса)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Flash-память программ (14-разрядных слов)	4K	4K	8K	8K
Память данных (байт)	192	192	368	368
EEPROM-память данных (байт)	128	128	256	256
Прерываний	13	14	13	14
Порты ввода-вывода	A, B, C	A, B, C, D, E	A, B, C	A, B, C, D, E
Таймеры	3	3	3	3
Модуль захвата/сравнения/ШИМ	2	2	2	2
Модули последовательного интерфейса	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Модули параллельного интерфейса	—	PSP	—	PSP
Модуль 10-разрядного АЦП	5 каналов	8 каналов	5 каналов	8 каналов
Команд	35	35	35	35

Попутно заметим, что буква "F" в маркировке МК означает, что память программ выполнена по технологии Flash.

Карта регистровой памяти микроконтроллера PIC16877A показана на рис. 1.9. Как видим, здесь уже четыре банка. Большинство регистров последних двух банков продублированы в первых двух, но есть и такие (достаточно редко используемые), которые находятся только в старших банках.

Теперь кратко познакомимся с системой команд микроконтроллера PIC16877A. В нем используется так называемая RISC-система, содержащая сокращенный относительно некоторого стандартного набора список команд. Сокращение не означает ограничений в возможностях

процессора. Просто некоторые операции, в отличие от полной системы, требуют для своего выполнения не одной, а двух команд. RISC-система содержит 35 команд.

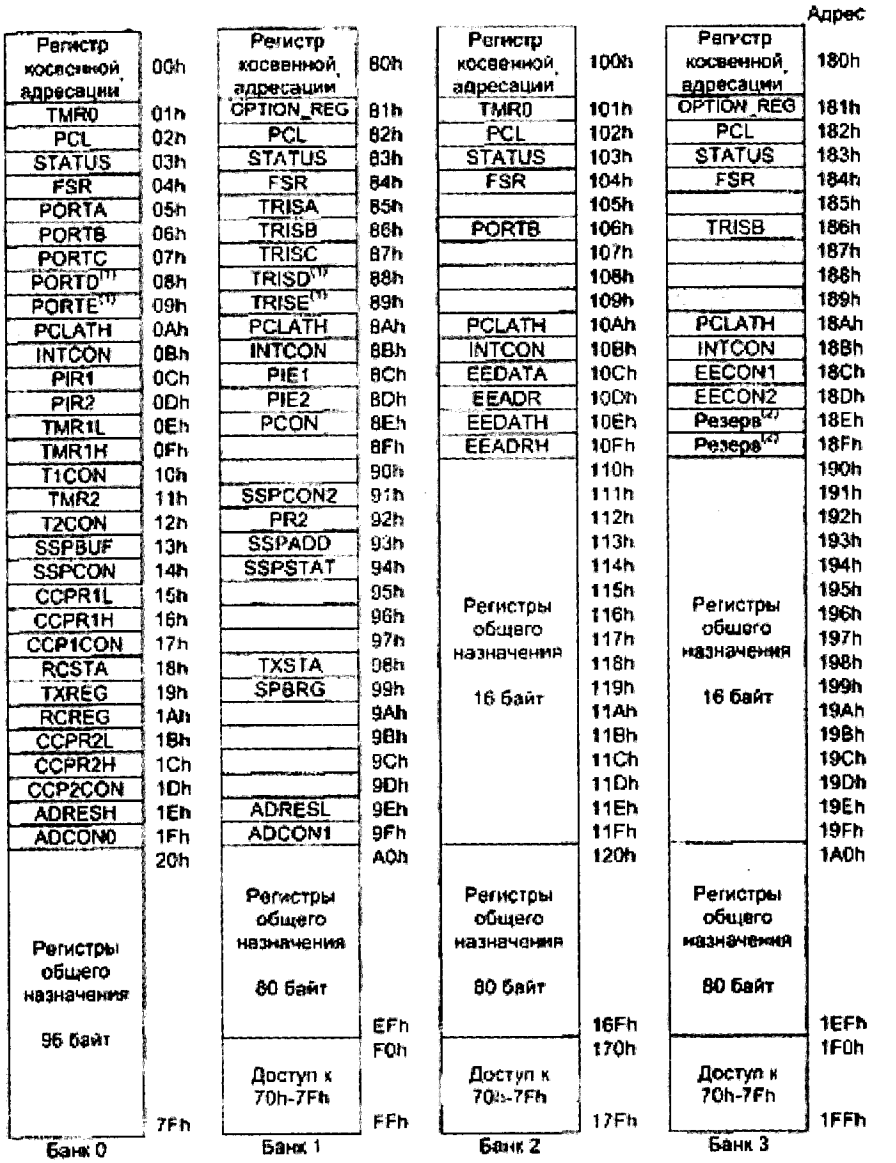


Рис. 1.9. Карта регистровой памяти микроконтроллера PIC16877A

Каждая машинная команда представляет собой 14-разрядное слово. Часть разрядов (с 3 по 6) содержит код операции, а младшие разряды могут содержать операнды команды (адреса, константы) или вообще ничего не содержать.

Команды программы хранятся последовательно в памяти программ. Адрес команды, передаваемой на выполнение, извлекается из специального 13-разрядного буфера, называемого счетчиком команд. Буфер образуют два регистра: PCL и PCLATH. Первый содержит младшие разряды адреса, а часть второго — старшие разряды. При линейном выполнении программы счетчик адреса инкрементируется (увеличивается на единицу), однако он доступен для записи так, что адрес следующей команды может быть программно установлен произвольным.

При написании программы (например, на ассемблере) команда записывается как мнемоническое имя и поля операндов. Расшифровка условных обозначений полей операндов представлена в табл. 1.2.

Табл. 1.2. Описание полей команд МК

Поле	Описание
f	Адрес регистра
w	Рабочий регистр
b	Номер разряда в восьмиразрядном регистре
k	Константа
x	Не используется. Ассемблер формирует код с x=0
d	Регистр назначения: d = 0 – результат в регистре w; d = 1 (по умолчанию) — результат в регистре f
label	Имя метки
TOS	Вершина стека
PC	Счетчик команд
PCLATH	Регистр PCLATH
GIE	Разряд разрешения всех прерываний
WDT	Сторожевой таймер
/TO	Тайм-аут
/PD	Выключение питания
dest	Регистр назначения: рабочий w или заданный в команде
[ ]	Необязательные параметры
( )	Содержание
→	Присвоение
< >	Поле номера разряда
∈	Из набора

Для команд работы с байтами **f** обозначает регистр, с которым производится действие, а **d** — разряд, определяющий, куда помещается результат. Если **d** = 0, то результат будет помещен в регистр **w** (рабочий регистр или аккумулятор). При **d** = 1 результат будет помещен в регистр **f**, упомянутый в команде. Для команд работы с разрядами **b** обозначает номер разряда, участвующего в команде, а **f** — регистр, в котором данный разряд расположен. Для команд передачи управления и операций с константами **k** обозначает 8- или 11-разрядную константу.

Почти все команды выполняются в течение одного командного цикла. В двух случаях исполнение команды занимает два командных цикла:

- проверка условия и переход;
- изменение счетчика команд как результат выполнения команды.

Как уже упоминалось, один командный цикл состоит из четырех периодов генератора. Таким образом, для генератора с частотой 8 МГц время исполнения командного цикла составляет 0,5 мкс.

В качестве справочной информации, полный список и описание команд RISC-системы представлен в табл. 1.3.

Таблица 1.3. RISC-система команд МК

Мнемоника	Описание команды	Циклы	Разряды состояния	Прим.
ADDWF f, d	Сложение W с f	1	C, DC, Z	1, 2
ANDWF f, d	Логическое "И" W и f	1	Z	1, 2
CLRF f	Сброс регистра f	1	Z	2
CLRW	Сброс регистра W	1	Z	
COMF f, d	Инверсия регистра f	1	Z	1, 2
DECF f, d	Декремент регистра f	1	Z	1, 2
DECFSZ f, d	Декремент f, пропустить команду, если 0	1(2)		1, 2, 3
INCF f, d	Инкремент регистра f	1	Z	1, 2
INCFSZ f, d	Инкремент f, пропустить команду, если 0	1(2)		1, 2, 3
IORWF f, d	Логическое "ИЛИ" W и f	1	Z	1, 2
MOVF f, d	Пересылка регистра f	1	Z	1, 2
MOVWF f	Пересылка W в f	1		
NOP	Нет операции	1		
RLF f, d	Сдвиг f влево через перенос	1	C	1, 2
RRF f, d	Сдвиг f вправо через перенос	1	C	1, 2
SUBWF f, d	Вычитание W из f	1	C, DC, Z	1, 2

Таблица 1.3. Окончание

Мнемоника	Описание команды	Циклы	Разряды состояния	Прим.
SWAPF f, d	Обмен местами тетрад в f	1		1, 2
XORWF f, d	Исключающее "ИЛИ" W и f	1	Z	1, 2
BCF f, b	Обнуление разряда в регистре f	1		1, 2
BSF f, b	Установка разряда в регистре f	1		1, 2
BTFSC f, b	Пропустить команду, если разряд в f равен нулю	1(2)		3
BTFSS f, b	Пропустить команду, если разряд в f равен единице	1(2)		3
ADDLW k	Сложение константы и W	1	C, DC, Z	
ANDLW k	Логическое "И" константы и W	1	Z	
CALL k	Вызов подпрограммы	2		
CLRWDT	Сброс сторожевого таймера	1	/TO, /P	
GOTO k	Переход по адресу	2		
IORLW k	Логическое "ИЛИ" константы и W	1	Z	
MOVLW k	Пересылка константы в W	1		
RETFIE	Возврат из прерывания	2		
RETLW k	Возврат из подпрограммы с загрузкой константы в W	2		
RETURN	Возврат из подпрограммы	2		
SLEEP	Переход в "спящий" режим	1	/TO, /P	
SUBLW k	Вычитание W из константы	1	C, DC, Z	
XORLW k	Исключающее "ИЛИ" константы и W	1	Z	

Примечания к таблице.

Если модифицируется регистр ввода-вывода (например, `MOVF PORTB, 1`), то используется значение, считываемое с выводов. Например, если в выходной защелке порта, включенного на ввод, содержится "1", а внешнее устройство формирует на этом выводе "0", то в разряде данных будет записан "0".

Если операндом команды является содержимое регистра TMR0 (и, если допустимо, d = 1), то предварительный делитель, если он подключен к TMR0, будет сброшен.

Если в результате выполнения команды изменяется счетчик команд или происходит переход по проверке условия, то команда выполняется за два цикла. Второй цикл выполняется как `NOP`.

Подробное изучение команд выходит за рамки этой книги. С их практическим использованием мы познакомимся на примерах программирования на ассемблере в заключительной главе.

Плата разработки EasyPIC5

Плата и разнообразные аксессуары к ней выпускаются сербской компанией Mikroelektronika. Вместе с платой поставляется достаточно подробное описание на английском языке, однако будет целесообразно рассмотреть конструктивные элементы и основные особенности устройства. Общий вид платы представлен на рис. 1.10. Далее все иллюстрации в данном разделе взяты из официального руководства по плате, выпущенного компанией Mikroelektronika.

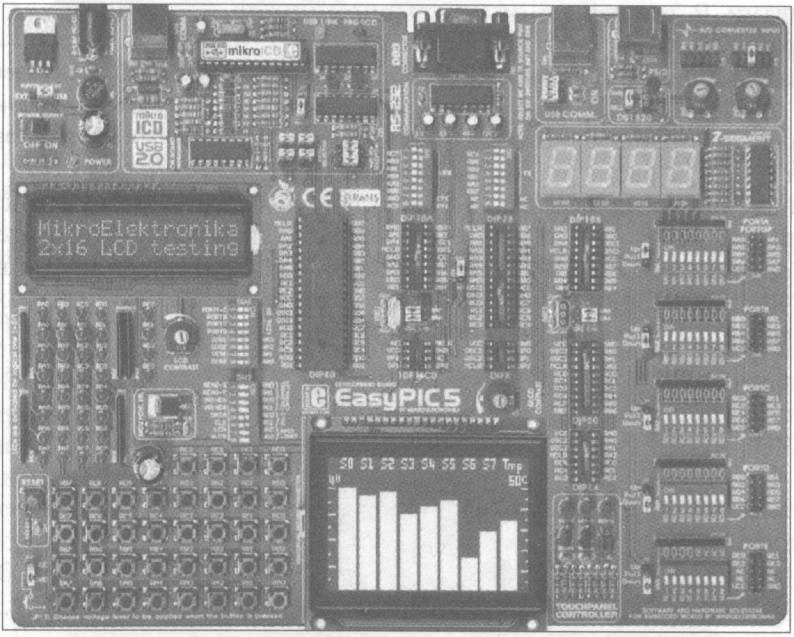


Рис. 1.10. Плата разработки EasyPIC5

Здесь, кроме базовой конструкции, представлены алфавитно-цифровой ЖК-дисплей, графический ЖК-дисплей и измеритель температуры. Эти устройства мы настоятельно рекомендуем приобрести вместе с платой. Кроме того, на рис. 1.11 показан дополнительный модуль: плата-разъем для подключения карты памяти CompactFlash, которой также желательно обзавестись. Имеет смысл приобрести в магазине электронных компонентов и так называемый пьезо-динамик. Это — простая "таблетка" для низкокачественного воспроизведения звукового сигнала. С помощью данного устройства мы в дальнейшем продемонстрируем некоторые звуковые возможности микроконтроллера.

На этой плате могут быть установлены практически все современные микроконтроллеры PIC (MCU) от компании Microchip, но только в так называемых DIP-корпусах. Различные панели для размещения MCU, отличающиеся количеством контактных ножек, расположены в центральной части платы (рис. 1.12).

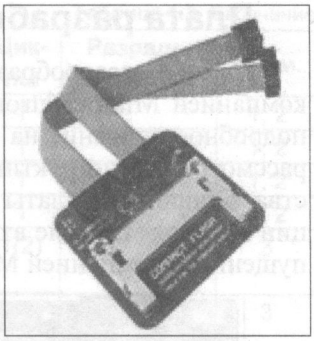


Рис. 1.11. Плата-разъем для подключения карты памяти CompactFlash

Здесь показан установленным MCU с 40 выводами. В дальнейшем мы в основном будем работать с микроконтроллером 16F877A (только в ряде примеров — с PIC18F4550). Обе эти микросхемы — с 40 выводами. Справа от установленного микроконтроллера на рис. 1.12 виден корпус внешнего кварцевого осциллятора, задающего тактовую частоту работы системы (в штатной схеме — 8 МГц). В дальнейшем такой осциллятор будет позиционироваться как HS. О возможностях использования других внешних или внутренних (установленных в самом MCU) осцилляторах при необходимости можно прочитать в описании платы.

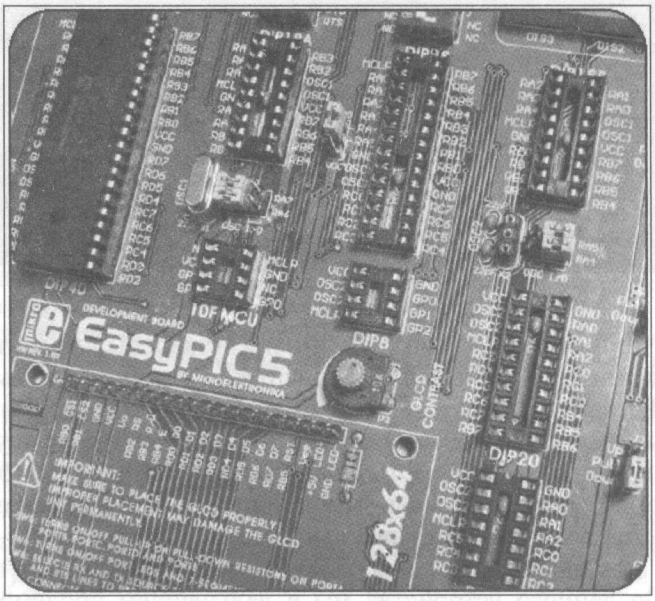


Рис. 1.12. Центральная часть платы EasyPIC5

Для конфигурирования платы на ней размещено множество управляющих элементов двух типов: групповые переключатели и перемычки. Эти элементы показаны на рис. 1.13.

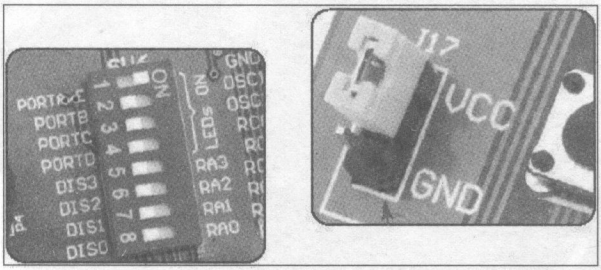


Рис. 1.13. Групповой переключатель и перемычка

Каждая позиция группового переключателя может быть включена (состояние "ON") или выключена. Перемычка может находиться в одном из двух положений: правое/левое или верхнее/нижнее.

Питание платы организовано от внешнего источника с напряжением 8..16 В мощностью порядка 1 Вт, подключаемого к разъему в левом верхнем углу платы, или через шину USB, подключенную к ПК. В этом случае шина выполняет двойную роль: источника питания и интерфейса с ПК. При использовании внешнего питания интерфейс USB с внешними устройствами реализуется через другой разъем на плате. Выбор типа питания осуществляется перемычкой J6 (положения "EXT" и "USB"). Ниже находится сам выключатель питания.

Устройство mikroICD на плате представляет собой программатор, осуществляющий запись (прошивку) программ в ПЗУ микроконтроллера. Работа устройства управляется программой PICFLASH, запущенной на ПК, и обеспечивается по шине USB при совмещении с функцией питания. В штатном режиме установленные по умолчанию состояния различных переключателей, связанных с программатором, обеспечивают работу с большинством типов микроконтроллеров.

Устройство платы обеспечивает полный доступ к портам микроконтроллера. Вдоль ее правого края (рис. 1.14) расположены пять разъемов на 10 выводов, подключенных к восьми входам/выходам портов A, B, C, D и E. Оставшиеся два вывода подключены к 5 В питающего напряжения (на схемах обозначается как VCC) и к "земле" (GND).

Расположенные рядом общие для всего порта перемычки J1–J5 и индивидуальные для каждого вывода групповые переключатели обеспечивают включение или отключение подтягивающих резисторов. Рези-

сторы могут подтягивать вывод к "земле" (положение переключателя "PullDown", как на рисунке) или к VCC (положение "PullUp").

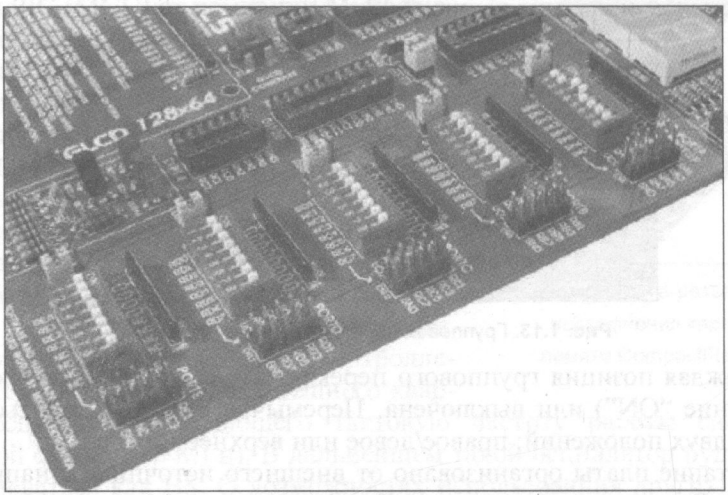


Рис. 1.14. Пять разъемов вдоль правого края платы

Вдоль левого края платы расположено поле светодиодов (LED) (рис. 1.15). Светодиоды подключены к выводам портов и включаются при формировании логических единиц в соответствующих разрядах.

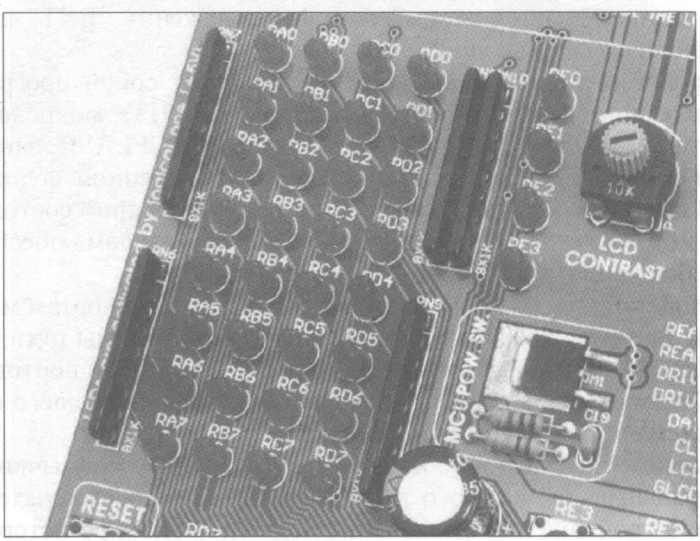


Рис. 1.15. Поле светодиодов вдоль левого края платы

С помощью первых четырех позиций группового переключателя SW6 светодиоды можно подключать или отключать. В левом нижнем углу платы (рис. 1.16) находятся кнопки, принудительно меняющие содержание разрядов портов.

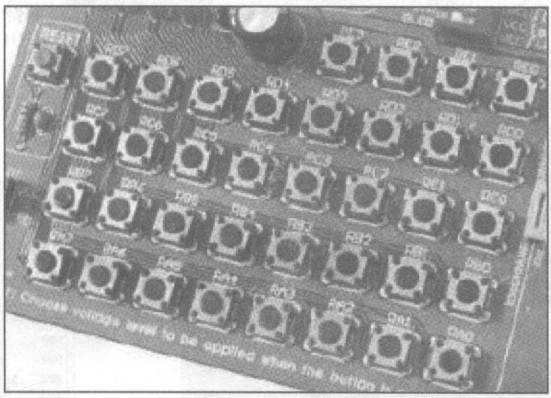


Рис. 1.16. Кнопки в левом нижнем углу платы

Если перемычка J17 находится в верхнем положении, то нажатие кнопки устанавливает соответствующий разряд в лог. 1 (временно, до отпускания кнопки). Если же J17 находится в нижнем состоянии, то в разряде устанавливается лог. 0. Кстати, здесь же рядом находится красная кнопка "RESET", перезапускающая МК.

В правой верхней части платы расположены три разъема: крайний правый — разъем PS/2 для подключения стандартной компьютерной клавиатуры, левее — разъем USB, альтернативный основному, совмещенному с питанием платы, и еще левее — коммуникационный разъем RS232. Для двустороннего обмена данными по интерфейсу RS232 (прием и передача) необходимо включить первые позиции групповых переключателей SW7 и SW8. Для использования альтернативного разъема USB все три перемычки J12 необходимо установить в правое положение. При работе с клавиатурой PS/2 требуется установка позиций 4 и 5 группового переключателя SW9 в положение "ON".

Для опробования аналого-цифрового преобразования на плате EasyPIC5 присутствует система из двух потенциометров, обеспечивающих в двух каналах плавную регулировку напряжения от 0 до 5 В, которое подается на преобразование. Соответствующая схема подключения показана на рис. 1.17.

Аналоговыми входами большинства микроконтроллеров являются выводы 0–5 порта A. Подключаемые каналы выбирают перемычками

J15 и J16. Согласно представленной схеме, подтягивающие резисторы для выводов, используемых в качестве аналоговых входов, должны быть отключены.

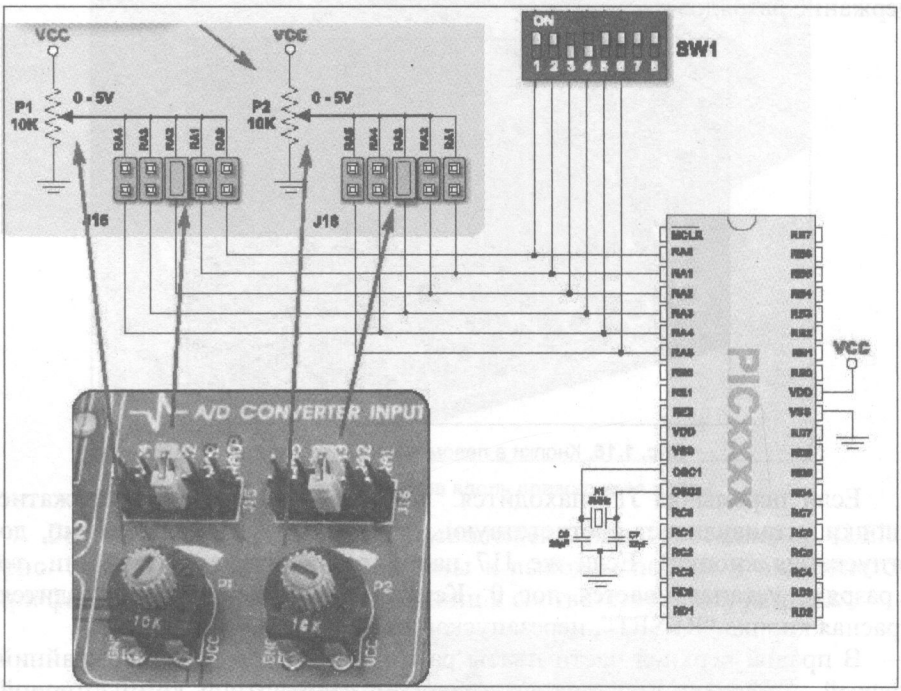


Рис. 1.17. Схема подключения двух потенциометров

Слева от потенциометров аналогового ввода находится термодатчик DS1820, который в комплект поставки платы не входит и должен приобретаться отдельно. Термодатчик подключается к MCU по специальному интерфейсу — так называемому OneWire, хотя на самом деле, это не один провод. При использовании специального программного обеспечения прибор позволяет измерять и отображать окружающую температуру в диапазоне от –55C° до +125C°. Датчик может быть подключен к выводу 5 порта A или к выводу 2 порта E, что выбирают с помощью перемычки J11. Необходимо строго следить за соблюдением полярности подключения датчика (рис. 1.18).

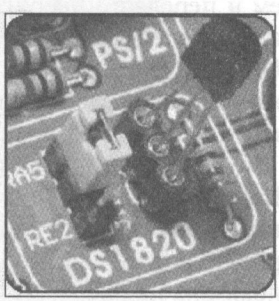


Рис. 1.18. Подключение термодатчика

При неправильном подключении термометр с большой долей вероятности сразу и окончательно выйдет из строя.

Ниже описанных компонентов находится группа из четырех семисегментных индикаторов (рис. 1.19). На каждом из них может быть высвечено схематическое изображение десятичных цифр от 0 до 9, а также символ десятичной точки, отделяющий целую часть числа от дробной. Индикаторы включаются и выключаются четырьмя старшими позициями переключателя SW6. При работе с индикаторами используются порты A и D.

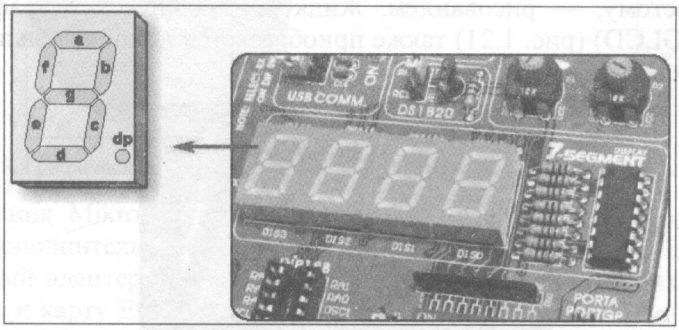


Рис. 1.19. Четыре семисегментных индикатора

Как нетрудно понять из рис. 1.19, состояние разрядов порта D управляет включением или выключением семи сегментов высвечиваемой цифры и десятичной точкой. В данном варианте устройства подключены по схеме с общим катодом. В других схемных реализациях могут применяться и другие подключения, определяющие, какое логическое значение на выводе соответствует высвечиванию элемента на индикаторе. Замечено, что яркость свечения элементов индикатора каким-то образом связана с положением потенциометров аналогового входа, что следует иметь в виду при работе с платой.

На рис. 1.20 показан установленный на плату алфавитно-цифровой жидкокристаллический дисплей (LCD), приобретаемый дополнительно. На дисплее могут отображаться символы, располагающиеся в двух строках, каждая длиной 16 позиций. Дисплей подключен к порту B по че-

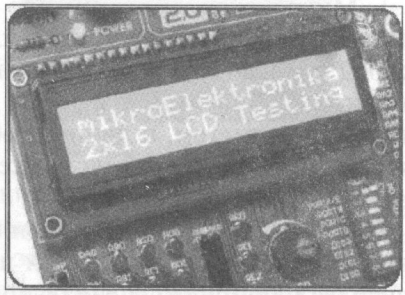


Рис. 1.20. Алфавитно-цифровой жидкокристаллический дисплей

тырехразрядной шине. Включение/выключение дисплея осуществляется седьмой позицией группового переключателя SW9. Ниже устройства расположен потенциометр, регулирующий яркость изображения. В руководстве по использованию платы подчеркивается, что ЖК-дисплей может подключаться в разъем на плате и извлекаться из него исключительно при выключенном питании.

Несмотря на, казалось бы, скромные ресурсы микроконтроллера PIC, он вполне успешно справляется с такими ресурсоемкими операциями как отображение информации на графическом дисплее или, говоря по-простому, — рисованием. Жидкокристаллический графический дисплей (GLCD) (рис. 1.21) также приобретается дополнительно к базовому комплекту EasyPIC.

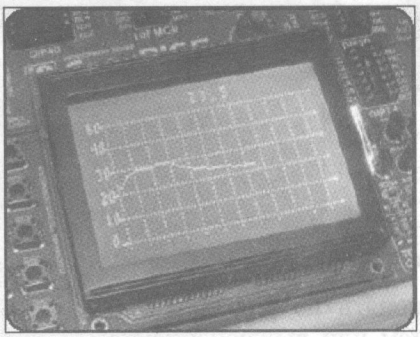


Рис. 1.21. Жидкокристаллический графический дисплей

На рис. 1.21 GLCD показан в рабочем состоянии. Для работы дисплея также задействован порт B. Регулятор яркости устройства расположен правее и выше. Для включения/выключения служит последняя позиция переключателя SW9.

Микроконтроллер успешно справляется и с обслуживанием такого аксессуара как Touch Screen или Touch-панель, которая представляет собой прозрачную рамку, накладываемую на экран графического ЖК-монитора. При этом по рамке можно рисовать стилусом или просто тонким предметом, и линии отображаются на экране. Кроме того, касаясь заданных позиций на рамке, можно выбирать графические кнопки разнообразных меню.

На рис. 1.22 показан графический дисплей с наложенной Touch-панелью в рабочем состоянии и подключение панели к разъему слева от дисплея с помощью специального шлейфа. Для активизации панели необходимо установить в состояние “ON” четыре младших позиции группового переключателя SW9.

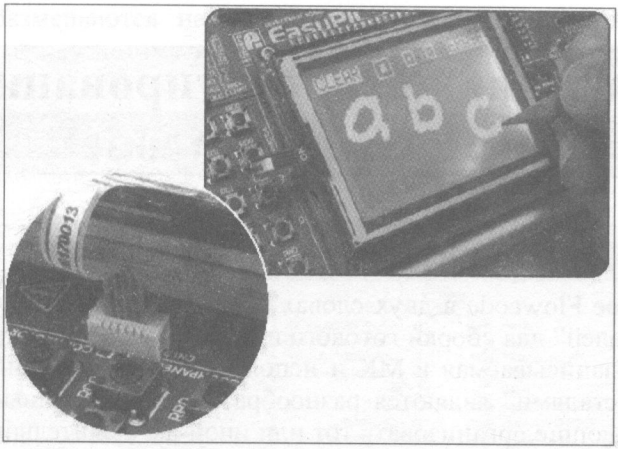


Рис. 1.22. Графический дисплей с наложенной Touch-панелью и его подключение

Компания Mikroelektronika предлагает широкий ассортимент различных дополнительных модулей, подключаемых к плате EasyPIC5. Упомянутый адаптер для карты Compact Flash — один из них. Можно упомянуть и карту Ethernet, часы реального времени (RTC), интерфейсный модуль RS485 и многое другое. Как правило, вместе с модулями продается программное обеспечение для поддержки этих устройств в виде исходных текстов программ (например, на языке C), реализуемых на компиляторах от этой же компании (например, MikroC).

Модули обычно подключаются непосредственно к разъемам прямого доступа к портам, как это показано на рис. 1.23. Уместно еще раз напомнить, что все подключения и отключения устройств следует выполнять исключительно при выключенном питании платы.

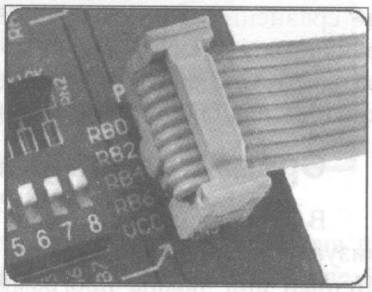


Рис. 1.23. Подключение модуля

В заключение отметим одну не вполне понятную особенность работы платы, выявленную эмпирически. Особую роль играет настройка подтягивающих резисторов порта B. Было обнаружено, что нормально разработанные проекты иногда правильно функционируют только при включении этих резисторов в состояние “PullDown”. Об этом следует помнить и проверять конфигурацию, если, казалось бы, абсолютно правильная разработка отказывается нормально работать.

Глава 2

Визуальное проектирование в системе Flowcode

Что такое Flowcode в двух словах? Это конструктор, состоящий из набора "деталей" для сборки готового изделия. Готовым изделием будет программа, записываемая в МК и исполняемая в микроконтроллерной системе. "Деталями" являются разнообразные программные конструкции, позволяющие организовать тот или иной вычислительный процесс, и компьютерные модели (симуляции) различных внешних устройств, подключаемых к МК. Поскольку все "детали" созданы разработчиками Flowcode, пользователю остается только правильно собрать из них действующую схему. Для этого, вообще говоря, не требуется даже знать языков программирования, хотя это все же желательно.

Хотя первоначально созданный проект работает именно в режиме симуляции, система создает и конечный, вполне работоспособный код, который может благополучно выполняться на реальном МК. За простоту разработки приходится расплачиваться, возможно, излишне расточительным использованием ресурсов микроконтроллера и более низким по сравнению с рассматриваемыми далее системами быстродействием, однако для многих задач это не критично. Кроме того, не будем забывать, что данная глава — лишь первый учебный шаг к освоению микроконтроллерного программирования.

Среда проектирования Flowcode

В этом вводном разделе мы познакомим читателя с организацией визуальной оболочки пакета Flowcode. Прежде всего, представим типичный вид экрана программы с открытым проектом. Основное окно будем называть рабочей областью. Ее вид показан на рис. 2.1.

Как во всех обычных Windows-приложениях, в верхней части окна расположено главное меню, пункты которого мы рассмотрим чуть позже. Ниже находится панель инструментов с кнопками, дублирующими вызов основных команд меню. Слева располагается инструментарий, который мы будем называть вертикальной линейкой. Отсюда мы будем брать "кубики конструктора", из которых будет состоять проект. Инст-

рументы размещаются на так называемой диаграмме: вертикальной структуре между автоматически создаваемыми началом BEGIN и концом END.

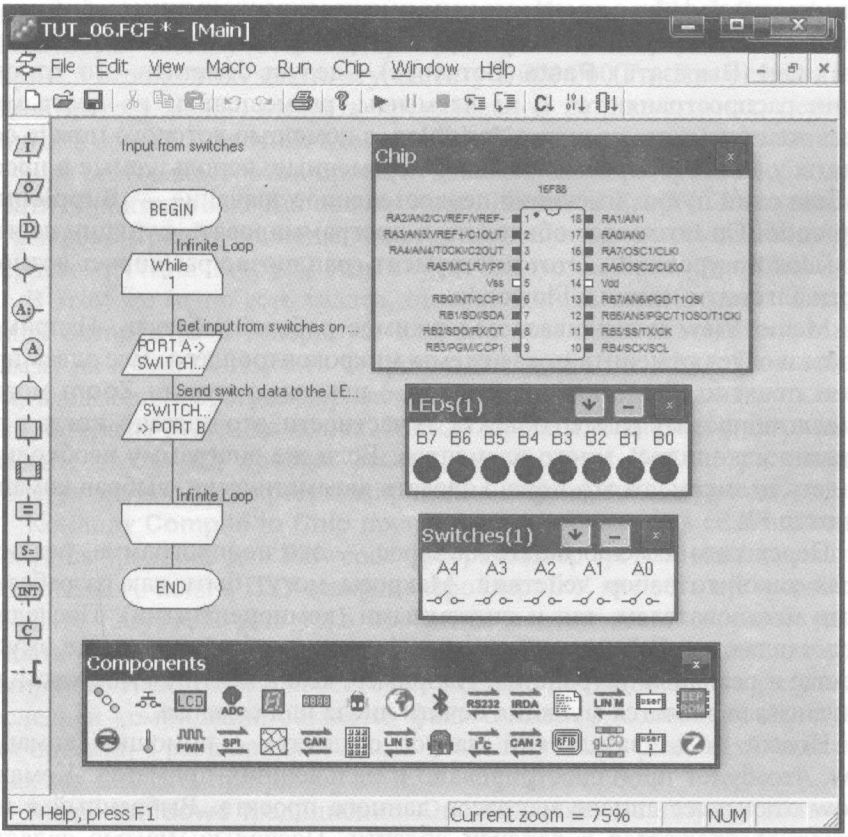


Рис. 2.1. Рабочая область Flowcode

В представленном варианте в нижней части окна расположена панель компонентов (может быть размещена и по-другому, в том числе, — под строкой меню). С ее помощью к проекту подключаются эмуляции различных внешних устройств. В частности, в примере на рис. 2.1 подключен эмулятор светодиодов (лампочки LEDs) и эмулятор кнопок Switches. Также в рабочей области показано схематическое изображение самого микроконтроллера со всеми его выводами. Оно не представляется существенно информативным и во всех описываемых ниже проектах будет отключено.

Итак, переходим к рассмотрению главного меню программы. Пункт **File** (Файл) совершенно типичен. Это меню позволяет открывать, закрывать и создавать проекты Flowcode. Проект представляет собой единственный файл с произвольным именем и расширением `.fcf`.

Меню **Edit** (Правка) содержит стандартные команды **Copy** (Копировать), **Cut** (Вырезать), **Paste** (Вставить). Следует отметить, что эти операции распространяются и на элементы, размещенные на диаграмме. Здесь же присутствует пункт **Variables**, с помощью которого можно создавать, удалять и переименовывать переменные, используемые в проекте. Еще один пункт, имеющий первостепенное значение, — **Supplementary code**. Он позволяет объявлять и программировать функции на языке высокого уровня С. Этот инструмент грандиозно расширяет возможности алгоритмизации в Flowcode.

Меню **View** настраивает содержимое рабочей области. Например, оно позволяет отменить показ схемы микроконтроллера. Все здесь просто и понятно. Отметим только, что с помощью пункта **Zoom** можно отмасштабировать размер объекта. В частности, это полезно, когда в диаграмме накопилось много элементов. Если же диаграмму необходимо увидеть целиком, то это можно сделать автоматически, выбрав команду **Zoom to Fit**.

Переходим к меню **Macro**. Макрос — это подпрограмма, реализующая какой-то набор действий. Макросы могут быть как разработанными пользователем, так и системными (компонентными). Последние представляют собой процедуры или функции, обслуживающие эмулируемые и реальные устройства. Например, компонентным макросом для ЖК-дисплея является функция печати числа или символа.

Новый пользовательский макрос создается с помощью команды **New**, что будет проиллюстрировано в дальнейших примерах. Команда **Show** открывает список макросов данного проекта. Выбранный в нем макрос отображается в рабочей области. Поскольку макрос является подпрограммой, он отображается в виде вторичной диаграммы, аналогичную по структуре основной.

Меню **Macro** содержит две важных команды: **Import** и **Export**. Макрос, созданный в одном проекте можно использовать в другом. Для этого он экспортируется в виде файла с расширением `.fcm`. В целевом проекте такой файл необходимо импортировать.

Следующий пункт меню, **Run**, служит для запуска проекта на выполнение в режиме эмуляции. Первая его команда просто запускает проект. Для этой же цели служит клавиша <F5>. Полезен также режим

пошагового исполнения, активизируемый по нажатию клавиши <F8>. Эмуляцию можно приостановить и завершить.

Пункт меню **Chip ► Target** позволяет выбрать конкретный микроконтроллер, для которого создается проект. Команда **Clock Speed** задает тактовую частоту осциллятора. Для реализации проекта на плате EasyPIC5 необходимо задать эту частоту 8 000 000 Гц. Для режима эмуляции частоту можно выбрать произвольной из допустимых, перечисленных в списке значений. Команда **Configure** позволяет выполнить некоторые особые настройки проекта. Здесь в первую очередь необходимо обратить внимание на тип осциллятора. На плате EasyPIC5 большинство микроконтроллеров подключается к внешнему высокостабильному кварцевому осциллятору. В этом случае следует выбрать тип **XTAL**.

В этом же меню есть раздел, отвечающий за компиляцию. Так, команда **Compile to C** генерирует код проекта, преобразованный в программу на языке С. Изучать С-программы крайне полезно. Пользователям, владеющим программированием на С, просмотр кода поможет понять алгоритм решения задачи. Между прочим, опытный программист заметит, что С-код проекта весьма примитивен. Это — расплата за удобства визуального программирования сверхвысокого уровня.

Команду **Compile to Chip** пропустим, поскольку для ее выполнения требуется "родной" для Flowcode программатор (устройство для записи исполняемого кода в ПЗУ микроконтроллера). У нас есть программатор на плате EasyPIC5. Для того чтобы им воспользоваться, необходимо создать этот самый исполняемый файл. Эту задачу решает команда **Compile to HEX**. Если в проекте все сделано правильно, то в результате последней компиляции создается файл с именем, совпадающим с именем проекта, с расширением `.hex`. В последующих разделах будет описано, как записать программу в МК и запустить проект "в железе".

Меню **Windows** традиционно. Оно позволяет манипулировать с окнами проекта. Наконец, меню **Help**, как всегда, помогает решать проблемы, возникающие при работе с Flowcode.

Теперь переходим к примерам реализации различных алгоритмов. Мы пойдем обычным путем от простого к сложному, поэтому последовательность рассмотрения примеров играет существенную роль.

Первый Blink

Слово "Blink" в нашем случае соответствует миганию лампочки. Говоря иначе, цель нашего первого проекта — программа, которая заставляет один из светодиодных индикаторов на плате EasyPIC5 переключаться с длительностью состояния, допустим, полсекунды.

Поскольку это, действительно, — наш "первый блин", то, чтобы он не оказался комом, процесс создания и модификации проекта опишем и проиллюстрируем графически более подробно. В последующих разделах стандартные аспекты разработки мы будем по возможности опускать.

Итак, при запуске программы Flowcode открывается основное окно приложения, на фоне которого сразу же отображается приглашение создать новый проект или открыть существующий (рис. 2.2).

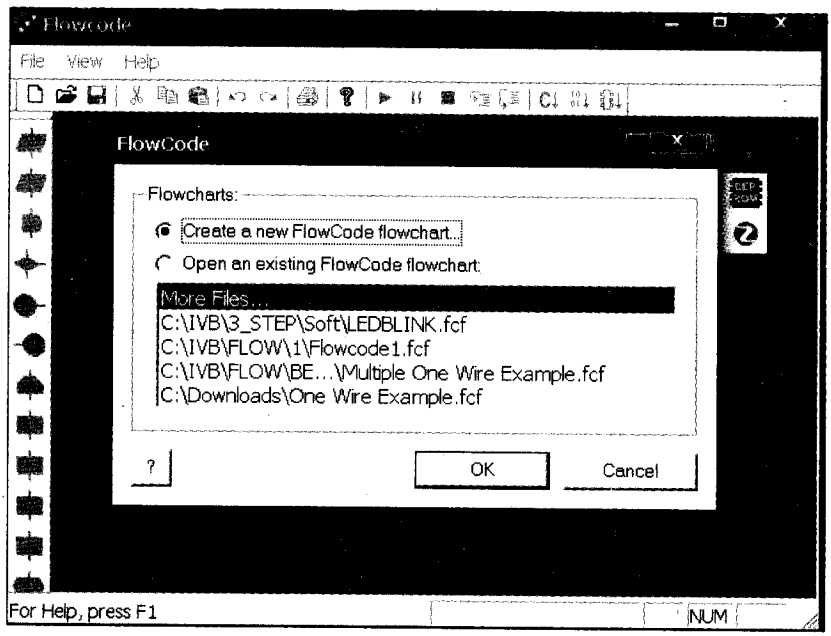


Рис. 2.2. Окно, открываемое при запуске Flowcode

Естественно, для первого проекта следует согласиться с выбранным по умолчанию вариантом **Create a new FlowCode flowchart**, и нажать кнопку **OK**.

Далее будет предложено выбрать тип микроконтроллера (MCU), с которым планируется работать (рис. 2.3). Как было отмечено ранее, в большинстве случаев мы будем программировать PIC16F877A.

На следующем этапе необходимо задать тактовую частоту осциллятора микроконтроллера. Для этого в главном меню программы выбираем команду **Chip ► Clock Speed**. В результате откроется диалоговое окно **Choose Clock and Simulation Speed**, показанное на рис. 2.4.

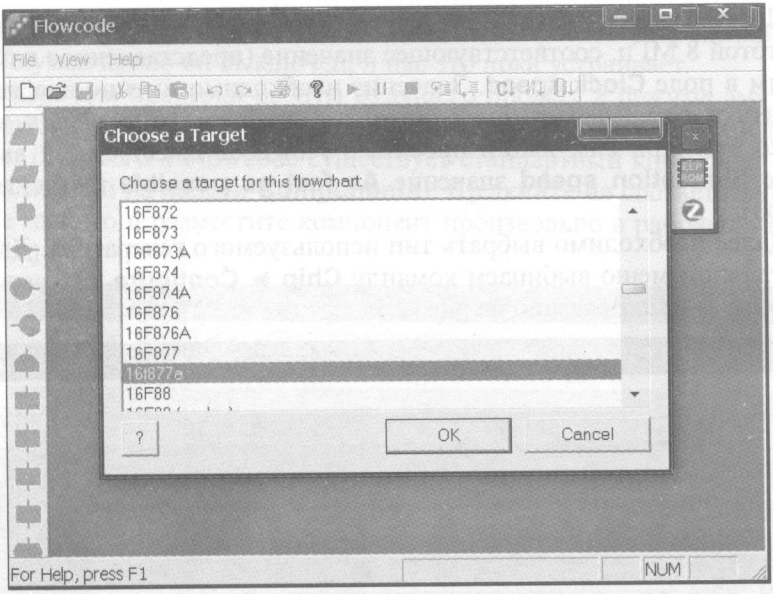


Рис. 2.3. Выбор типа микроконтроллера

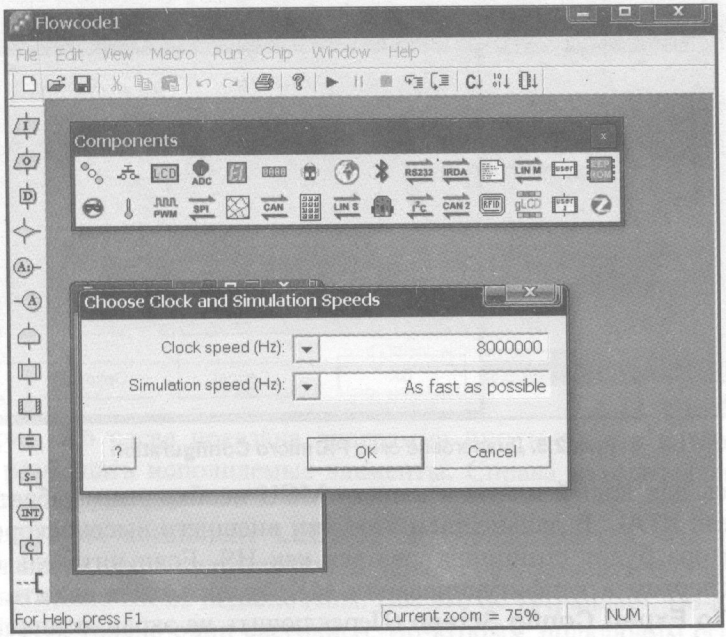


Рис. 2.4. Диалоговое окно **Choose Clock and Simulation Speed**

Поскольку на плате EasyPIC5 установлен кварцевый осциллятор с частотой 8 МГц, соответствующее значение (представленное в герцах) вводим в поле **Clock speed**. Значение необходимо именно ввести, а не выбрать в раскрывающемся списке, т.к. требуемого значения в этом списке нет. При симуляции проекта в Flowcode имеет смысл выбрать в поле **Simulation speed** значение **As fast as possible** (Максимально быстро).

Далее необходимо выбрать тип используемого генератора. Для этого в главном меню выбираем команду **Chip ► Configure**. Открывшееся диалоговое окно показано на рис. 2.5.

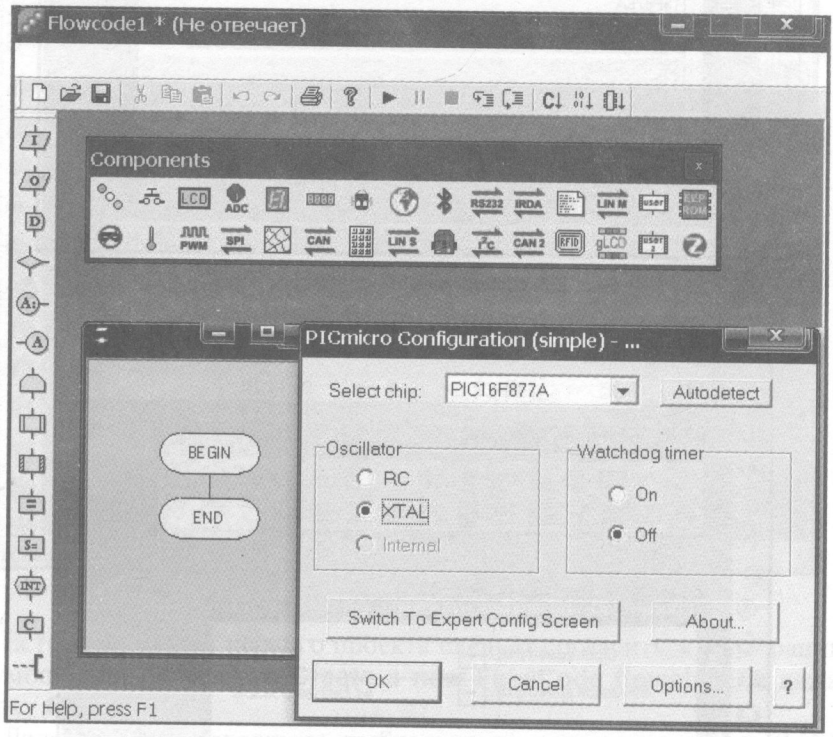


Рис. 2.5. Диалоговое окно **PICmicro Configuration**

Здесь для нашей платы и нашего MCU необходимо выбрать переключатель **XTAL**. В дальнейшем этот тип внешнего высокоскоростного осциллятора будет позиционироваться как HS. Если читатель считает себя экспертом в конфигурировании МК, то он может нажать кнопку **Switch To Expert Config Screen** (Переключить на экран экспертной на-

стройки). Это, в частности, позволит активизировать сторожевой таймер. По умолчанию он выключен, и нас это пока устраивает.

Теперь для проверки работы будущего проекта в режиме эмуляции мы должны разместить в рабочем окне лампочку, которой предстоит мигать. Для этого в Flowcode существует стандартный компонент **LEDs** (самый первый на панели **Components**). Щелкните на соответствующей кнопке мышью и разместите компонент произвольно в рабочей области (рис. 2.6).

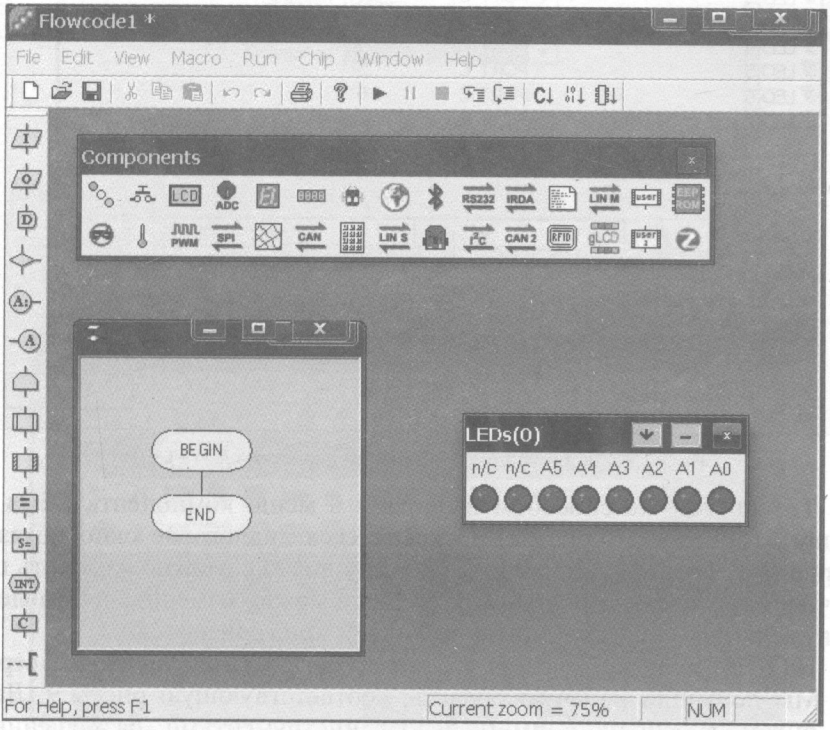


Рис. 2.6. Компонент **LEDs**

На рис. 2.6 слева показана диаграмма BEGIN–END, в которой мы и будем размещать исполняемые элементы. Справа находится линейка из восьми светодиодов.

Взглянув на компонент **LEDs(0)**, можно догадаться, что первые (справа налево) шесть светодиодов подключены к выводам 0–5 порта A. Остальные лампочки не подключены. Нас это не устраивает. Во-первых, нам требуется только один светодиод. Во-вторых, подключим его к выводу 0 порта C. Для этого нажмите кнопку с направленной вниз стрел-

кой, расположенную в строке заголовка компонента **LEDs**. В открывшемся меню выберите пункт **Component Connections** (Соединения компонента). Интересующая нас конфигурация показана на рис. 2.7.

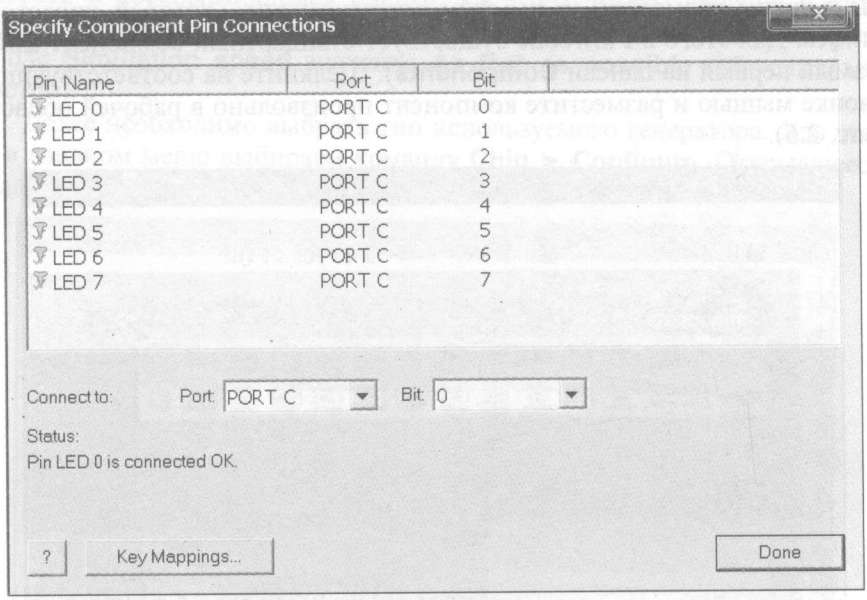


Рис. 2.7. Конфигурация соединений для компонента **LEDs**

Теперь тем же способом выбираем в меню компонента **LEDs** команду **Properties** (Свойства). Открывшееся диалоговое окно показано на рис. 2.8. Здесь задается число светодиодов (**1**). Можно изменить цвет свечения. Если светодиодов несколько, то можно изменить направление их нумерации и ориентацию по вертикали или горизонтали.

На этом предварительная работа по организации проекта завершена. Мы получили рабочую область, соответствующую рис. 2.9. Далее мы будем, в основном, иметь дело с инструментами, размещенными в вертикальной линейке вдоль левого края окна Flowcode.

Прежде всего, необходимо создать бесконечный цикл работы МК. В отличие от программ для ПК, микроконтроллерные системы, как правило, работают в бесконечном цикле, который прерывается только при аварийных ситуациях или по принудительному выключению. Для организации цикла необходимо перетащить мышью с линейки инструментов на диаграмму `BEGIN-END` элемент **Loop** (седьмой сверху). Отпустить кнопку мыши можно в тот момент, когда появится желтая стрелка. Полученная рабочая область теперь соответствует рис. 2.10.

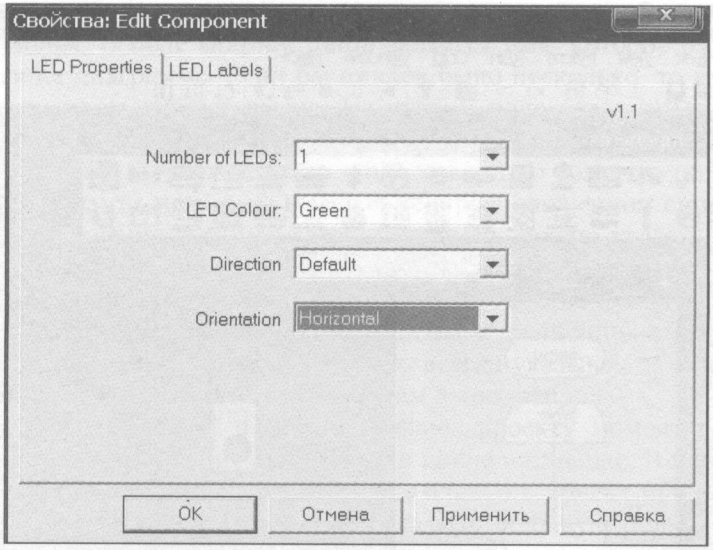


Рис. 2.8. Окно свойств компонента **LEDs**

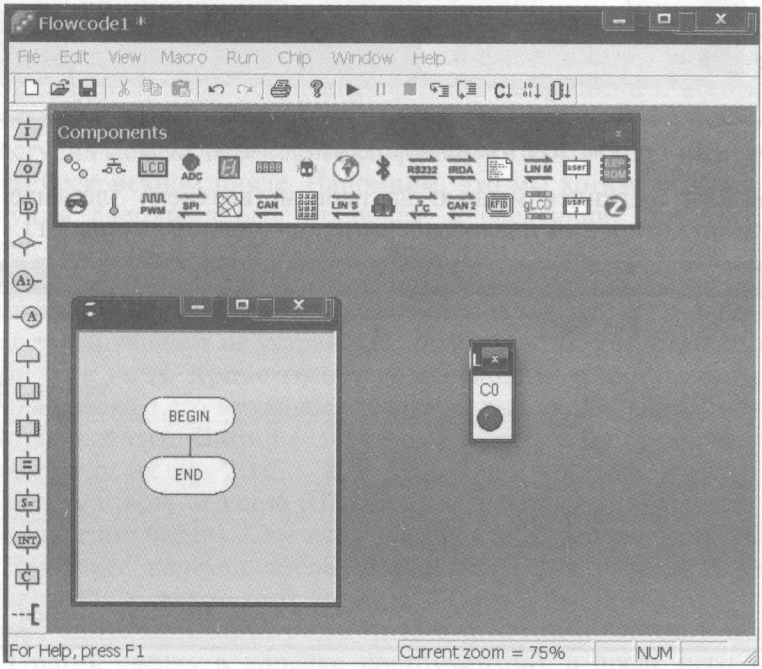


Рис. 2.9. Измененный вид компонента **LEDs**

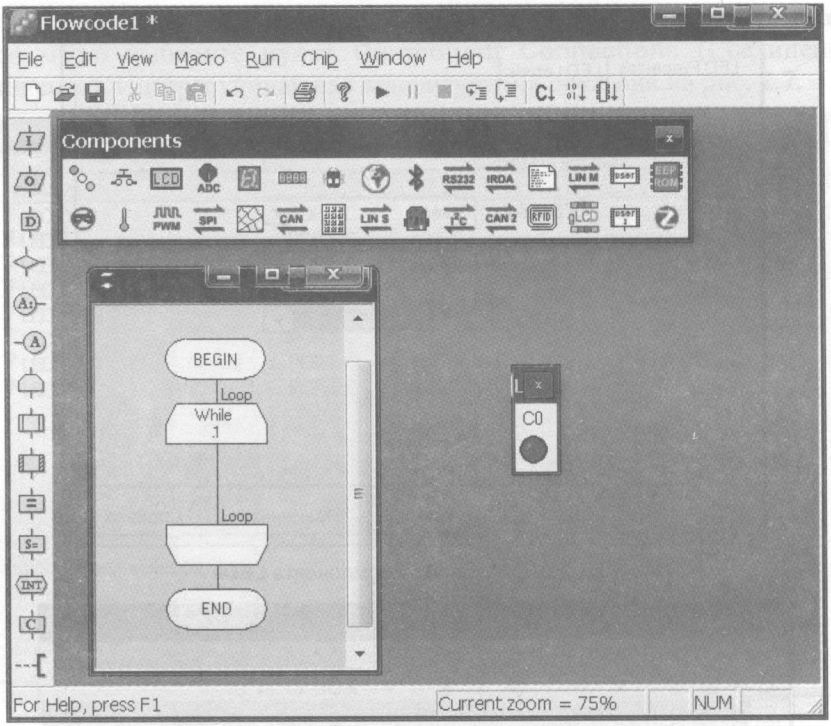


Рис. 2.10. К диаграмме добавлен элемент **Loop**

Дважды щелкните мышью на первом (верхнем) элементе **Loop**. В результате откроется диалоговое окно, показанное на рис. 2.11.

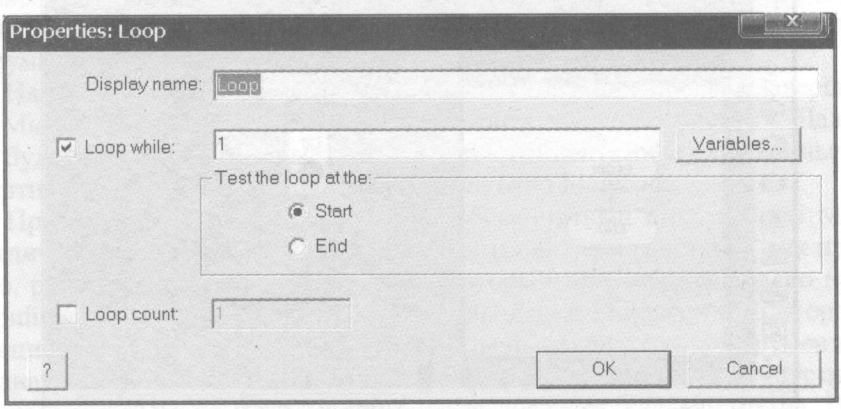


Рис. 2.11. Окно свойств элемента **Loop**

Здесь изменять ничего не нужно, однако следует разобраться в том, что мы видим. В поле **Display name** задается имя, которое будет отображаться на диаграмме. Если бы циклов было несколько, то имело бы смысл дать им различные имена. На втором поле остановимся поподробнее.

Установленный флажок **Loop while** говорит о том, что цикл будет выполняться до тех пор, пока выражение в расположенном справа поле возвращает значение ИСТИНА. Здесь, например, можно указать условие **x < 1**. В общем случае такое выражение возвращает значение ИСТИНА, если оно не равно нулю, или ЛОЖЬ в случае равенства нулю. Такое толкование поля **Loop while** наиболее близко к концепциям языка программирования С. В нашем случае здесь всегда указывается **1**, т.е. цикл выполняется всегда (бесконечно), что нам и требуется.

Переключатели **Start** и **End** указывают проекту момент проверки условия: перед выполнением операций в цикле или после. В нашем случае это, естественно, значения не имеет, однако в других примерах выбор одного из переключателей может играть большую роль. Заметим, что предварительная проверка и постпроверка реализованы в языке С различными циклами: `while` и `do-while`.

Если необходимо задать конкретное количество выполнений цикла (как, например, в цикле `for` языка С), то флажок **Loop while** следует сбросить, а вместо него установить флажок **Loop count** и ввести интересующее число в расположенном справа поле.

Наступил момент определить операции, которые будут выполняться в нашем бесконечном цикле. Напомним, что мы хотим, чтобы в цикле полсекунды светодиод был отключен, а следующие полсекунды — включен. Для начала перетащим на диаграмму с вертикальной линейки инструментов элемент **Output** (второй сверху). Действие этого элемента по умолчанию заключается в подаче на порт А некоторого значения.

Нас такой вариант не устраивает. Мы хотим, чтобы выходным портом был С, а не А. Кроме того, нам необходимо работать не со всем портом, а только с его младшим (нулевым) выводом. Дважды щелкните мышью на элементе **Output** в диаграмме. В результате откроется окно, представленное на рис. 2.12.

В поле **Variable or value** (Переменная иди значение) введите **0** (выходное значение порта). Сам порт задаем в следующем поле: **PORT C**. Далее выберите переключатель **Single Bit** (Один разряд) и укажите в расположенном справа поле значение **0** (рис. 2.13).

Ниже элемента **Output** разместите в диаграмме элемент задержки **Delay** (третий сверху в линейке инструментов), а после него — еще

один элемент **Output** и еще один элемент **Delay**. Все эти элементы должны находиться внутри организованного цикла. Теперь рабочая область соответствует рис. 2.14.

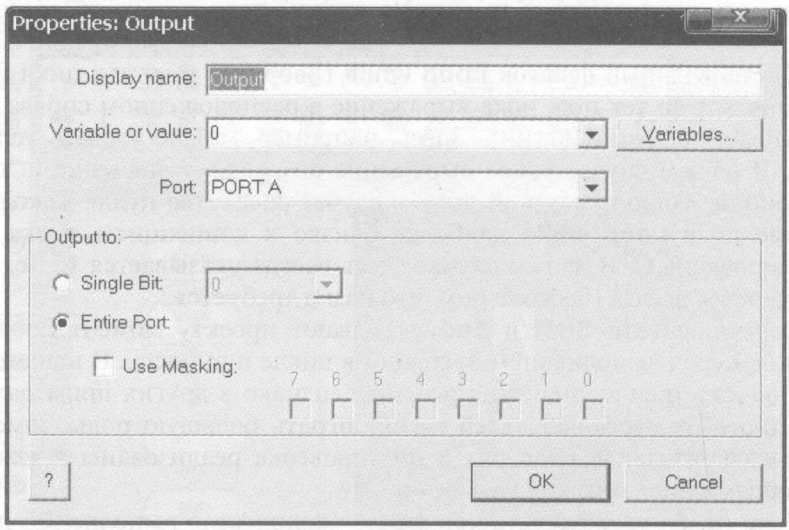


Рис. 2. 12. Окно свойств элемента **Output**

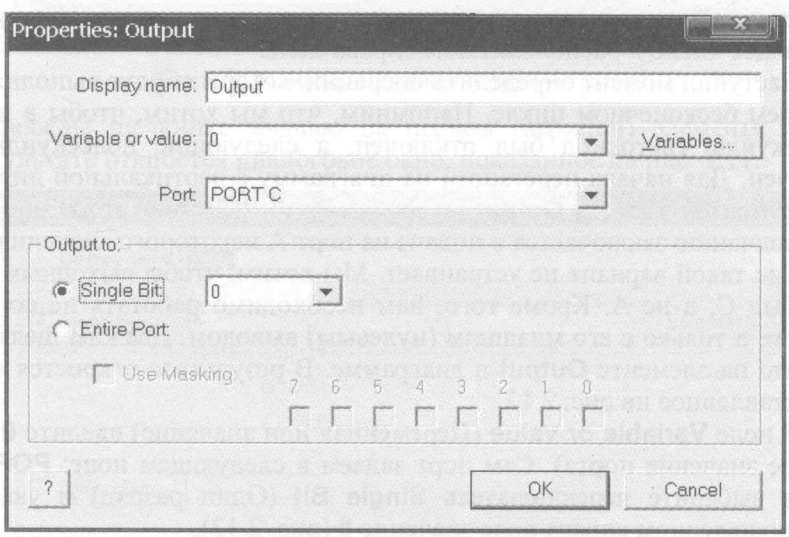


Рис. 2.13. Измененные свойства элемента **Output**

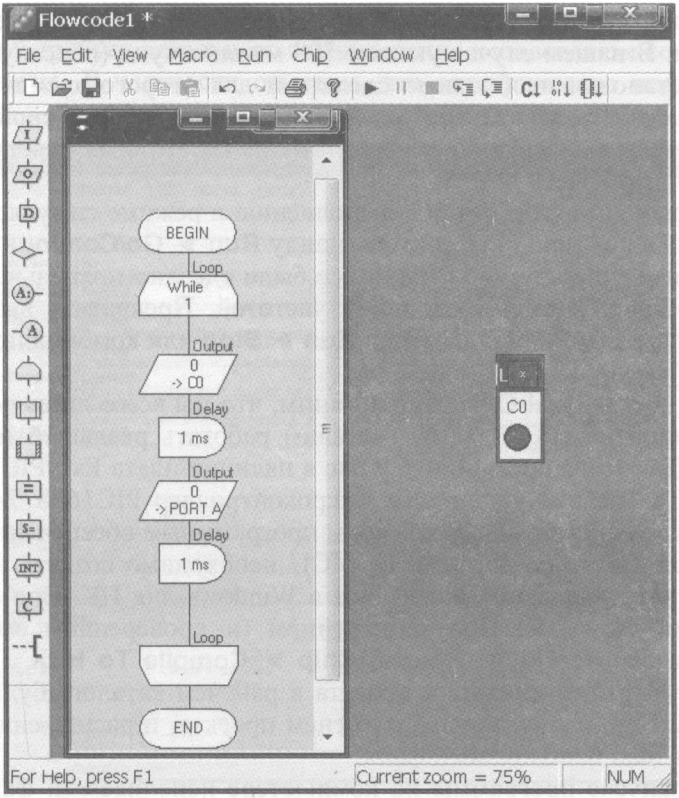


Рис. 2.14. Диаграмма с двумя элементами **Output** и двумя элементами **Delay**

На очереди разговор о задержках. Они создают в работе программы паузу заданной длительности. Дважды щелкните мышью на первом из элементов **Delay** в диаграмме. В результате откроется соответствующее окно свойств (рис. 2.15).

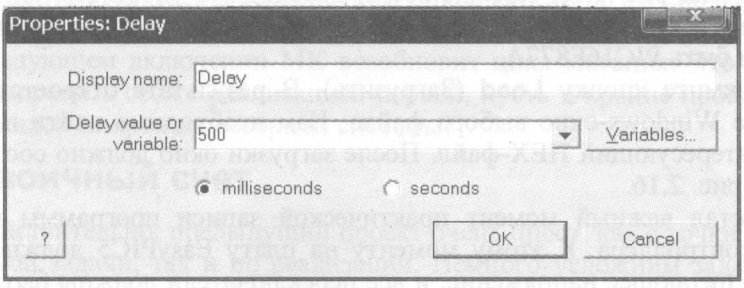


Рис. 2.15. Окно свойств элемента **Delay**

Здесь, в принципе, все понятно. Задается единица измерения длительности. В нашем случае укажем 500 миллисекунд (полсекунды). Такие же установки необходимо сделать и для второго элемента **Delay**. Для второго элемента **Output** следует определить те же свойства, что и для первого, однако выводимое значение на этот раз будет не **0**, а **1** (светодиод включен).

Теперь наш проект готов к выполнению в режиме симуляции Flowcode. В главном меню выберите команду **Run ► Go/Continue** или просто нажмите клавишу <F5>. Если все было сделано правильно, то лампочка начинает мигать с заданной частотой. Прекратить выполнение можно с помощью команды меню **Run ► Stop** или комбинации клавиш <Shift+F5>.

Цель достигнута! Однако, напомним, что мы всего лишь реализовали эмуляцию на ПК. Давайте заставим работать реальный микроконтроллер. Для этого примем, что у нас в наличии плата EasyPIC5 от Mikroelektronika, на ней установлен микроконтроллер PIC16F877A и соответствующий программатор со своим программным обеспечением.

На данном этапе для нашего MCU необходимо создать исполняемую программу, аналог файла `.exe` в Windows для ПК. Она создается средствами Flowcode. При загруженном (и проверенном эмуляцией) проекте выберите команду меню **Chip ► Compile To HEX**. После нескольких секунд компиляции проекта в рабочем каталоге будет создан файл с именем, совпадающим с именем проекта, и расширением `.hex`. Это и есть наш исполняемый файл.

При запуске программы на компьютере исполняемый файл загружается в оперативную память. Для микроконтроллеров дело обстоит иначе. Программа записывается ("прожигается") в постоянную перезаписываемую память. Именно этим и занимается программно-аппаратный инструмент PicFLASH от компании Mikroelektronika. Программная часть установлена на компьютере. При ее запуске открывается окно, показанное на рис. 2.16. Прежде всего, необходимо удостоверится в том, что здесь выбран требуемый тип микроконтроллера. В нашем случае это должен быть PIC16F877A.

Нажмите кнопку **Load** (Загрузить). В результате откроется стандартное Windows-окно выбора файла. Нам необходимо найти и загрузить интересующий HEX-файл. После загрузки окно должно соответствовать рис. 2.16.

Настал важный момент практической записи программы в ПЗУ микроконтроллера. К этому моменту на плату EasyPIC5 должно быть подано питающее напряжение, и все переключатели должны быть уста-

новлены в состояние по умолчанию. Последнее замечание достаточно серьезное и требует детального ознакомления пользователя со спецификацией платы. В частности отметим, что электрическое подключение светодиодов к портам осуществляется переключателем SW6.

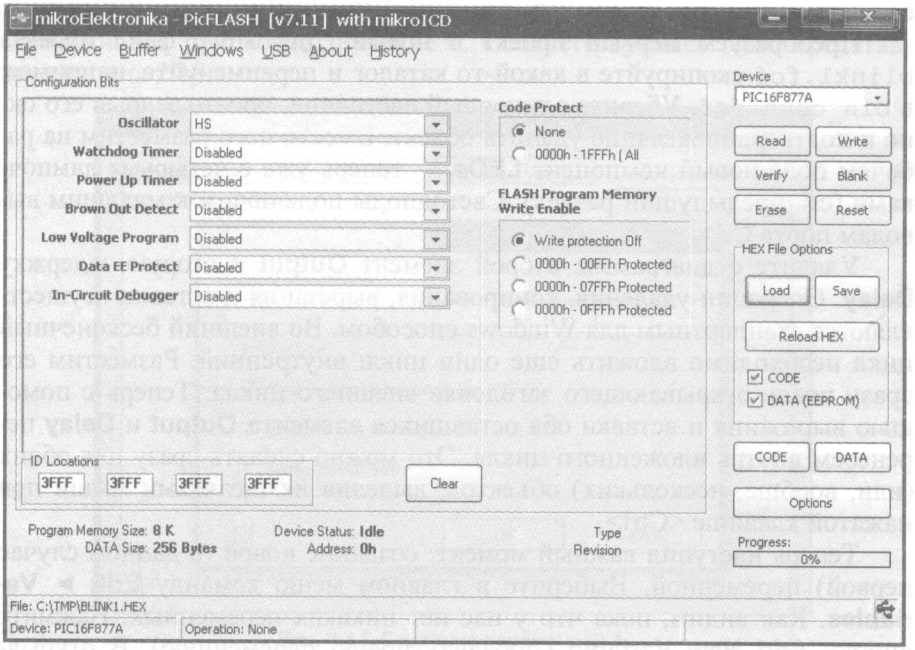


Рис. 2.16. Окно **PicFLASH**

Нажмите кнопку **Write** (Записать). Процесс "прожига" займет несколько секунд, после чего МК начнет выполнять действия, предписываемые ему разработанной программой: с частотой 1 Гц начнет мигать светодиод, соответствующий выводу 0 порта С. Как и ожидалось, процесс будет повторяться в бесконечном цикле вплоть до выключения питания прибора. Поскольку программа записана в постоянную память, при следующем включении МК возобновит цикл мигания. Что ж, можем поздравить себя с реализацией первой, пусть и крайне простой, но работающей микропроцессорной системы!

Двоичный счет

Действительно, предыдущий проект очень прост, как с точки зрения решаемой задачи, так и по реализации. Немного усложним задачу. Заставим МК отображать на светодиодах последовательно во времени

двоичные числа от 0000 до 1111 с бесконечным повторением последовательностей. Примем, что в каждом из четырех двоичных разрядов 0 соответствует выключенному светодиоду, а 1 — включенному. Экспозиция высвечивания каждого числа, по-прежнему, пусть составляет полсекунды.

Преобразуем первый проект в новый. Для этого файл проекта `blink1.fcf` скопируйте в какой-то каталог и переименуйте, например, в `bin_coun.fcf`. Уберите одиночный светодиод, просто закрыв его окно и подтвердив желание удалить объект. Вместо него разместим на рабочем поле новый компонент **LEDs** — теперь уже с четырьмя лампочками (см. предыдущий раздел). Светодиоды подключим к младшим выводам порта С.

Удалите с диаграммы второй элемент **Output** и вторую задержку **Delay**. Операции удаления, копирования, вырезания и вставки осуществляются стандартным для Windows способом. Во внешний бесконечный цикл необходимо вложить еще один цикл: внутренний. Разместим его сразу после открывающего заголовка внешнего цикла. Теперь с помощью вырезания и вставки оба оставшихся элемента **Output** и **Delay** перенесем внутрь вложенного цикла. Это можно сделать сразу для обоих (или, вообще, нескольких) объектов, выделив их щелчками мыши при нажатой клавише <Ctrl>.

Теперь наступил важный момент: создание новой (в данном случае первой) переменной. Выберите в главном меню команду **Edit ▶ Variables**. Как видим, пока что у нас нет никаких переменных. Нажмите кнопку **Add New Variable** (Добавить новую переменную). В открывшемся диалоговом окне **Create a New Variable** введите в поле **Name of new variable** (Имя новой переменной) **i** (рис. 2.17).

Как видим, Flowcode не блещет разнообразием типов данных. Их всего три:

- байтовый — целочисленные значения в диапазоне от 0 до 255;
- знаковый — целые в диапазоне от –32768 до +32767;
- строковый — массивы символов.

По умолчанию длина строки составляет 20 знаков, но может быть изменена. Поскольку мы будем оперировать двоичными числами от 0 до 1111, что соответствует десятичным значениям в диапазоне от 0 до 15, нам достаточно одного байта. Таким образом, соглашаемся с предложенным по умолчанию байтовым типом.

Сразу после открытия внешнего цикла необходимо инициализировать нашу новую переменную, присвоив ей значение **0**. Для этого в указанную точку диаграммы перетащите мышью с вертикальной линейки

десятый сверху элемент **Calculation**. Это очень важный инструмент. С его помощью можно организовывать последовательность вычислений, что и отражено в его названии. Двойной щелчок мышью на элементе **Calculation** открывает окно его свойств. В поле **Calculations** необходимо вписать простую операцию `i=0`, что следует понимать так: переменной `i` присвоить значение 0. Соответствующее окно представлено на рис. 2.18.

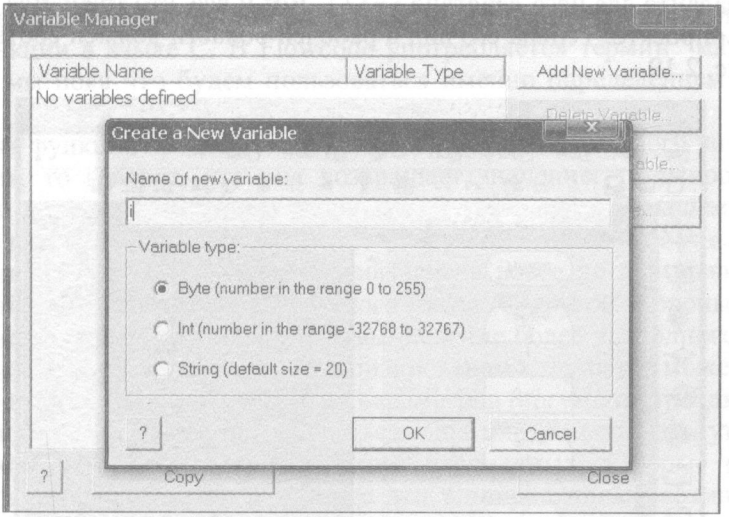


Рис. 2.17. Создание переменной `i`

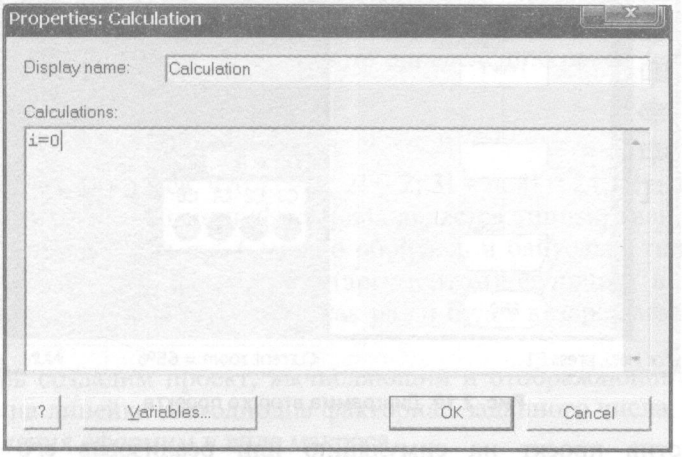


Рис. 2.18. Свойства элемента **Calculation**

Сейчас необходимо изменить заголовок вложенного цикла. Пока что он тоже выполняется бесконечно, нам же требуется, чтобы он выполнялся до тех пор, пока `i` меньше 16. Сделать это не составит труда. Открываем окно цикла и в поле **Loop while** вводим **i<16**.

Естественно, в цикле необходимо обеспечить инкрементирование (увеличение на единицу) переменной `i`. Это должно происходить после задержки. Вставьте в соответствующей точке еще один элемент **Calculation** и задайте для него действие `i=i+1`. Вот и все, что требуется сделать в новом проекте. Теперь его вид в рабочей области должен соответствовать рис. 2.19.

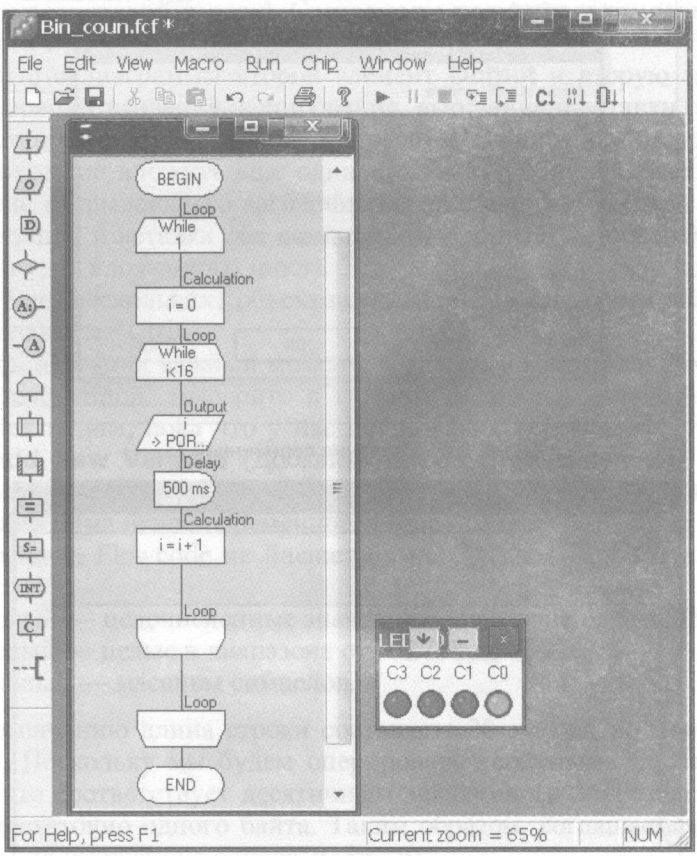


Рис. 2.19. Диаграмма второго проекта

Запустив проект на симуляцию или реализовав его на плате EasyPIC5, убеждаемся, что все работает, как требовалось.

Макросы, подпрограммы, процедуры, функции

Все эти названия, перечисленные в заголовке раздела, означают примерно одно и то же: запрограммированные алгоритмы решения каких-то задач, которые можно многократно выполнять в главной программе или даже использовать в отдельных самостоятельных программах. В наиболее обобщенном виде такие конструкции позиционируются как функции в языке С. В Flowcode употребляется термин "макросы", однако мы пока что будем пользоваться именно определением "функция".

Если функция предназначена для вычисления некоторого числового значения, то говорят, что она возвращает значение, типизированная, и ее тип определяется типом вычисляемого значения. В нашем случае функция (макрос) возвращает целое (`int`) или байтовое (`byte`) значение. Однако она может быть предназначена вовсе не для вычисления значений, т.е. не возвращает никакого числа. С другой стороны, функция может быть ориентирована на вычисление более чем одного значения. Тогда функция не является типизированной. Функция может иметь (или не иметь) один или несколько параметров (аргументов). Для реализации алгоритма может потребоваться одна или несколько внутренних или локальных для данной функции переменных, которые видимы только внутри самой функции. Все эти альтернативные возможности предусмотрены в Flowcode при организации макросов.

Рассмотрим математическую функцию, называемую факториалом числа. Факториал целого числа *n* — это произведение всех натуральных чисел от 1 до *n*. Факториал от нуля по определению равен 1. Обозначается факториал как *n!*:

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

Нетрудно убедиться, что 1! = 1; 2! = 2; 3! = 6; 4! = 24; 5! = 120 и т. д.

В нашей классификации факториал является типизированной функцией целого типа. Для $n < 6$ можно обойтись и байтовым типом. Само значение *n* является параметром (аргументом) функции, а результат произведения натуральных чисел как раз и будет возвращаемым значением.

Теперь создадим проект, вычисляющий и отображающий в двоичном виде на линейке светодиодов факториал заданного числа. Процедуру вычисления оформим в виде макроса.

Создайте новый проект Flowcode, назовите его `Macro` и сохраните в заданной папке. Настройте проект (частоту генератора, тип осциллятора) так же, как и раньше. Разместите компонент из восьми светодиодов и подключите его к порту C. В диаграмму вложите бесконечный цикл. Создайте новую переменную `n` байтового типа. С помощью инструмента **Calculation** перед циклом присвойте переменной `n` значение **5**. Это и будет то число, для которого рассчитывается факториал. Создайте еще одну байтовую переменную `fac`. В нее, в конце концов, будет записано вычисленное значение факториала. Все эти операции нам уже знакомы. В результате рабочее поле будет соответствовать рис. 2.20.

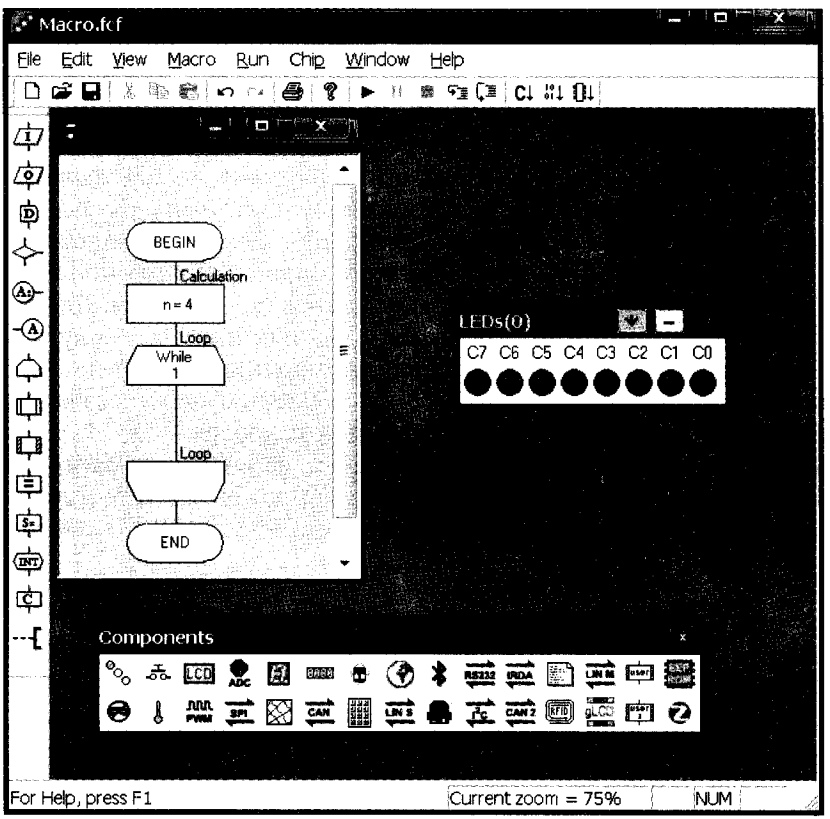


Рис. 2.20. Диаграмма для вычисления факториала

Приступаем к созданию макроса. Выберите команду меню **Macro ▶ New**. Открывшееся окно с введенным именем макроса в первом поле показано на рис. 2.21

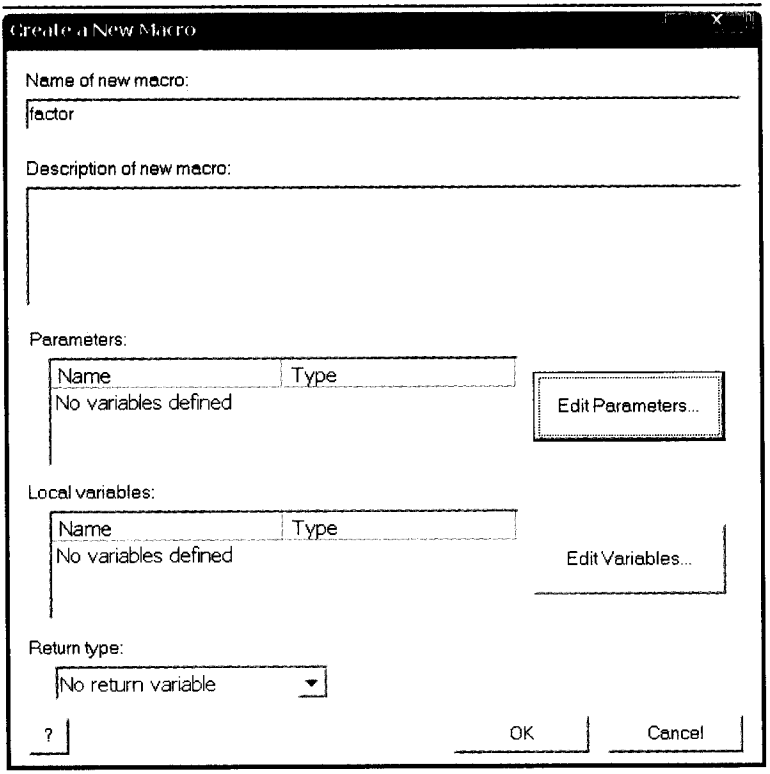


Рис. 2.21. Создание нового макроса

Пока у макроса нет ни параметров, ни локальных переменных, ни возвращаемого значения. Исправим эту ситуацию. Нажмите кнопку **Edit Parameters** (Редактировать параметры) и добавьте один параметр байтового типа с именем `param`. Локальные переменные в нашем макросе не понадобятся, а в качестве типа возвращаемого значения выберите `BYTE`. В результате в рабочей области появится еще одна, пустая диаграмма. Это и есть подпрограмма или макрос. Перетащите мышью с вертикальной линейки инструментов внутрь цикла макрос (восьмой сверху элемент). Теперь рабочая область соответствует рис. 2.22.

Дважды щелкните мышью на элементе **Call macro** в основной диаграмме и в открывшемся диалоговом окне (рис. 2.23) укажите, что в качестве параметра (аргумента) макроса будет использоваться переменная `n`, а в качестве возвращаемого значения — переменная `fac`.

Теперь разместите после вызова макроса элемент **Output**, свяжите его с портом C и задайте выводимое значение переменной `fac`.

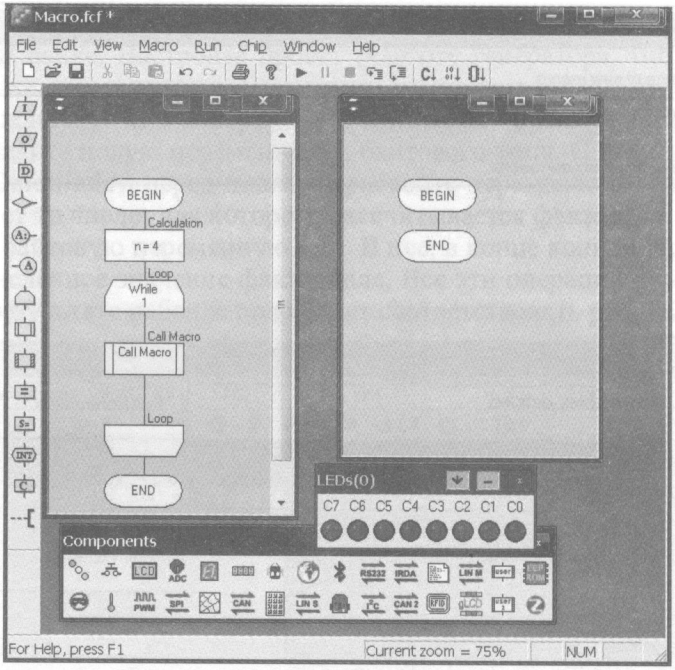


Рис. 2.22. В цикл помещен макрос

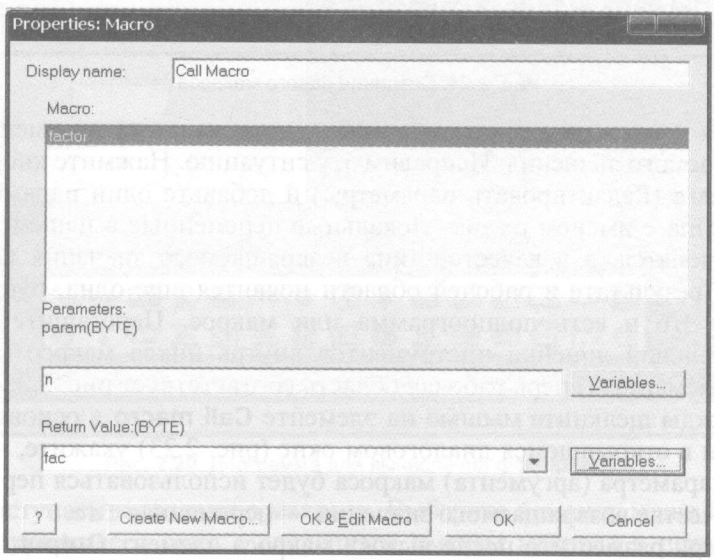


Рис. 2.23. Свойства макроса

В основной диаграмме все готово, однако сам алгоритм макроса пока еще не запрограммирован. Теперь будем работать с диаграммой макроса. Прежде всего отметим, что у внутренних объектов макроса (параметры и возвращаемое значение, а также локальные переменные, если они присутствуют) составные имена. Вначале идет имя макроса `factor`, а затем через разделитель "." — имя. В нашем случае это выглядит как `factor.param` и `factor.Return`. Имя `Return` мы не задавали — оно формируется автоматически.

Для правильной работы алгоритма необходимо, чтобы начальное значение возвращаемой величины было равно 1. Разместите в области между `BEGIN` и `END` диаграммы макроса элемент **Calculation** и определите в нем действие `factor.Return = 1`. Ниже разместите цикл, который выполняется до тех пор, пока параметр макроса больше нуля. В цикле разместите элемент **Calculation**, и создайте в нем два выражения присваивания (рис. 2.24).

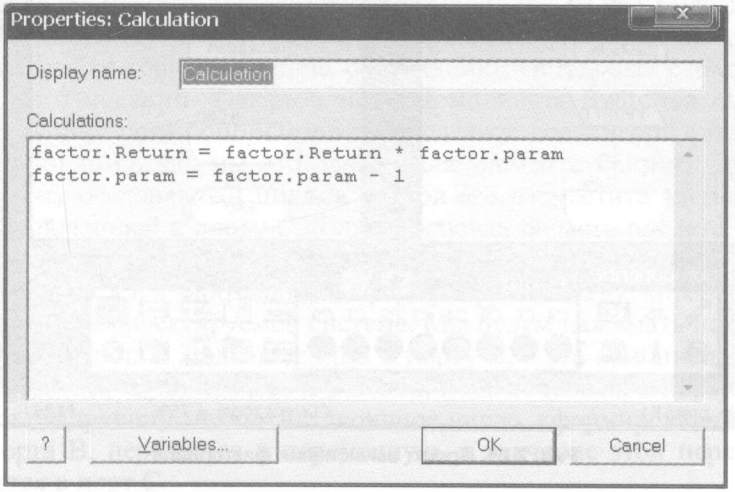


Рис. 2.24. Содержимое элемента **Calculation**

Здесь каждый раз в цикле в переменной `factor.Return` накапливается произведение, а параметр `factor.param` декрементируется, т.е. уменьшается на единицу.

Окончательный вид рабочей области проекта показан на рис. 2.25. Запустив проект на симуляцию, убеждаемся, что на линейке светодиодов высвечивается число 24, представленное в двоичной системе счисления. Попробуем изменить входное значение `n` на 0, просто отредактировав первое вычисление **Calculation** в основной диаграмме. Все пра-

вильно. Особый случай факториала, как и оговорено, дает 1. Теперь можно "прошить" программу на реальном микроконтроллере.

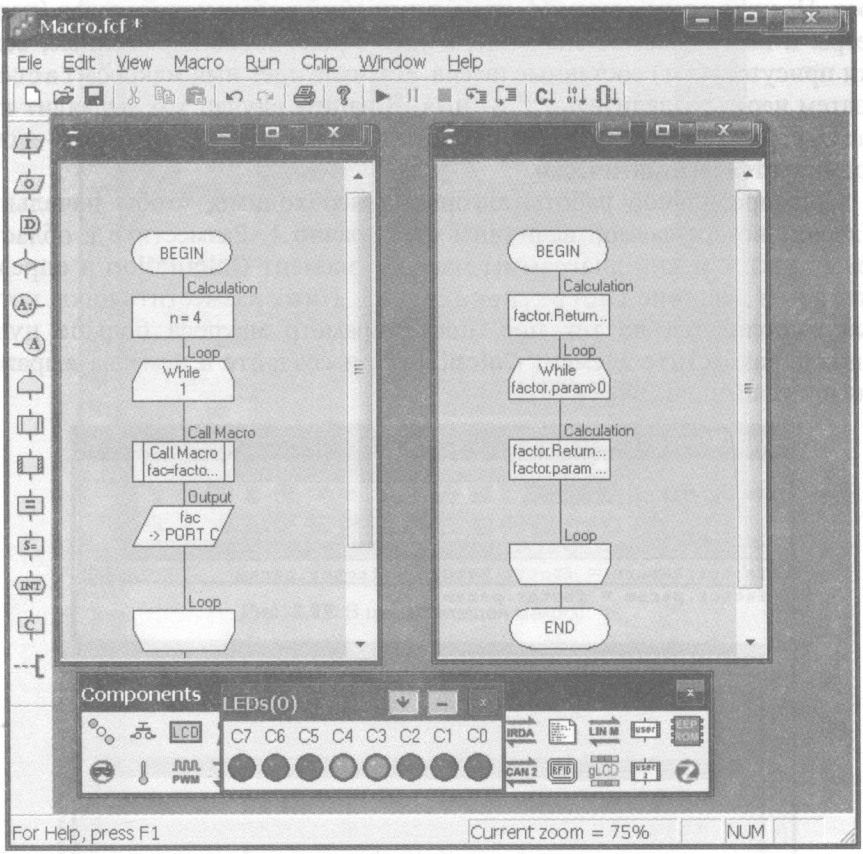


Рис. 2.25. Проект вычисления факториала

Кнопочный ввод

На данном этапе мы научились, пусть и в самой простейшей форме, выводить информацию, сформированную МК во внешний мир. Делаем мы это действительно довольно примитивно: отображаем на светодиодах двоичные числа. В дальнейшем мы, конечно, освоим более удобные способы представления информации, а пока займемся противоположной задачей: вводом данных в МК извне. Это мы также для начала реализуем в простейшем варианте. В режиме эмуляции число будем вводить в

виде двоичного кода (набора нулей и единиц), набираемого на компоненте линейки кнопок. В "железе" это реализуется нажатием комбинации кнопок на плате EasyPIC5, подключенных ко всем разрядам всех портов МК. Имеется ввиду, что при нажатии кнопки на соответствующий вывод МК подается напряжение питания, и соответствующий разряд устанавливается в 1. При отпускании кнопки напряжение снимается, и разряд обнуляется.

Создайте новый проект по тем же правилам, что и предыдущие. Назовем его `knop`. Разместите компонент **LEDs** с восемью светодиодами и подключите его к порту C. Добавьте в рабочую область новый компонент: линейку кнопок под названием **Switches** (второй слева компонент на панели **Components**). По умолчанию эта линейка располагается вертикально, что не очень удобно. Исправим ситуацию. В данном случае добраться до свойств компонента с помощью мыши довольно затруднительно. Лучше это сделать по-другому.

Щелкните мышью в любой точке компонента, чтобы выделить его. Выберите команду меню **Edit ▶ Properties**. В открывшемся окне свойств задайте горизонтальную ориентацию. Остальные свойства оставьте без изменений. Теперь в меню компонента **Switches** выберите команду **Component connections** и определите соединение с портом B (аналогично тому, как мы это делали для элемента **Output**). Добавьте в диаграмму бесконечный цикл, а внутри его разместите элемент **Output**, подключенный к порту C. Теперь рабочая область проекта должна соответствовать рис. 2.26.

Теперь, хотя и с запозданием, сформулируем задачу, которую поручим выполнять проектируемой системе. Мы будем нажимать некоторую комбинацию кнопок, подключенных к порту B, и эта комбинация должна отобразиться на линейке светодиодов, подключенных к порту C. Реализуется следующий механизм: двоичное число, сформированное кнопками порта B, передается в переменную, а значение этой переменной передается в порт C.

Итак, нам понадобится переменная и инструмент ввода. Перетащите мышью с вертикальной панели инструментов элемент **Input** (самый верхний). Разместите его сразу после начала цикла. Откройте окно его свойств и привяжите ввод к порту B. Здесь же, нажав кнопку **Variables**, создайте новую переменную `s` (рис. 2.27).

Окончательный вид готового к исполнению проекта представлен на рис. 2.28.

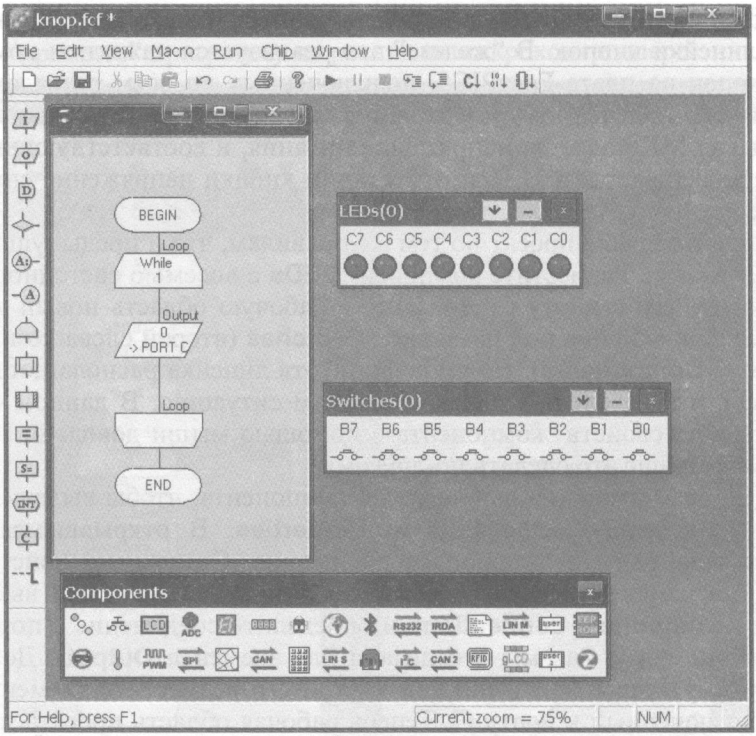


Рис. 2.26. Проект кнопочного ввода

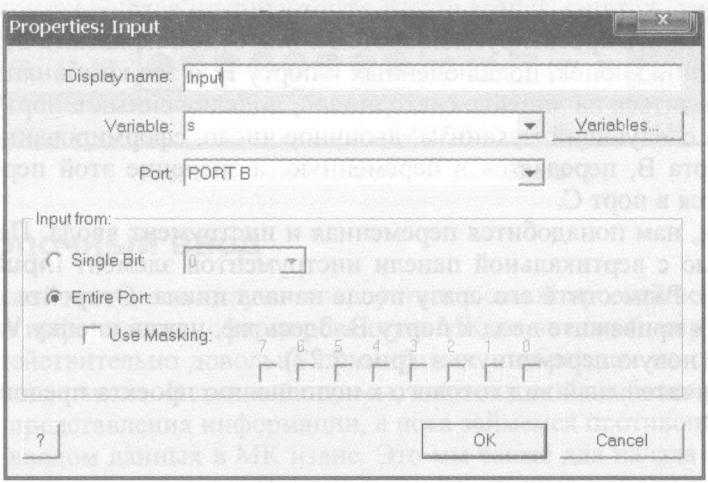


Рис. 2.27. Свойства элемента **Input**

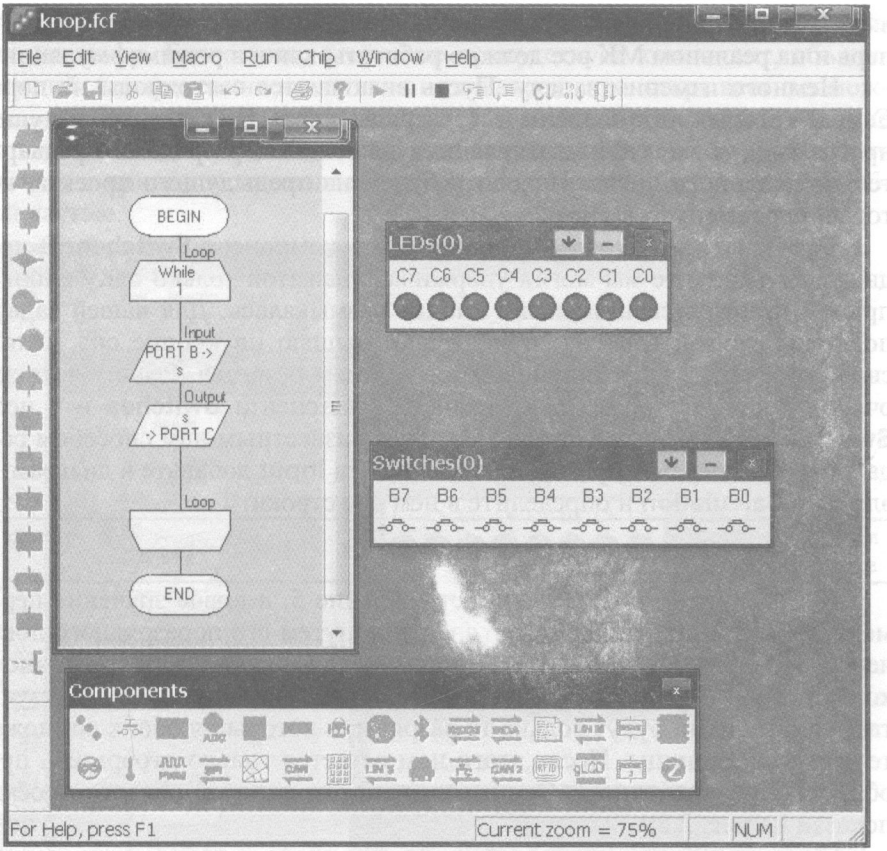


Рис. 2.28. Проект кнопочного ввода

Запустите программу на эмуляцию, и понажимайте мышью кнопки на компоненте **Switch**. Если все было сделано правильно, то в линейке **LEDs** будут загораться соответствующие лампочки. А вот при запуске программы на реальном микроконтроллере может обнаружиться сюрприз. Скомпилируйте проект, чтобы получить HEX-файл, занесите программу в память МК и подайте на него питание. Скорее всего, окажется, что все светодиоды порта С светятся, хотя ни одна из кнопок на порте В еще нажата не была. Эта ситуация связана с упомянутыми в предыдущей главе подтягивающими резисторами. В нашем случае входы порта В "болтаются в воздухе" или, по крайней мере, не согласованы с входными устройствами: кнопками. К счастью проблема решается очень просто. Достаточно на плате EasyPIC5 установить переключатель J2

в состояние "PullDown", т.е. замкнуть средний и нижний контакты. Теперь и на реальном МК все должно работать, как и в режиме эмуляции.

Немного изменим задачу. Пусть включаются светодиоды на порте C, для которых установлены в 1 разряды порта B (как в предыдущей программе), и эти единицы являются двоичными разрядами предварительно заданного числа. Исходить будем из предыдущего проекта, который переименуем в `knop2`.

Для начала необходимо видоизменить компонент **Switches**. В предыдущем варианте мы могли удерживать нажатой только одну кнопку, причем при отпускании мыши кнопка размыкалась. Для нашей задачи подходит другой вариант: при щелчке мышью на кнопке она меняет свое состояние на противоположное и в нем остается. Реализовать это очень просто. Откройте окно свойств компонента **Switches** и в поле **Switch Type** выберите **Toggle**. Далее, уже известным нам способом создайте новую переменную `m`. После элемента **Input** добавьте в диаграмму элемент **Calculation** и определите в нем две строки:

```
m = 5
s = s AND m
```

Переменная `m` здесь принимает значение 5, а новое значение переменной `s` получается из старого значения путем его поразрядного логического умножения на `m`. Что это означает? Каждый разряд двоичного кода числа `s` умножается на соответствующий разряд числа `m`. В результате единичными будут только те разряды, в которых у обоих сомножителей были единицы. После этого нам остается только отобразить преобразованное значение `s`. Окончательный вид рабочей области проекта показан на рис. 2.29.

Поскольку в качестве маски мы задали число `m`, равное 5 (двоичный код 101), соответствующие светодиоды будут загораться, только если нажата кнопка 0 или 2 или обе эти кнопки. На нажатие остальных кнопок реакции не будет. Заметим также, что реализовывать данный проект на плате EasyPIC5 не имеет смысла, поскольку в ней отсутствуют кнопки переключателей с фиксированным состоянием.

Семисегментный индикатор

Среди множества компонентов в системе Flowcode присутствует так называемый семисегментный индикатор. Этот элемент хорошо нам знаком по почтовым конвертам и открыткам, на которых мы пишем почтовый индекс в специальных шаблонах: каждая цифра в своем шаблоне. Аналогичный шаблон, собранный из светящихся сегментов, и представляет собой индикатор. Таким образом мы получаем средство

отображения информации, а именно, любой из десяти (от 0 до 9) десятичных цифр. Более того, в системе присутствует компонент, объединяющий четыре семисегментных индикатора, что открывает возможность отображения четырехзначных целых чисел. Кроме того, в отличие от почтового шаблона, индикатор снабжен точкой в правом нижнем углу, что, в принципе, позволяет отображать и числа с десятичной дробной частью.

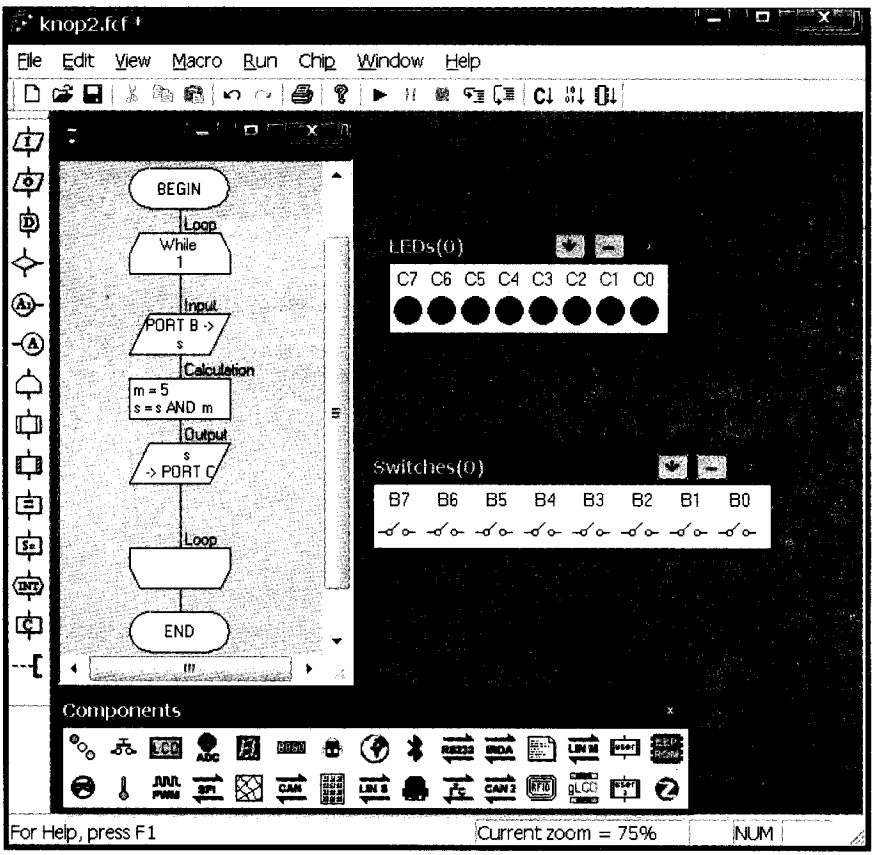


Рис. 2.29. Видоизмененный проект кнопочного ввода

Спроектируем систему, которая в бесконечном цикле отображает последовательно с некоторой выдержкой на индикаторе все десятичные цифры. Создайте проект с именем `7seg`. Определите две байтовые переменные `i` и `j`. Создайте бесконечный цикл. В цикле с помощью элемента **Calculation** присвойте этим переменным значения 0 и 1 соответ-

ственно. Создайте внутренний цикл, выполняющийся до тех пор, пока i<10. В нем разместите элемент **Calculation** с инкрементированием переменной i и организуйте задержку длительностью 500 мс. Разместите в рабочей области компонент **LED7Seg**. Полученный проект должен соответствовать рис. 2.30.

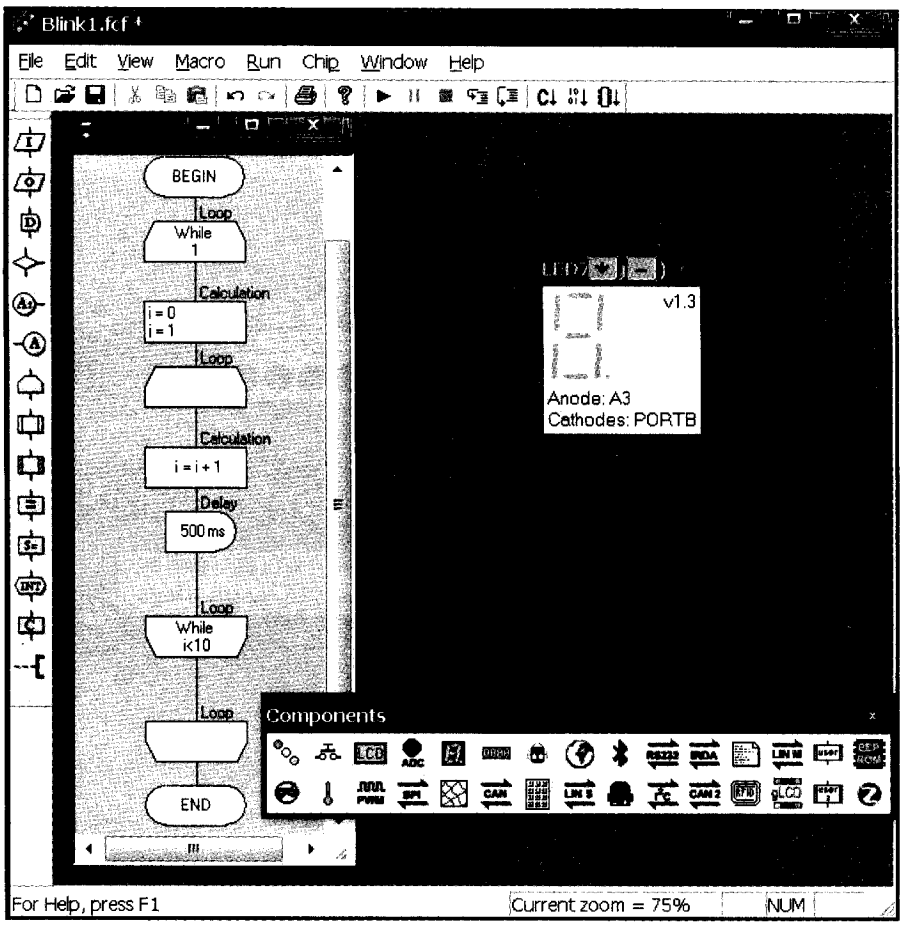


Рис. 2.30. Проект с семисегментным индикатором

Для своей работы индикатор использует два порта микроконтроллера. По умолчанию это порты A и B. Для совместимости с работой на плате EasyPIC5 нам потребуется изменить умолчания. Необходимо подключить компонент не к порту B, а к порту D. Для этого окно **Component connections** приводим к виду, показанному на рис. 2.31.

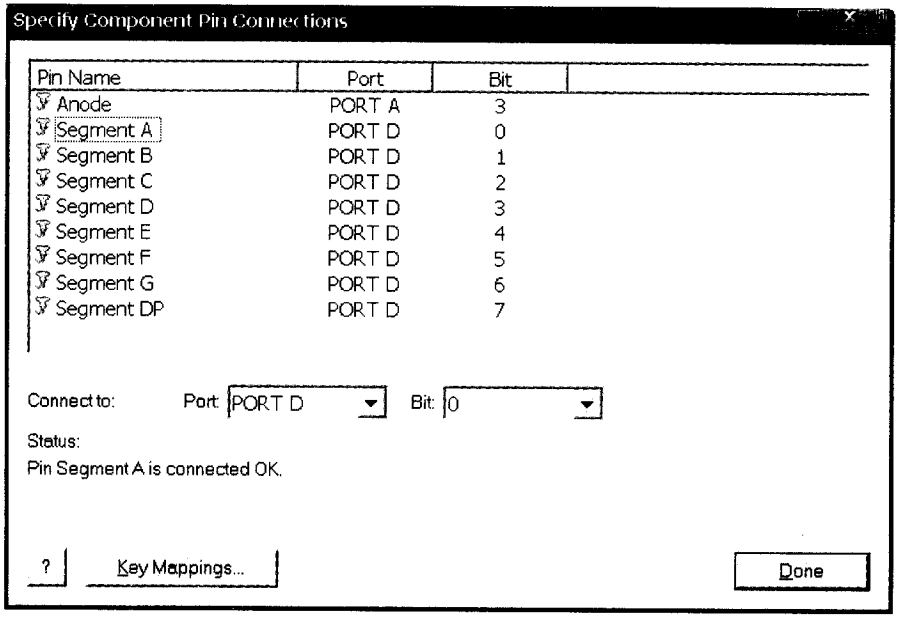


Рис. 2.31. Организация подключений для семисегментного индикатора

У многих компонентов есть собственные макросы (один или несколько), обеспечивающие те или иные действия. В отличие от пользовательских, их называют компонентными и вызывают с помощью девятого сверху элемента вертикальной линейки. Разместите компонентный макрос в начале внутреннего цикла и дважды щелкните на нем мышью, чтобы открыть окно его свойств.

Поскольку компонент у нас один, в списке **Component** будет указано только **LED7Seg1(0)**. Выделите этот элемент. Справа, в поле **Macro** отобразится имя единственного компонентного макроса нашего объекта: `ShowDigit`. Выделите его. В поле **Parameters** появится список параметров (аргументов) макроса. Что это за параметры, можно узнать из справки, вызываемой с помощью кнопки **?** в левом нижнем углу окна.

У данного макроса — два параметра. Первый — байтовая переменная, значение которой отобразится на индикаторе, а второй — переменная или значение, управляющее отображением десятичной точки. В поле **Variables** эти параметры следует выбрать или ввести, используя в качестве разделителя запятую. У нас будет отображаться переменная `i`, значение которой последовательно изменяется от 0 до 9. Переменная `j` равна 1, что определяет отображение десятичной точки. В результате все должно соответствовать рис. 2.32.

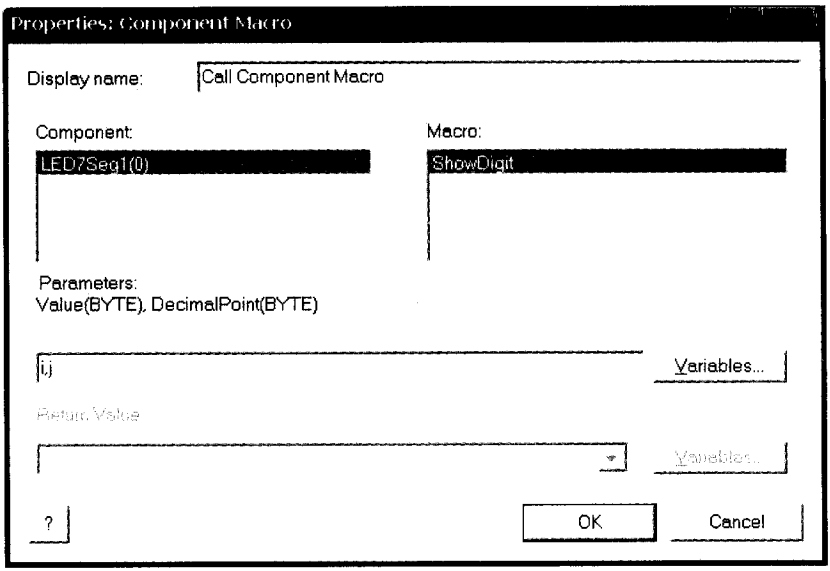


Рис. 2.32. Свойства компонентного макроса

Конечный вид рабочей области проекта перед запуском на эмуляцию показан на рис. 2.33.

С эмуляцией в Flowcode проблем возникнуть не должно. Что касается фактической реализации на МК, на плате необходимо включить переключатель DIS0 (или все четыре DIS0–DIS3) на SW6. Этим мы подключаем к порту D семисегментные индикаторы. А теперь — странность №1... Необходимо установить в состояние "PullDown" порт B, хотя непонятно, при чем здесь этот порт. Очевидно, все дело в некоторой несовместимости конфигурации MCU, заданной в Flowcode, со спецификой платы EasyPIC5.

Странность №2... Система работает, но с точностью до наоборот. Те сегменты индикатора, которые должны светиться, отключены, и наоборот. Но это объяснимо. Индикатор может включаться по схеме с общим анодом или с общим катодом. Очевидно, в Flowcode предполагается один вариант, а на плате — другой. Если бы у нас была возможность отредактировать компонентный макрос, то мы могли бы исправить ситуацию, но такой возможности нет. Приходится довольствоваться тем, что есть.

Как уже упоминалось ранее, в наличии — компонент и четырех семисегментных индикаторов. Попробуем отобразить на нем четырехзначное десятичное число (например, 2345). Подключение четырех ин-

дикаторов к портам мало чем отличается от подключения одного индикатора. Что же касается параметров, то у данного компонента их не два, а три. Первый параметр — номер индикатора в четверке (0, 1, 2 или 3), а остальные — прежние. Окно подключений показано на рис 2.34.

Разместите компонент с четырьмя индикаторами в новом проекте `47seg` и определите для него подключения согласно рис. 2.34. Нам понадобятся четыре байтовых переменных `i`, `j`, `k`, `r` и одна переменная `s` типа `int` (число 2345). Еще до начала бесконечного цикла необходимо создать элемент **Calculation** с весьма интересной арифметикой. Дело в том, что в байтовые переменные требуется записать цифры, составляющие наше четырехзначное число, — задача совсем не тривиальная.

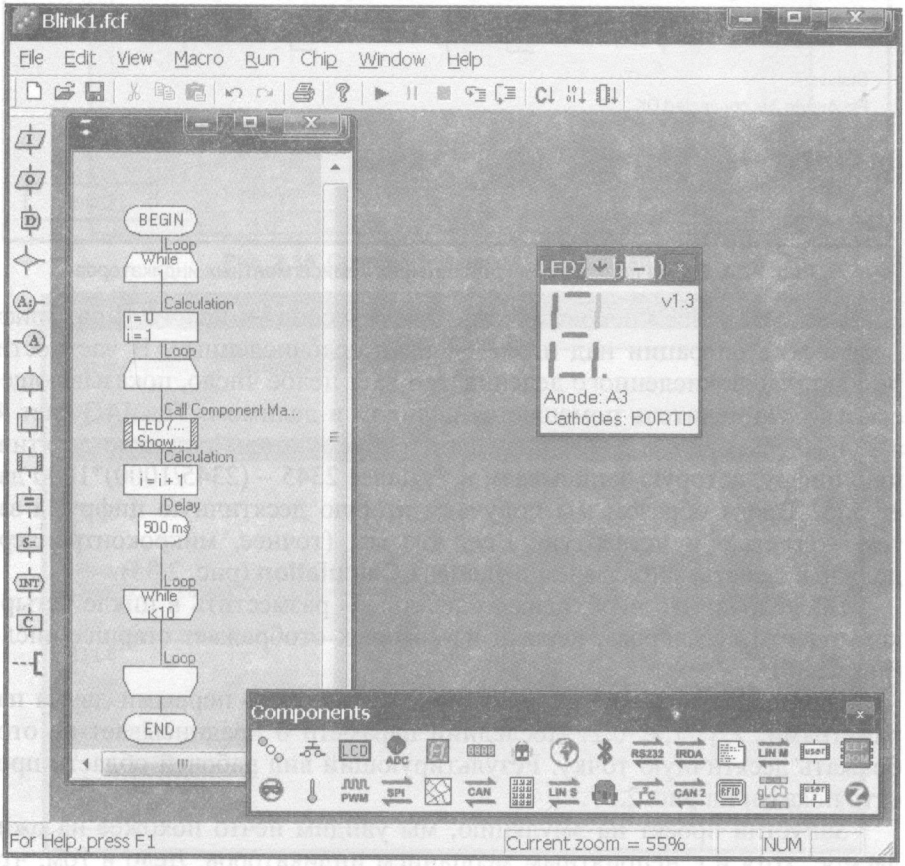


Рис. 2.33. Завершенный проект для семисегментного индикатора

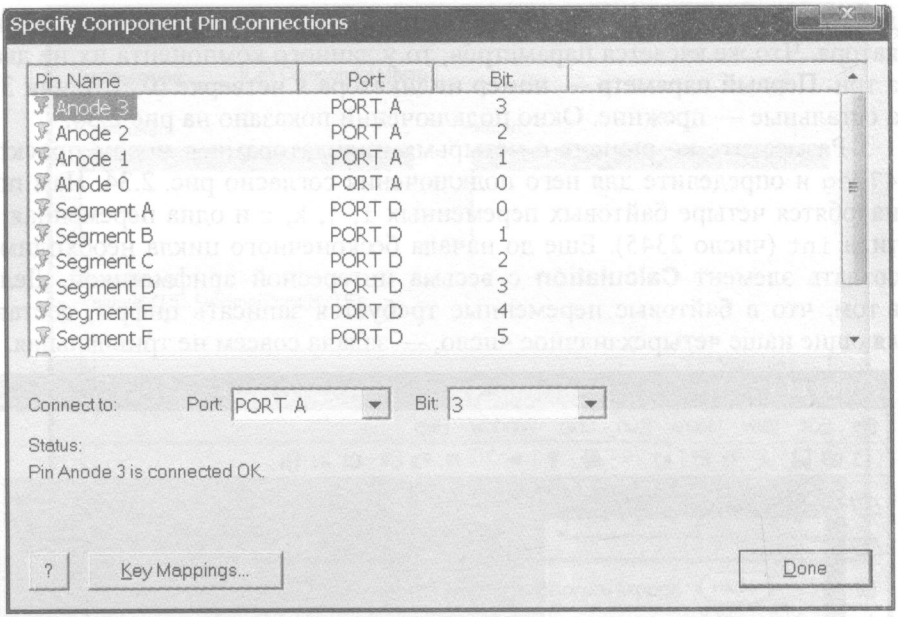


Рис. 2.34. Схема подключений для четырех семисегментных индикаторов

Поскольку все числовые данные в Flowcode — целого типа, арифметические операции над ними — тоже целочисленные. В частности, результат целочисленного деления `a/b` дает целое число, показывающее сколько раз делитель помещается целиком в делимом. Так 14/3 дает 4, а 2345/1000 дает 2, т.е. мы получаем требуемую нам старшую десятичную цифру, которую записываем в `i`. Далее 2345 – (2345/1000)*1000 дает 345. Таким образом мы получаем вторую десятичную цифру, а затем — третью и четвертую. Все, что мы (точнее, микроконтроллер) должны сделать, записываем в элемент **Calculation** (рис. 2.35).

Последнее, что необходимо сделать, — разместить в цикле четыре компонентных макроса, первый из которых отображает старшее число (рис. 2.36).

Остальные макросы будут отличаться от этого первыми двумя параметрами: `2,j; 1,k; 0,r`. Последний параметр 0 предписывает не отображать десятичную точку. Результирующий вид рабочей области проекта показан на рис. 2.37.

Запустив проект на эмуляцию, мы увидим нечто похожее на ожидаемое, хотя и с неприятным мерцанием индикаторов. Дело в том, что на самом деле индикаторы здесь включаются и отключаются поочередно, а не одновременно. Реализовать проект в "железе" даже не будем

пытаться, поскольку результат получится совсем неприглядным. Однако огорчаться не стоит, поскольку далее мы переходим к гораздо более привлекательному представлению выходной информации.

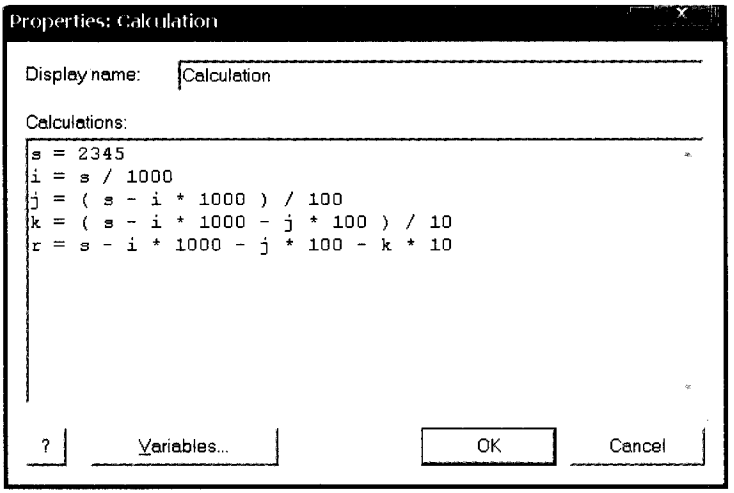


Рис. 2.35. Содержимое элемента **Calculation**

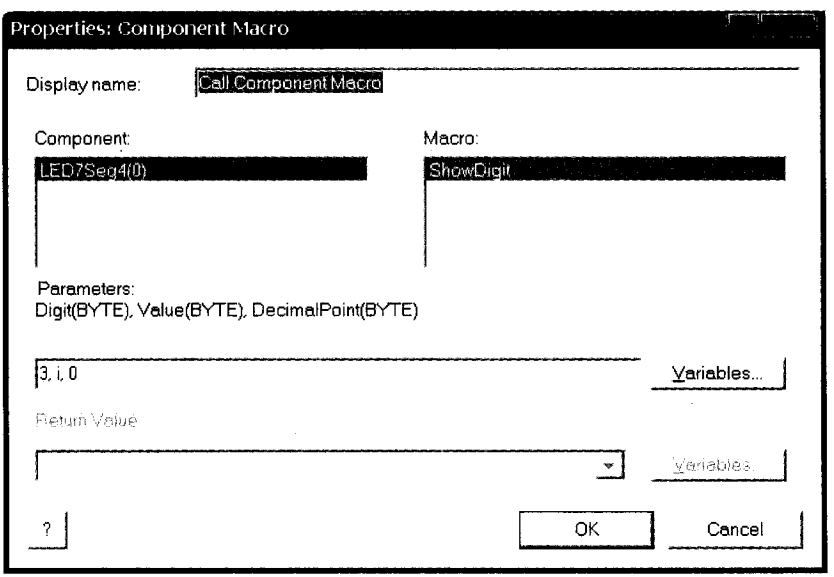


Рис. 2.36. Содержимое первого компонентного макроса

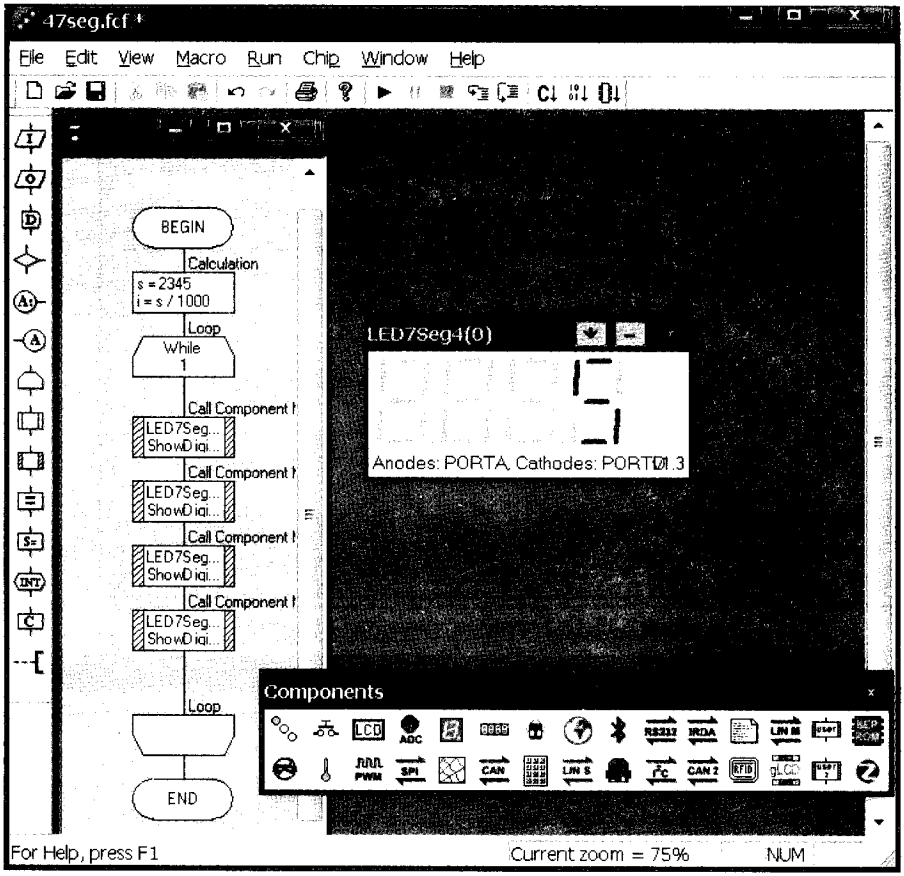


Рис. 2.37. Проект для четырех семисегментных индикаторов

ЖК-дисплей

Удобным средством отображения информации является жидкокристаллический (ЖК) дисплей. Как в Flowcode, так и на плате EasyPIC5 используется двухстрочный дисплей, в каждой из строк которого может отображаться до 16 алфавитно-цифровых символов.

Проект `lcd`, который мы сейчас разработаем, во всех учебных курсах должен бы быть первым, поскольку речь идет о программе, которая здоровается с пользователем. То есть, при запуске микроконтроллера на дисплее должна появиться надпись: “Hello World!”. Примем, что она будет начинаться с первой позиции первой строки.

Во избежание мерцаний дисплея, структуру диаграммы проекта сформируем следующим образом. Вначале выполним все необходимые действия, а затее организуем бесконечный пустой цикл. Попутно отметим, что в Flowcode пустые циклы, вообще говоря, не обязательны. Если все действия достаточно выполнить однократно, то их можно просто разместить между `BEGIN` и `END`. В таком случае, отработав один раз, Flowcode сообщит о том, что эмуляция завершена. Для "живого" микроконтроллера такой режим не считается штатным (хотя, в принципе, допустим), поэтому мы программы всегда зацикливаем.

Прежде всего, необходимо создать переменную `STRING` строкового типа и с помощью элемента **String Manipulation** (четвертый снизу в вертикальной линейке) присвоить строке значение `Hello World!`. Элемент **String Manipulation** напоминает **Calculation** и потому в комментариях не нуждается (рис. 2.38).

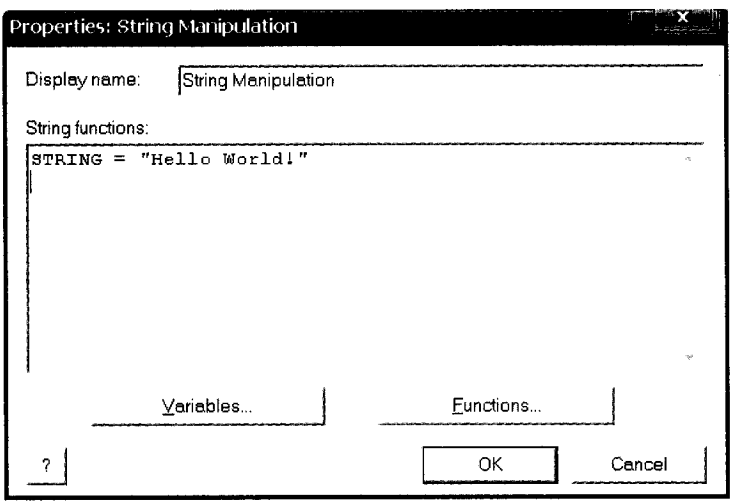


Рис. 2.38. Свойства элемента **String Manipulation**

Разместите в рабочей области компонент **LCDDisplay** (третья слева кнопка панели **Components**). Изменять свойства его подключения не требуется, поскольку как в Flowcode, так и на плате EasyPIC5 он подключается к порту B. Дисплей снабжен несколькими собственными компонентными макросами. Нам понадобятся два из них. Разместите два макроса сразу после инициализации строки. В первом элементе вызовем макрос без параметров `Start`. Он просто активизирует дисплей. Вторым разместим макрос `PrintString`. Его параметр — имя строковой переменной (рис. 2.39).

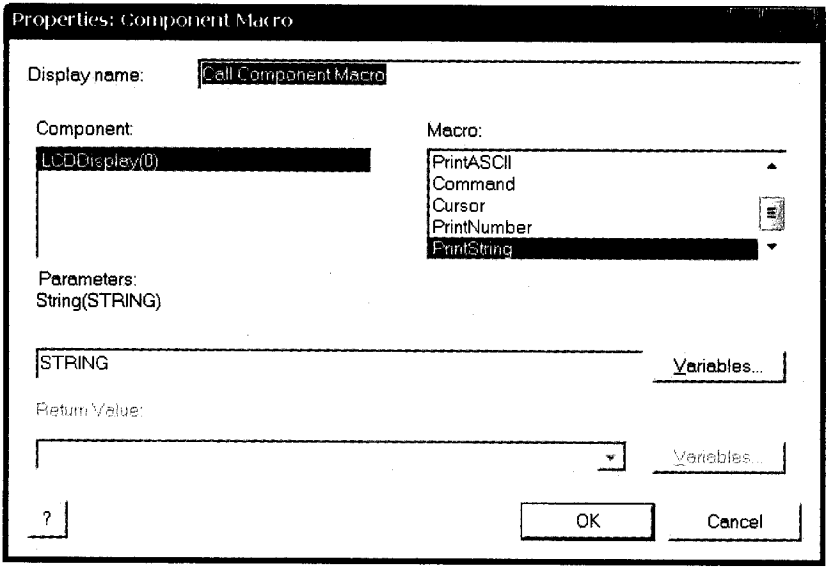


Рис. 2.39. Свойства макроса `PrintString`

Теперь все готово. Окно проекта показано на рис. 2.40. Запустите проект на эмуляцию, чтобы убедиться, что все происходит, как планировалось. Без каких-либо проблем задача реализуется и на EasyPIC5. Не забудьте только включить сам дисплей на плате с помощью переключателя LCD на SW9.

Над строками в системе можно производить различные действия: объединение, инвертирование и т.д. Они называются строковыми функциями и доступны через элемент **String Manipulation**. В окне свойств этого элемента присутствует кнопка **Functions**, предоставляющая доступ к нескольким функциям (их описание можно найти в справке).

Модифицируем наш проект (назовем его `lcd2`) таким образом, чтобы во второй строке дисплея отображалось количество символов, выведенных в верхней строке. Для этого определим новую байтовую переменную `len`.

После первого компонентного макроса вызовем еще один. Это будет макрос `Cursor` с параметрами `0,1` (переход в начальную позицию второй строки дисплея). Далее разместим элемент **String Manipulation** и с помощью кнопки **Functions** приведем его к виду, показанному на рис. 2.41.

Проект готов! Убеждаемся в том, что на дисплее отображается требуемая нам информация (рис. 2.42).

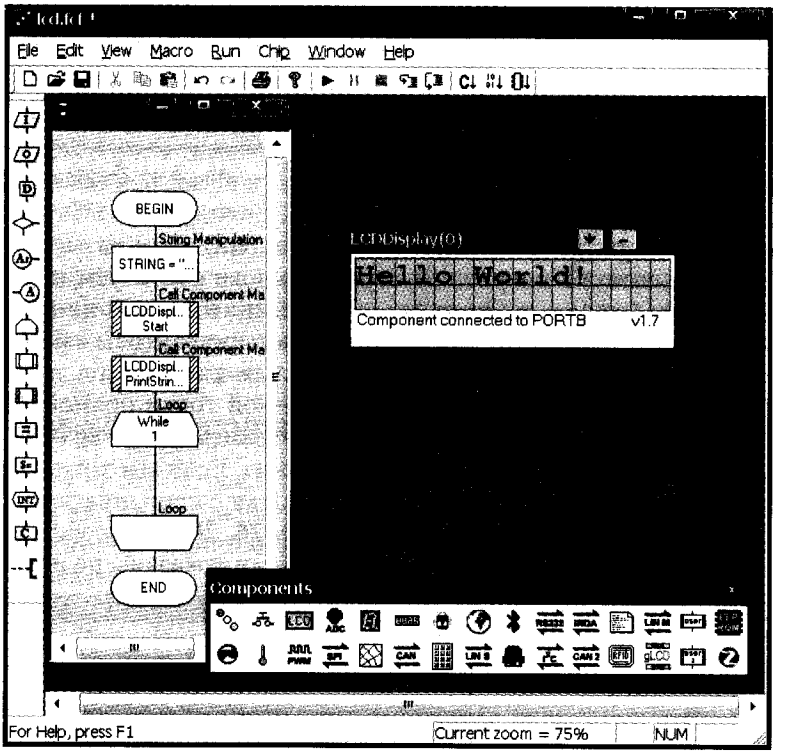


Рис. 2.40. Проект для работы с ЖК-дисплеем

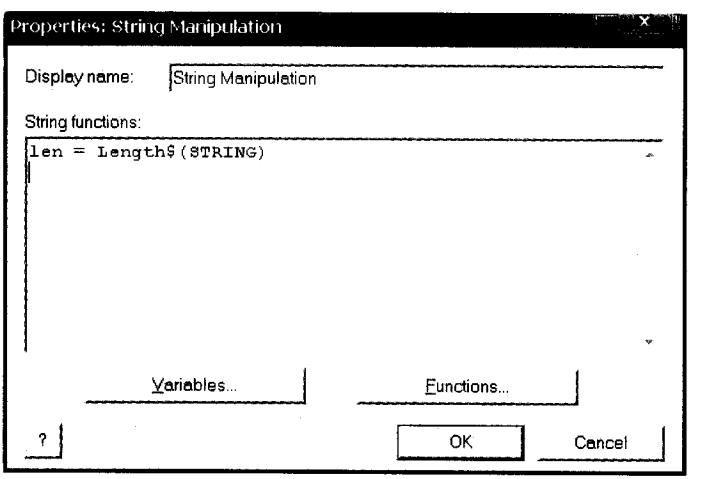


Рис. 2.41. Содержимое элемента **String Manipulation**

При необходимости можно заставить микроконтроллерную систему выводить информацию на графический ЖК-дисплей и в графическом виде. В новом проекте GLCD разместите в рабочей области компонент **GLCD**. Настройки подключения и свойства объекта менять не будем.

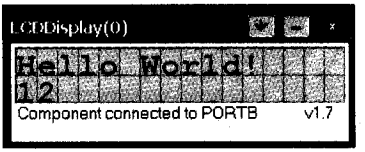


Рис. 2.42. Отображение информации на ЖК-дисплее

На диаграмме опять создайте пустой цикл, а перед ним разместите два вызова компонентного макроса. В первом вызове реализуем инициализацию дисплея (Init), а во втором — проведение прямой линии на экране (DrawLine). У макроса DrawLine — четыре байтовых параметра. Два первых — горизонтальная и вертикальная координаты начальной точки, а два последних — координаты конечной точки прямой. Начало системы координат располагается в левом верхнем углу экрана. Ось X направлена слева направо, ось Y — сверху вниз. Если мы хотим провести прямую из начала системы координат в точку с координатами 20, 20, то в поле **Variables** необходимо указать **0, 0, 20, 20**. Вот, собственно, и все, что требуется сделать. Результаты эмуляции показаны на рис. 2.43.

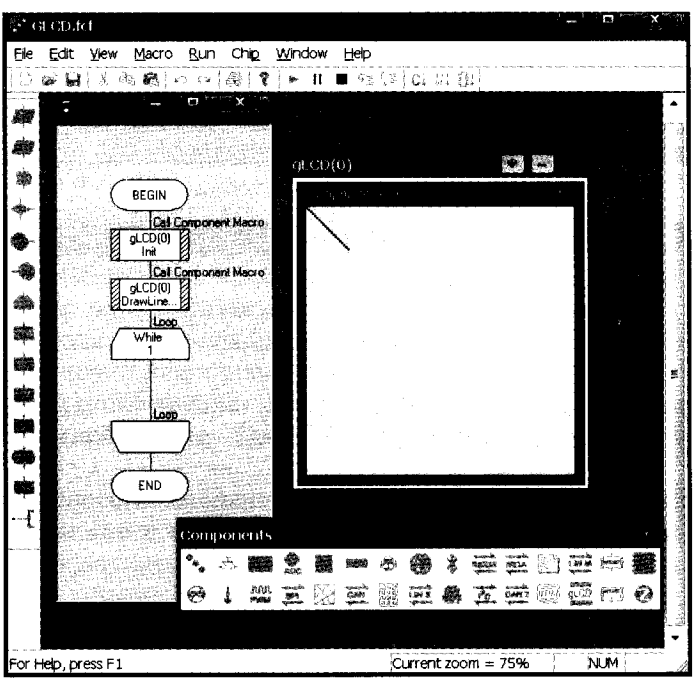


Рис. 2.43. Рисование с помощью графического ЖК-дисплея

Компонент **GLCD** содержит целый ряд настроек, с которыми интересно поэкспериментировать. Графический дисплей снабжен довольно большим количеством собственных макросов. Проведение прямой линии — лишь один из простейших. Попробуйте самостоятельно поработать с другими функциями графики.

К сожалению, графический дисплей поддерживаемый в Flowcode, совершенно не совместим с устройством, которое подключается к плате EasyPIC5. В связи с этим не стоит даже пытаться реализовать данный проект на плате. Впрочем, не стоит и печалиться. Практическое применение графического ЖК-дисплея мы рассмотрим в следующей главе. На этом основные средства отображения информации в системе Flowcode будем считать изученными.

Мини-клавиатура 4x3

Настало время познакомиться с более развитыми по сравнению с кнопочным вводом средствами передачи информации из внешнего мира в МК. Достаточно типично для этих целей применение мини-клавиатуры (так называемый Keypad), похожей на ту, которая используется в простых кнопочных телефонах. В Flowcode доступен компонент, эмулирующий 12-кнопочную клавиатуру (четыре строки на три столбца), на которой можно набрать все десятичные цифры и два специальных символа: * и #. Компонент называется **KeyPad**. Он сопровождается двумя компонентными макросами, одним из которых мы и воспользуемся. Макрос называется `GetKeypadNumber`. Его назначение — считать числовое значение цифры, нажатой на клавиатуре, и передать его в заданную переменную. Числовое значение для клавиши * — 10, а для клавиши # — 12.

На наш взгляд, организацию макроса можно оценена по шкале от "не очень удобно" до "очень неудобно", поэтому придется сделать довольно пространные объяснения. Если после первого запуска макроса с его помощью в цикле опрашивается клавиатура, то до момента нажатия какой-либо клавиши возвращаемое значение (`ReturnValue`) равняется 255. После нажатия кнопки возвращаемое значение становится требуемым числом, которое сохраняется и после отпускания кнопки (за исключением особых случаев). Условия возникновения особых случаев, когда `ReturnValue` возвращается в 255, нами не установлены, однако их все же придется учесть при разработке следующего проекта.

Попробуем сделать программу, которую можно назвать простейшим калькулятором, перемножающим две десятичные цифры (именно однозначные цифры, а не числа). Он должен функционировать следую-

щим образом. На клавиатуре нажимаем цифру первого сомножителя, затем — знак умножения, и, наконец, — второй сомножитель. Все это вместе с результатом умножения отображается на ЖК-дисплее. Индикация задерживается на одну секунду, после чего дисплей очищается, и можно повторить ввод.

Создайте проект KEY. В рабочей области разместите ЖК-дисплей и клавиатуру **KeyPad**. Оба компонента по умолчанию подключаются к порту B. Во избежание конфликтов настроим клавиатуру на работу с портом C. Создайте три байтовых переменных: i, j и s. В первой и второй сохраняются сомножители, считанные с клавиатуры, а в третьей — формируется результат умножения.

Первым действием на диаграмме будет инициализация дисплея, после чего следует бесконечный цикл. В начале цикла очищается дисплей. Далее создайте внутренний цикл, в котором в переменную i записывается значение с клавиатуры. В соответствии с предыдущим абзацем цикл повторяется до тех пор, пока i равно 255, т.е. пока не будет нажата одна из кнопок. Макрос опроса клавиатуры прост и комментироваться не будет. Для того чтобы разобраться с ним, у читателя уже достаточно опыта. После выхода из внутреннего цикла первый сомножитель отображается на дисплее. Фрагмент диаграммы до данного этапа показан на рис. 2.44 слева.

Теперь программа ожидает нажатия на клавиатуре кнопки *. Поскольку соответствующее числовое значение равно 10, организуем цикл, который повторяется до тех пор, пока в переменную j не будет считано значение 10. В условии While необходимо указать j!=10, что означает "j не равно 10" (заимствовано из языка программирования C). Когда требуемая кнопка будет нажата, произойдет выход из цикла, и на дисплее отобразится символ *. Для этого вызываем макрос дисплея PrintString, которому в качестве параметра передается строка из одного символа "*".

Далее необходимо организовать цикл считывания в переменную j второго сомножителя. Войдем мы в этот цикл с j, равным 10, — наследство от предыдущего действия. Как мы уже говорили, в следующем цикле, пока не нажата кнопка, переменная j или остается равной 10, или сбрасывается в 255. Таким образом, цикл работает до тех пор, пока выполняется условие (j==10) OR (j=255). OR — это операция логического сложения "ИЛИ". При нажатии цифровой кнопки данное условие нарушится, цикл прервется с требуемым значением второго сомножителя в j. Описанный фрагмент диаграммы показан на рис. 2.44 по центру.

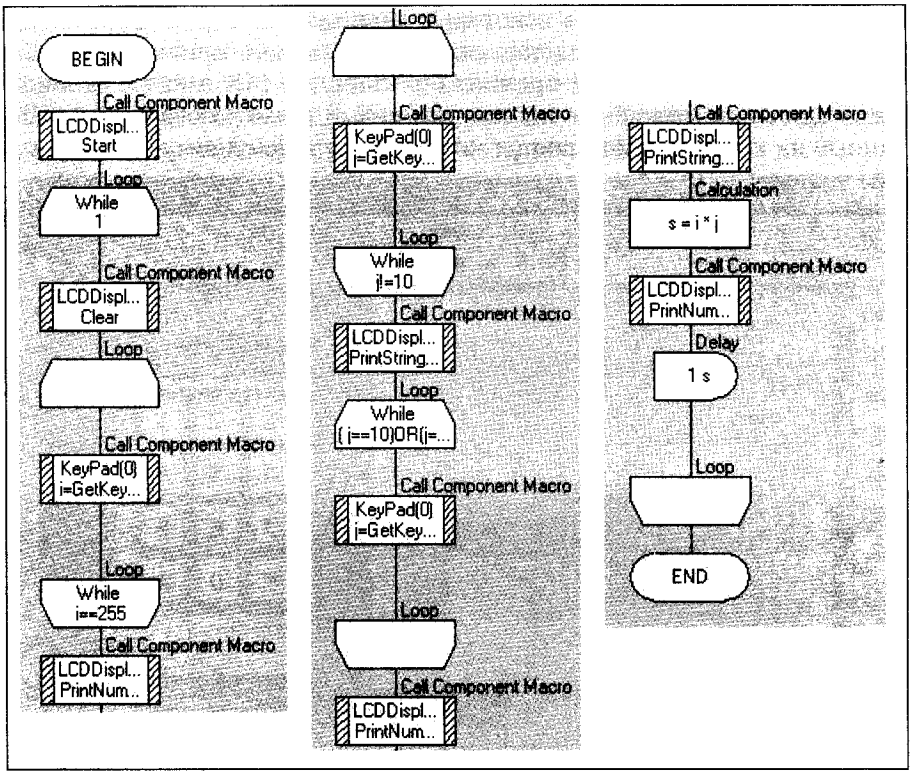


Рис. 2.44. Диаграмма, реализующая простой калькулятор

Теперь на дисплей выводится символ "=". После этого в элементе **Calculation** выполняется умножение `i` на `j` с занесением результата в переменную `s`, значение которой отображается на дисплее. Следует задержка длительностью в одну секунду и переход в начало основного цикла. Окончание диаграммы показано на рис. 2.44 справа. Проект (рис. 2.45) готов к запуску на эмуляцию.

В распоряжении автора не было реальной 12-кнопочной клавиатуры, поэтому о судьбе проекта в случае его реализации на плате EasyPIC5 ничего неизвестно.

АЦП

Микроконтроллеры широко применяются в сфере обработки информации от различных датчиков. Как правило, такая информация поступает в аналоговом виде как электрический ток или напряжение, из-

меняющиеся непрерывно в некотором диапазоне значений. Дальнейшая обработка, передача и представление информации обычно реализованы в цифровом виде. По этой причине большинство МК имеет в своем составе один или несколько аналого-цифровых преобразователей (АЦП), данные на которые поступают, соответственно, по одному или нескольким каналам.

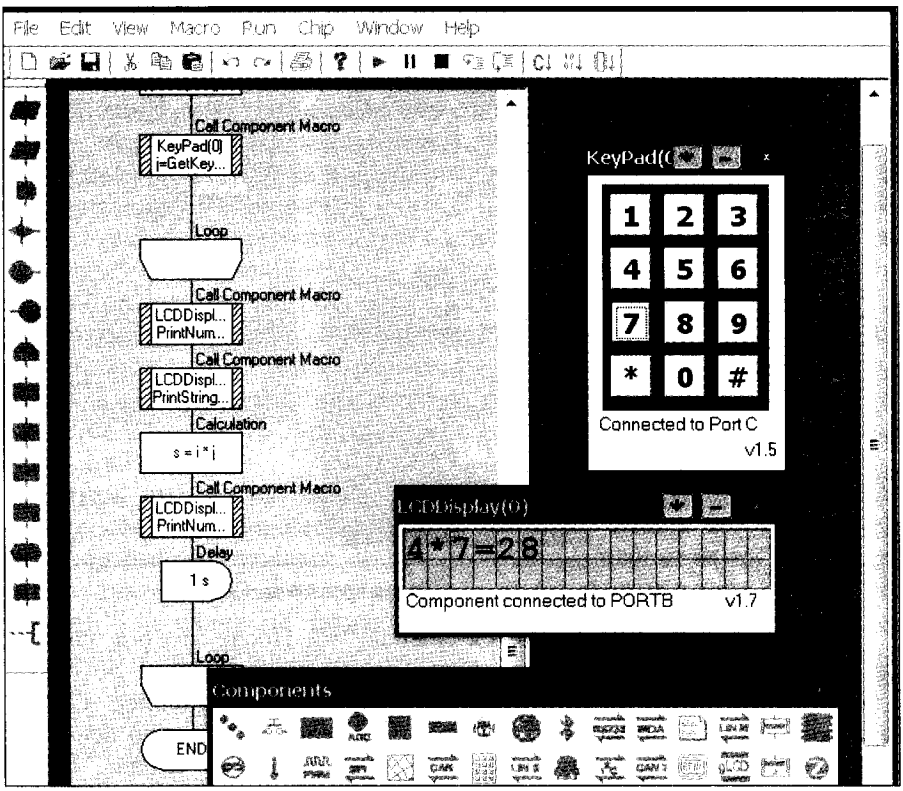


Рис. 2.45. Калькулятор в действии

Обычно аналоговый сигнал подается на один из выводов какого-либо порта. Различные устройства обеспечивают оцифровку с различной разрядностью. Типичными для МК являются 8-, 10- и 12-разрядные АЦП. При этом исходный диапазон изменения входного сигнала разбивается на 256, 1 024 и 4 096 градаций, каждая из которых представляется соответствующим целым числом.

Создайте новый проект ADC, и разместите в рабочей области ЖК-дисплей и компонент **ADC**. Для последнего измените установленное по

умолчанию значение свойства **Type** на **Slider**, поскольку бегунком удобнее управлять с помощью мыши. Выбрав в меню компонента **ADC** команду **Component connection**, можно убедиться, что АЦП считается подключенным к каналу 0. Это нас устраивает. Теперь диаграмму необходимо привести к виду, показанному на рис. 2.46.

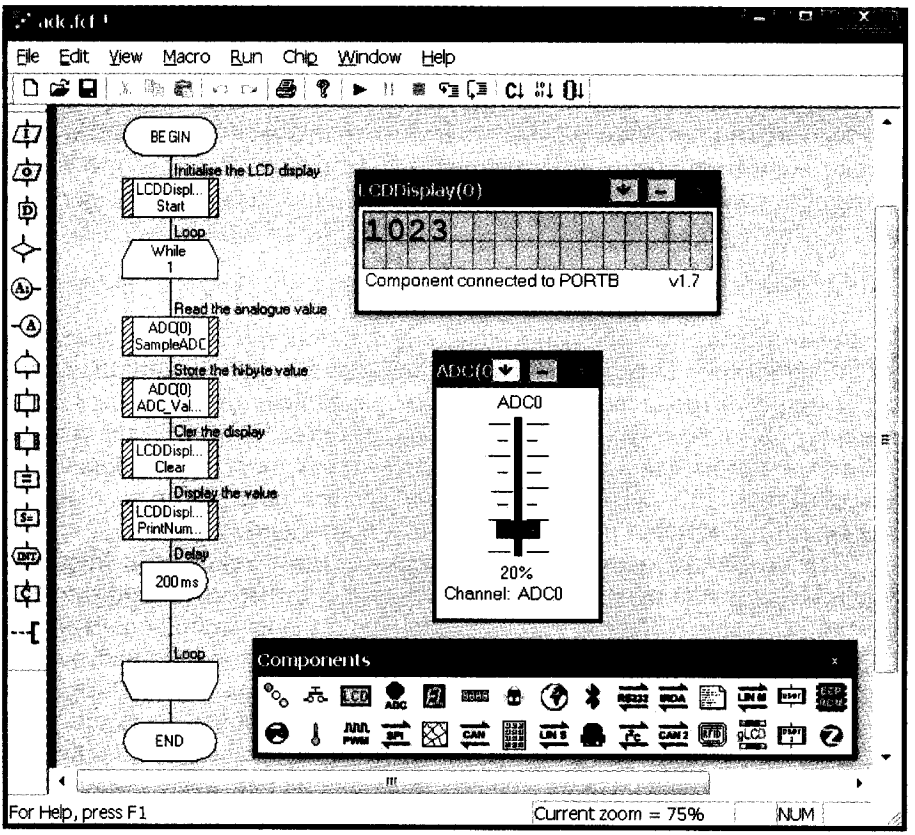


Рис. 2.46. Проект АЦП

Вначале инициализируется дисплей, после чего организуется бесконечный цикл. Первым в цикле вызывается компонентный макрос для **ADC** под названием `SampleADC`. У него нет ни параметров, ни возвращаемого значения. Назначение этого макроса — одиночный опрос значения входного сигнала. В нашем проекте АЦП выполняет 10-разрядную оцифровку, поэтому для представления результата нам понадобится переменная `ADC_Value` типа `INT`.

Следующий макрос на диаграмме (ReadAsInt) передает результат оцифровки в переменную ADC_Value. Далее следует макрос очистки дисплея от предыдущего отображения. После этого дисплейный макрос PrintNumber выводит значение ADC_Value на экран. Последний элемент в цикле (**Delay**) обеспечивает выдержку экспозиции в 200 мс, после чего цикл повторяется.

Запустите проект на эмуляцию. Перетаскивая бегунок вверх-вниз, можно видеть, что оцифрованное значение сигнала изменяется в пределах от 0 до 1 023.

Проект без проблем реализуется и на плате EasyPIC5. Необходимо только в соответствии с инструкцией удостовериться в том, что подтягивающие резисторы не подключены к задействованным аналоговым входам порта A.

Обмен данными с внешними устройствами

Несомненно, важнейшая задача для микропроцессорных систем — обмен данными с внешними устройствами по коммуникационным каналам. Микроконтроллеры поддерживают различные интерфейсы как напрямую, так и через дополнительные устройства. В этой главе мы рассмотрим один из наиболее распространенных и достаточно простых интерфейсов: RS232. В частности, он позволяет обмениваться данными между МК и ПК через последовательный порт COM.

Новый проект (назовем его RS232) передает текстовую информацию в ПК, где она отображается на мониторе. Разместите в рабочей области компонент **RS232**. Он не подразумевает каких-либо подключений, а в окне его свойств необходимо убедиться, что интерфейс настроен на передачу символов (**Character**). Альтернативой может быть передача байтов (**Byte**). Кроме того, в свойствах можно настроить скорость обмена данными, однако мы оставим значение по умолчанию (9 600 бод). Полученная рабочая область проекта показана на рис. 2.47.

Компонент **RS232** содержит три дисплея, среди которых нас интересует первый, отображающий передаваемые микроконтроллером символы. Сформулируем задачу таким образом. Система должна с частотой пять раз в секунду отсылать на компьютер строку “Hello PC!”, где это приветствие должно отображаться каждый раз в новой строке, причем — с начала строки.

Прежде всего, нам понадобится строковая переменная STR, которая с помощью элемента **String manipulation** заполняется текстом “Hello PC!\n\r”. Что означает !\n\r? Это последовательность двух специальных символов, первый из которых предписывает перейти на новую

строку, а второй — перейти в начало текущей строки. Хотя, эта запись занимает четыре позиции, она представляет именно два символа, так что общая длина строки составляет 11 символов. Рассматривая строку как массив символов, следует помнить, что нумерация его элементов соответствует синтаксису языка С. Это значит, что индекс (порядковый номер) начального элемента массива — 0. Так, `STR[0]` — это буква "H". Следовательно, индекс последнего элемента — 10, т.е. `STR[1]` соответствует символу `\r`.

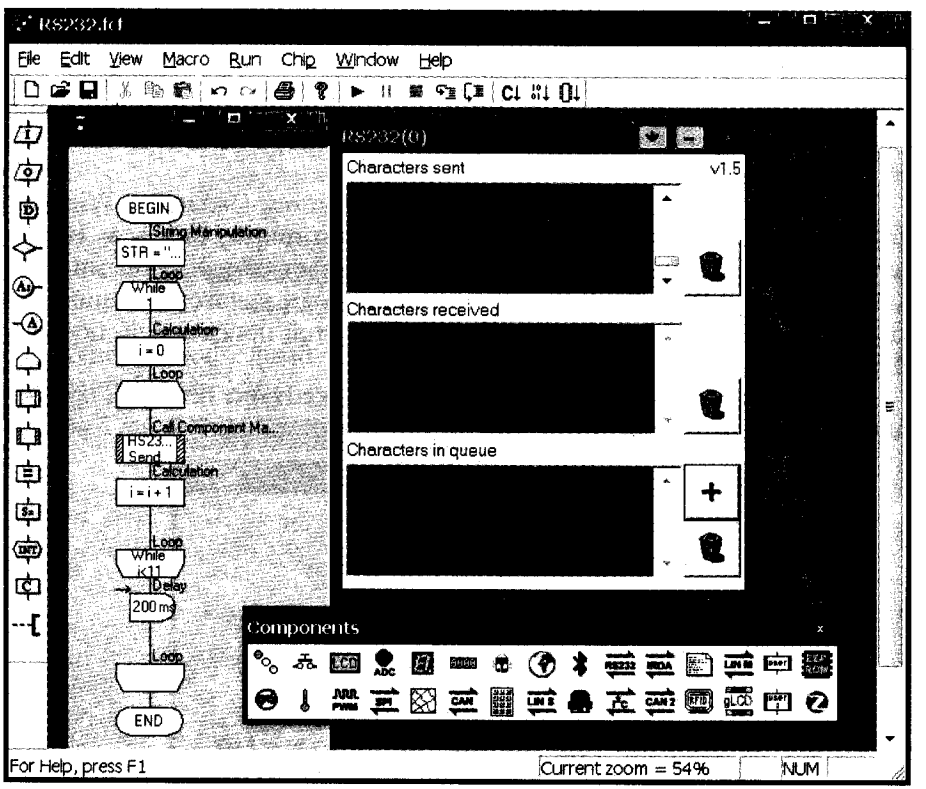


Рис. 2.47. Проект для интерфейса RS232

Нам также понадобится байтовая переменная `i`. В начале бесконечного цикла она устанавливается в 0, после чего организуется внутренний цикл. Во внутреннем цикле вызывается компонентный макрос `SendRS232Char`, который выдает в интерфейс RS232 символ, являющийся параметром макроса. Поскольку мы хотим отправить очередной символ строки, параметром будет `STR[i]`. Таким образом, переменная `i`

является счетчиком элементов массива. Очевидно, что счетчик в цикле должен увеличиваться на единицу, что и реализует элемент **Calculation**. Поскольку мы организовали цикл с проверкой условия в конце, условие должно иметь вид `i<11`. Как только условие нарушится, внутренний цикл прервется, и система начнет передавать строку заново. При этом во внешнем цикле установлена задержка длительностью 200 мс.

На этом этапе можно запустить эмуляцию, однако ничего интересного, кроме отображения символов в первом дисплее компонента **RS232**, мы не увидим. Скомпилируйте HEX-файл и загрузите его в микроконтроллер на плате EasyPIC5. Перед этим плату необходимо соединить с ПК так называемым нуль-модемным кабелем (кабель COM-COM). Кроме того, согласно инструкции по EasyPIC5, для вывода данных из платы необходимо включить первый переключатель на SW8. После этого систему можно запускать в работу.

Теперь переходим на ПК. Выберите в системном меню Windows команду **Пуск ► Все программы ► Стандартные ► Связь ► HyperTerminal**. Программа HyperTerminal обеспечивает обмен данными между ПК и внешними устройствами по соединениям некоторых типов. Рядовой пользователь вряд ли близко знаком с этой программой, поэтому рассмотрим ее чуть подробнее.

Первое окно, которое откроется при запуске HyperTerminal, предназначено для присвоения имени создаваемому соединению (рис. 2.48).

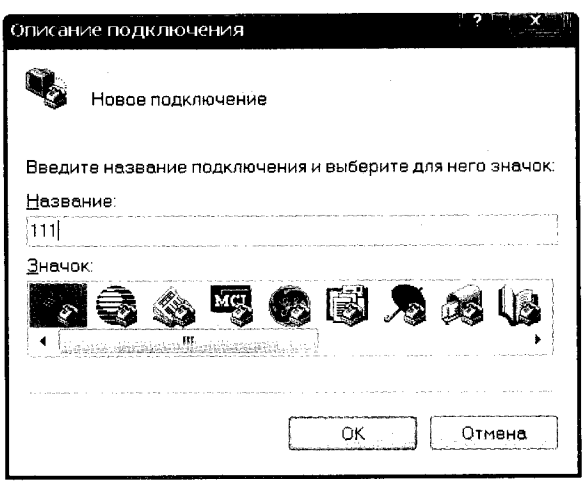


Рис. 2.48. Создание нового соединения в программе HyperTerminal

Введите произвольное имя и выберите для соединения пиктограмму на свой вкус.

Далее необходимо установить тип подключения. Если кабель подключен к первому последовательному порту, то окно **Подключение** должно соответствовать рис. 2.49.

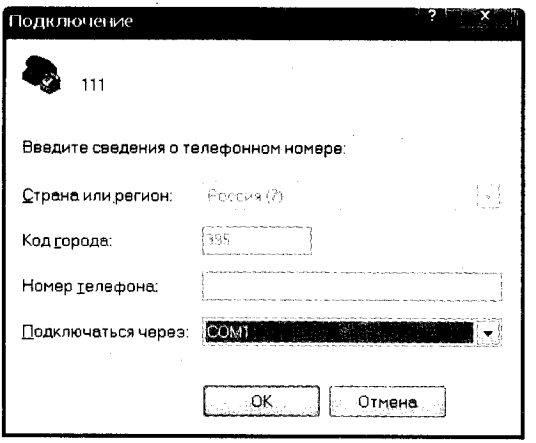


Рис. 2.49. Подключение через порт COM1

Теперь необходимо настроить параметры порта. Окно настройки показано на рис. 2.50.

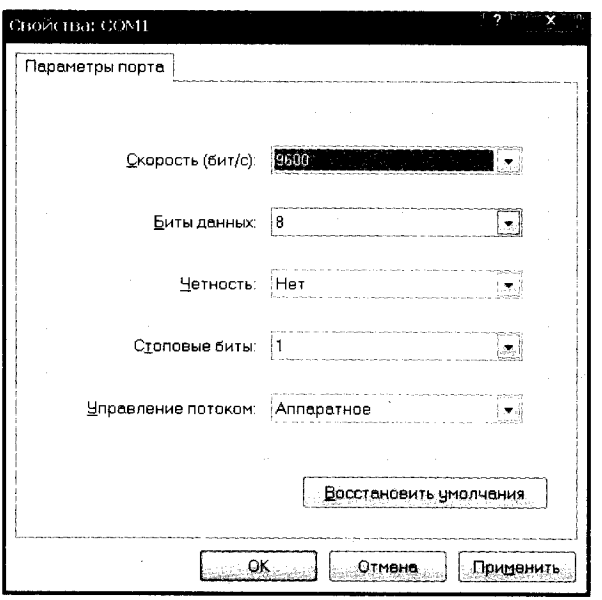


Рис. 2.50. Настройки порта COM1

Мы выставили скорость передачи данных на МК 9 600 бод. Это же значение необходимо выбрать и в первом раскрывающемся списке окна **Свойства: COM1**. По нажатию кнопки **ОК** откроется рабочее окно терминала (рис. 2.51).

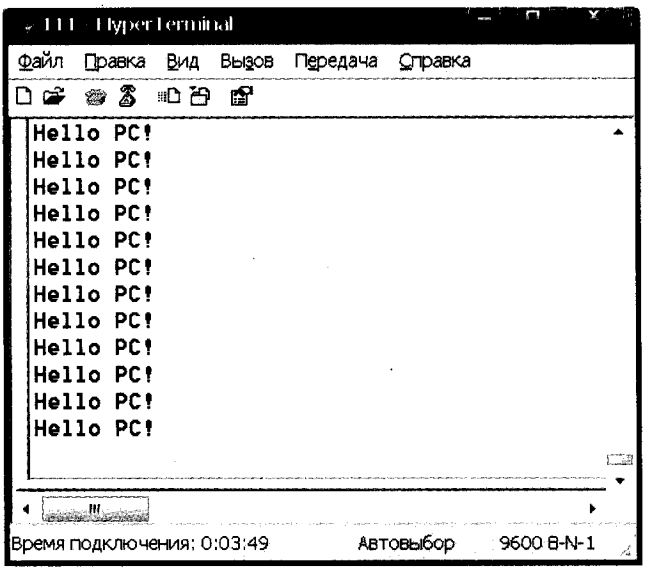


Рис. 2.51. Рабочее окно терминала

Конечно, хотелось бы наладить и обратную связь для передачи данных от ПК к МК, но, к сожалению, в Flowcode этого сделать не удастся, поскольку реального аналога компоненту RS232 на EasyPIC5 нет. Оставим эту задачу на будущее, а точнее — до следующей главы.

Прерывания

Прерыванием называется приостановка выполнения программы вследствие наступления некоторого события. Реакцией со стороны МК на это событие является выполнение заранее подготовленного макроса, после чего естественный порядок исполнения программы восстанавливается. Мы проиллюстрируем два типичных примера прерываний: от таймера микроконтроллера и по появлению сигнала на определенном выводе порта.

Обычно микроконтроллеры снабжены несколькими таймерами, но в любом случае в устройстве присутствует таймер TMR0, с которым мы и будем работать. Он функционирует независимо от выполнения про-

граммы, и его работа контролируется исключительно тактовым генератором MCU. По сути дела, все, что происходит в таймере, — это инкрементирование (увеличение на единицу) содержимого буфера памяти таймера на каждом цикле (четырех тактах) генератора. Рано или поздно наступает переполнение этого буфера, что и является событием, порождающим прерывание от таймера.

Создайте новый проект `tmr0`. Он будет просто отсчитывать секунды, прошедшие с момента запуска или перезапуска системы. Количество секунд будем отображать на ЖК-дисплее, поэтому разместите в рабочей области соответствующий компонент.

Первым делом в диаграмме запустим дисплей. Определите две байтовые переменные `int_count` и `sec_count` для хранения текущего количества произошедших переполнений таймера и числа прошедших секунд. Первоначально обнулим эти переменные. Попутно отметим, что, как можно будет видеть из будущей диаграммы, оба обнуления в виде двух операций присваивания можно объединить в одном элементе **Calculation**.

Перетащите мышью в диаграмму с вертикальной линейки новый для нас элемент **Interrupt**, позволяющий использовать прерывания. Его свойства показаны рис. 2.52.

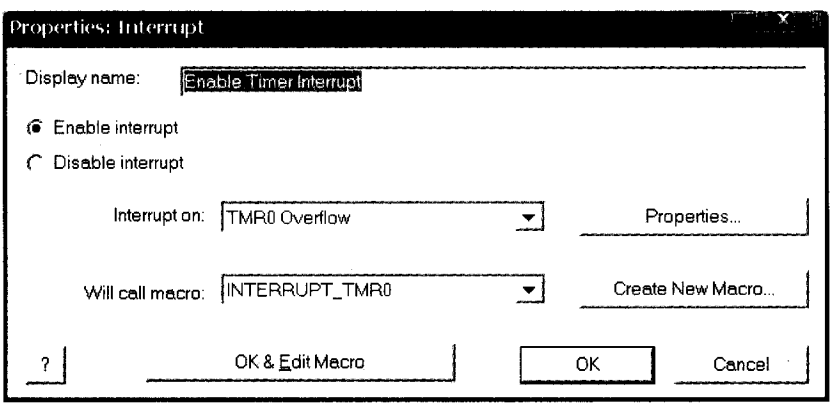


Рис. 2.52. Свойства элемента **Interrupt**

В раскрывающемся списке **Interrupt on** (Прерывание по) выберите элемент **TMR0 Overflow** (Переполнению таймера 0). Поле **Will call macro** (Будет вызывать макрос) по умолчанию пустое, поскольку у нас пока нет макроса обработки прерывания. Создадим такой обработчик, нажав кнопку **Create New Macro**. Отроется окно, показанное на рис. 2.53.

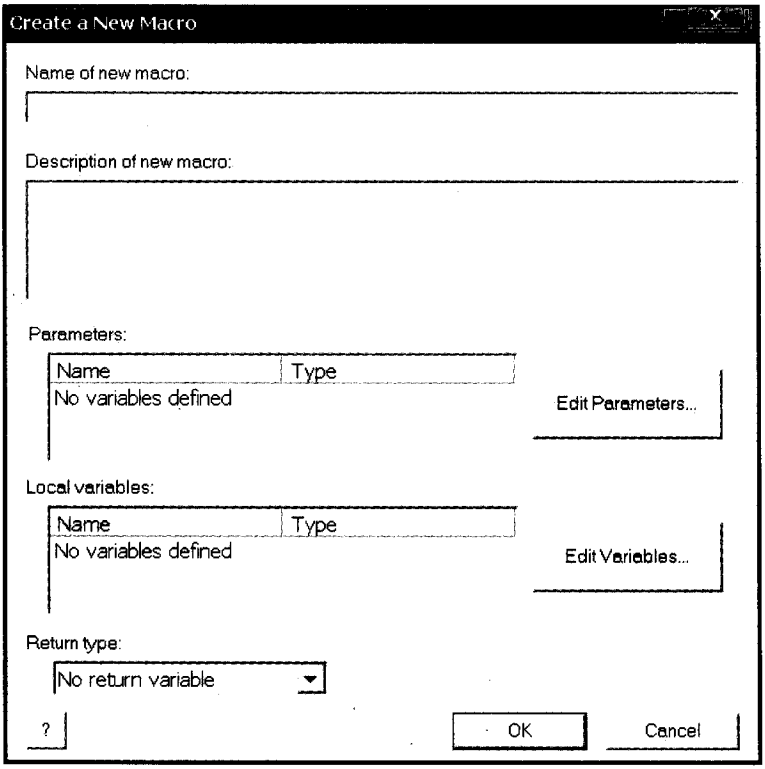


Рис. 2.53. Диалоговое окно **Create a New Macro**

Здесь нам понадобится ввести только имя нового макроса (допустим, `INTERRUPT_TMR0`), а затем выбрать его в предыдущем окне в раскрывающемся списке **Will call macro**. В данном случае наш макрос-обработчик не имеет ни параметров, ни локальных переменных, ни возвращаемого значения. В других же пользовательских макросах такие объекты могут понадобиться.

Итак, новый макрос мы создали. Как его просмотреть? Выбрав в списке, который открывается через пункт меню **Macro ▶ Show**. Визуализировав макрос, мы увидим, что он, естественно, пока пуст. Просто в рабочей области появится еще одна диаграмма `BEGIN-END`. В нее необходимо обычным образом вставить элемент **Calculation**, увеличивающий на единицу значение переменной `int_count`. Другими словами, реакцией на переполнение таймера будет приращение счетчика переполнений. Диаграмма макроса показана на рис. 2.54.

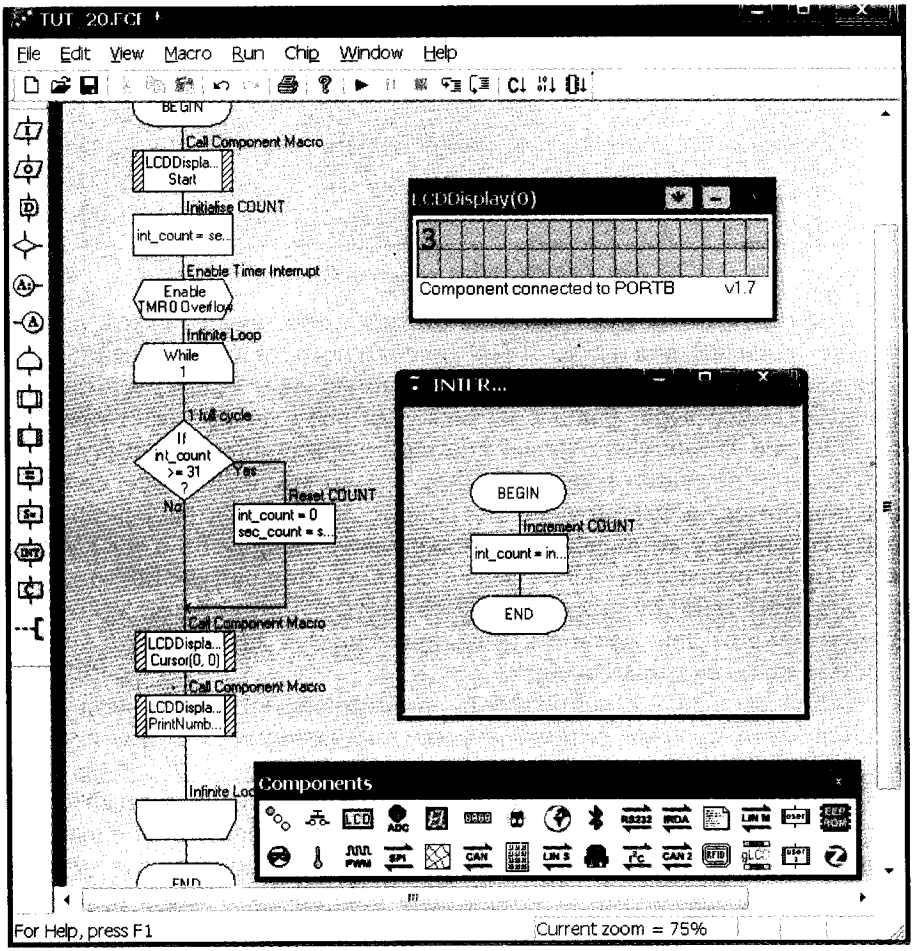


Рис. 2.54. Диаграмма макроса `INTERRUPT_TMR0`

Теперь возвращаемся в основную диаграмму и организовываем бесконечный цикл. В него поместите с вертикальной линейки новый элемент **Decision** — разветвитель, позволяющий направлять алгоритм по одному из альтернативных путей в зависимости от выполнения или не выполнения условия. Свойства элемента **Decision** должны соответствовать рис. 2.55. В нашем случае по ветке "Yes" программа последует, если переменная `int_count` станет больше 30, а по ветке "No" — при всех остальных значениях.

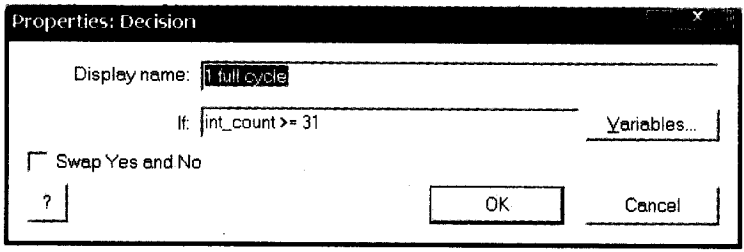


Рис. 2.55. Свойства элемента **Decision**

В ветке "No" не будет вообще никаких действий (следуем по программе дальше), а в ветке "Yes" необходимо обнулить переменную `int_count` и инкрементировать переменную `sec_count`. Что все это значит? Таймер через равные промежутки времени вызывает прерывания. Каждое из них увеличивает счетчик прерываний на единицу до тех пор, пока он не станет равным 32. Как только это произойдет, сам счетчик обнулится, а переменная `sec_count` увеличится на единицу. На плате EasyPIC5 с установленным микроконтроллером PIC16F877A таймер 0 будет переполняться 32 раза в секунду (при симуляции это будет не так). Нетрудно понять, что в таком случае значение `sec_count` инкрементируется два раза в секунду (примерно), а значит `sec_count` — действительно счетчик секунд.

Осталось только в конце цикла организовать визуализацию счетчика. Для этого вначале верните курсор в начальную позицию дисплея, после чего введите числовое значение.

Теперь рассмотрим прерывания по сигналу на порт. Создайте проект `rb0`, подсчитывающий число нажатий кнопки, связанной с выводом 0 порта B. Результат будем отображать в двоичном виде в четырех младших разрядах порта C с помощью светодиодов. Здесь нам почти все знакомо, так что основную конфигурацию проекта можно понять из рис. 2.56.

В отличие от предыдущего проекта, в элементе **Interrupt** в качестве источника прерывания необходимо выбрать **RB0/INT**. Макрос обработки прерывания должен увеличивать на единицу счетчик `COUNT`. При запуске проекта на плате EasyPIC5 не забудьте, что вывод 0 порта B является входом и, следовательно, его подтягивающий регистр следует подключить в состоянии "PullDown". Кроме того, во избежание известного эффекта дребезга контакта, нажатие на кнопку RB0 должно быть кратким и энергичным. Только при таком условии счетчик будет выполнять свои функции правильно.

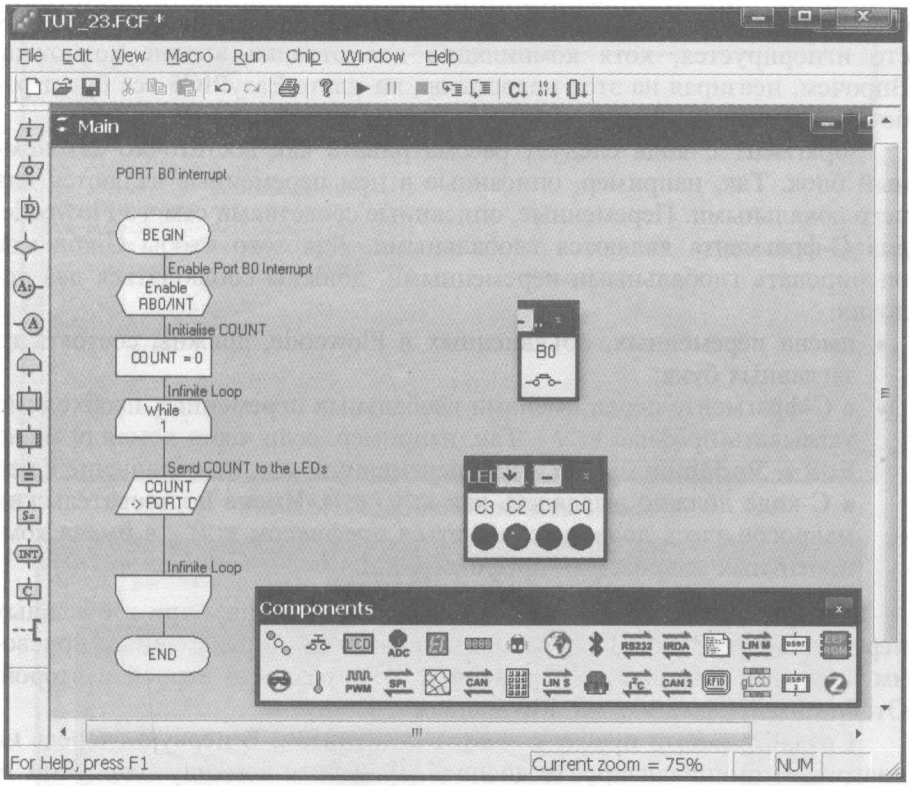


Рис. 2.56. Проект для прерывания по сигналу на выводе порта

Программирование на C

Как уже, наверное, понял читатель, возможности организации вычислительных процессов, предоставляемые в Flowcode элементом **Calculation**, весьма ограничены: присваивание и арифметические операции. Как говорится, особенно не разгуляешься. К счастью, проблема создания сложных вычислительных алгоритмов может быть решена включением в диаграмму фрагментов кода на языке программирования высокого уровня C.

На вертикальной линейке присутствует элемент **C code**, который можно разместить в любой точке диаграммы. Он представляет собой окно простого текстового редактора, позволяющий вводить код на C по правилам синтаксиса этого языка. Существуют некие особенности взаимодействия C-кода с самой программой Flowcode. Как это ни

странно, в режиме эмуляции код на С вообще не выполняется. Он просто игнорируется, хотя компиляция для отладки вполне возможна. Впрочем, невзирая на этот недостаток, на плате EasyPIC5 все будет работать нормально.

Фрагмент С-кода следует рассматривать как достаточно автономный блок. Так, например, описанные в нем переменные являются для него локальными. Переменные, описанные средствами самого Flowcode, для С-фрагмента являются глобальными. Для того чтобы С-код мог оперировать глобальными переменными, должны соблюдаться два условия:

- имена переменных, объявленных в Flowcode, должны состоять из заглавных букв;
- в С-фрагменте перед именами глобальных переменных необходимо указывать префикс `FCV_`. Так, например, если через команду меню **Edit ▶ Variables** мы объявили переменную `TIM`, то обращение к ней в С-коде должно выглядеть как `FCV_TIM`. Имена пользовательских макросов здесь должны снабжаться префиксом `FCM_`, а имена компонентных макросов — префиксом `FCD_`.

Теперь спроектируем следующую задачу. Опишем три глобальных переменных `A`, `B` и `C`. В С-фрагменте первым двум переменным присвоим числовые значения, а переменной `C` — разность первой и второй. Отобразим арифметику на ЖК-дисплее.

Создайте новый проект `C_code1` с дисплеем. В первую очередь на диаграмме инициализируйте дисплей. Выберите команду меню **Edit ▶ Variables** и объявите наши переменные типа `INT`. Добавьте в диаграмму элемент **C code**. В окне его свойств введите выражения языка С согласно рис. 2.57.

С первыми тремя выражениями присваивания все ясно. Далее идут вызовы компонентных макросов. Следует отметить, что работа с макросами (особенно с компонентными) в справочной системе Flowcode документирована крайне скупо. По сути дела, автору пришлось выяснять их синтаксис чисто эмпирическим путем, так что придется принять форму записи последних пяти строк без особых разъяснений. Хотя, по большому счету, их смысл довольно очевиден. Вначале на дисплей выводится число, содержащееся в переменной `AA`, затем — символ "-", далее — число в `BB` и символ "=". В завершение отображается разность значений.

В конце диаграммы организуйте пустой бесконечный цикл. В итоге рабочая область проекта должна соответствовать рис. 2.58.

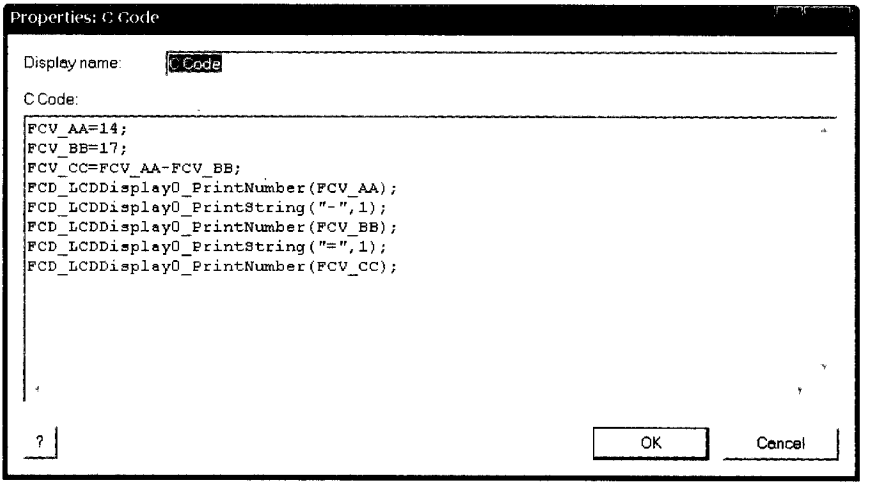


Рис. 2.57. Содержимое элемента **C Code**

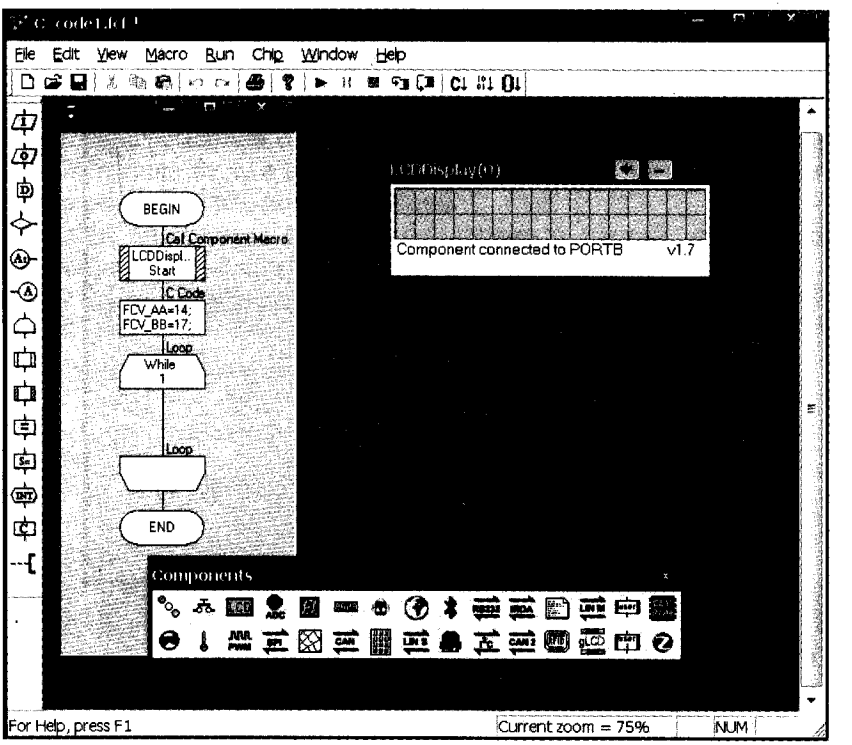


Рис. 2.58. Проект, использующий элемент **C code**

Скомпилируйте проект до HEX-файла и запустите его на плате EasyPIC5. Ожидается, что на дисплее высветится строка:

```
14-17=-3
```

Разработанная нами программа содержит простейший код на языке C. Собственно, для поставленной задачи без C можно было бы и вообще обойтись, так что этот пример можно считать чисто учебным. В следующем варианте используем более сложную конструкцию и другой подход к использованию C. Цель программы будет заключаться в вычислении факториалов чисел (скажем, от 0 до 5) с отображением их на ЖК-дисплее. Поиск факториала оформим в виде функции, написанной на C.

Создайте новый проект `C_code2` с ЖК-дисплеем. Для проектирования функции на языке C в Flowcode существует специальное средство, называемое сопутствующим кодом. Выберите команду меню **Edit ▶ Supplementary code**. В результате откроется диалоговое окно с двумя текстовыми полями (рис. 2.59).

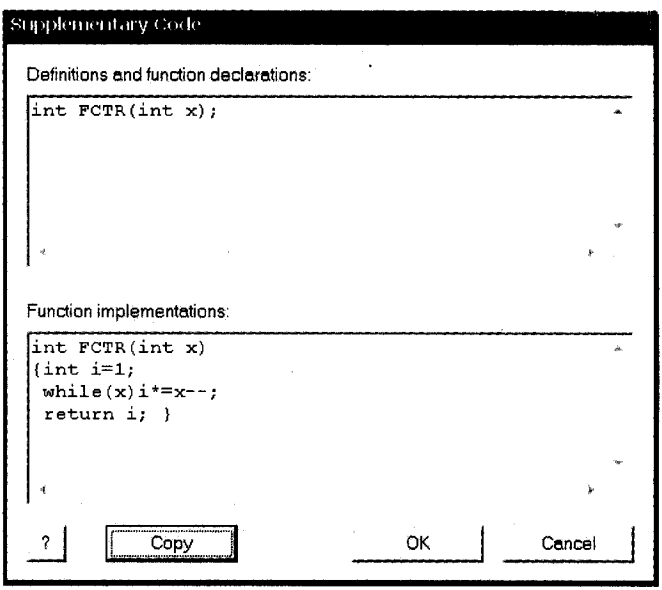


Рис. 2.59. Диалоговое окно **Supplementary Code**

Согласно концепциям построения проектов в C, в верхнем окне объявляются функция, а в нижнем вводится ее реализация. В детали мы вдаваться не будем. Для тех, кто знаком с языком C, здесь все понятно. Остальным же можем сказать одно: "Изучайте C"!

Выберите команду меню **Edit ► Variables** и создайте переменную `A` типа `INT`. Разместите в диаграмме инициализацию дисплея и элемент **C code** с фрагментом кода, показанным на рис. 2.60.

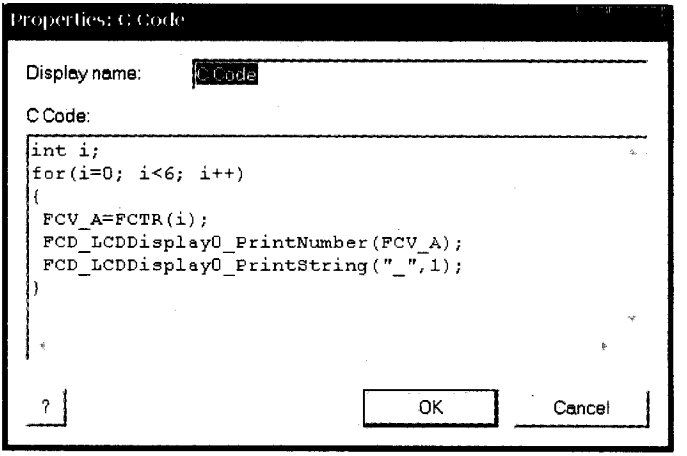


Рис. 2.60. Фрагмент кода на C

Здесь описана локальная переменная `i`, и организован цикл с изменением этой переменной от 0 до 5. В цикле вызывается функция вычисления факториала от `i`, а результаты отображаются на дисплее с разделением чисел символом подчеркивания.

В конец диаграммы вставьте бесконечный пустой цикл. Готовый проект показан на рис. 2.61.

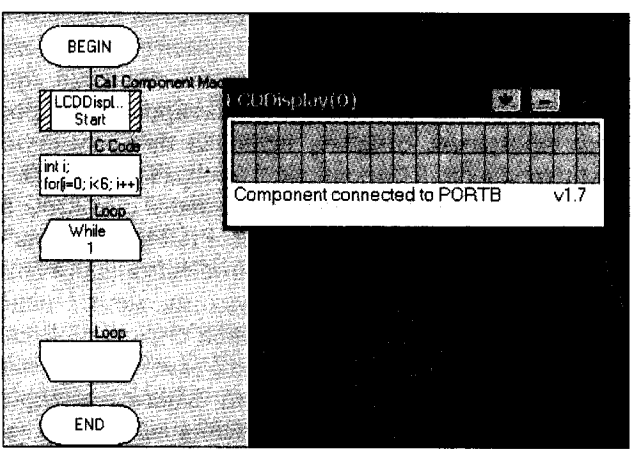


Рис. 2.61. Проект `C_code2`

Глава 3

Программирование микроконтроллеров на языке С

Язык программирования С (и его "потомок" С++) в настоящее время — наиболее распространенное средство профессионального программирования. В последние годы он все шире используется и в программировании микроконтроллеров. К настоящему времени создано несколько хороших С-компиляторов, с одним из которых мы и познакомимся в данной главе. Можно считать, что программирование на языке высокого уровня занимает промежуточное положение между проектированием типа Flowcode и низкоуровневым программированием на языке ассемблера. Промежуточным оно является как с точки зрения трудоемкости самого процесса разработки программ, так и в плане эффективности (ресурсоемкости, быстродействия) конечного продукта. Как бы там ни было, для освоения этой главы особых навыков работы на языке С от читателя не потребуется.

MikroC

Компилятор MikroC (точнее, интегрированная среда разработки проектов для микроконтроллеров PIC) от сербской компании Mikroelectronika — очень полезный и удобный инструмент как для начинающих, так и для профессиональных программистов. Система предоставляет полный набор сервисов по созданию и редактированию проекта, редактированию кода программ, отладке, включая программную эмуляцию работы микроконтроллера, и "прошивке" программы в MCU. Последняя функция обеспечивается устанавливаемой вместе с MikroC программой обслуживания аппаратного программатора PICFlash. Среда ориентирована на работу с платой разработки от Mikroelectronika (в данном случае EasyPIC5), однако в режиме отладки/эмуляции может использоваться и автономно. Пакет сопровождается хорошей справочной системой, содержащей файл справки на русском языке (см. прилагаемый к книге компакт-диск). Демонстрационные версии MikroC распространяются бесплатно. В них накладываются ограничения на размер (занимаемую память) разрабатываемых программ.

Сразу после установки пакета из дистрибутива система без каких либо особых настроек готова к работе. Главное окно программы может соответствовать рис. 3.1.

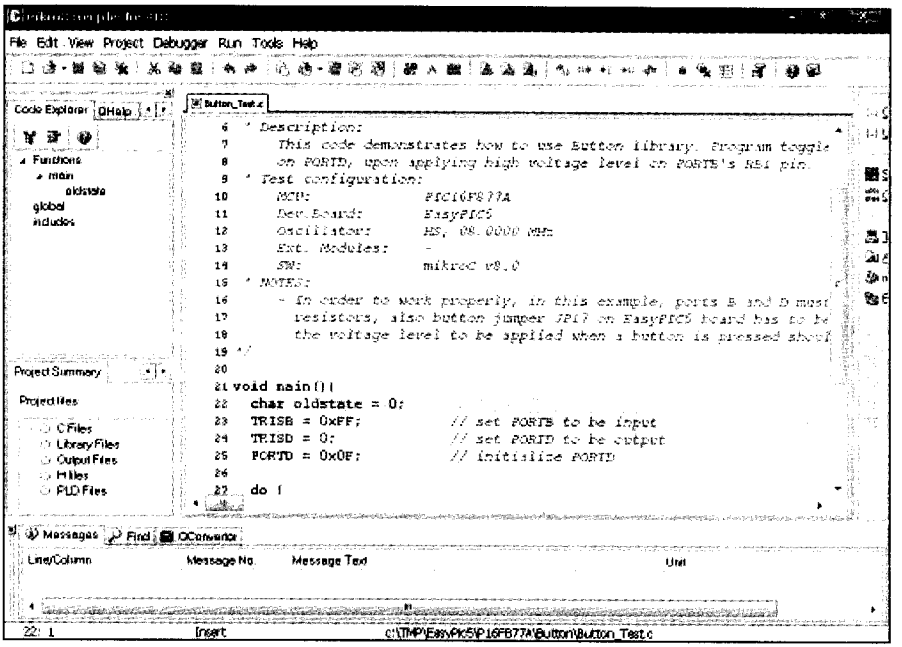


Рис. 3.1. Программа MikroC

Основная часть окна занята областью текстового редактора, в которой отображен текущий текстовый файл (например, С-код программы). Тем не менее, представленный на рис. 3.1 вариант отличается от предлагаемого по умолчанию. В исходном варианте установлено не очень удобное, на наш взгляд, цветовое решение окна текстового редактора. Для установки показанной на рис. 3.1 цветовой схемы в главном меню необходимо выбрать команду **Tools ► Options**. В открывшемся окне **Preferences** (Настройки) следует перейти в раздел **Editor ► Colors** и выбрать в раскрывающемся списке **Scheme** (Схема) элемент **mikro-Dream**. (рис. 3.2)

В левом верхнем углу главного окна MikroC целесообразно держать открытой вкладку **Code Explorer**, поскольку она позволяет легко переходить к различным функциям в С-коде. Ниже, на вкладке **Project Summary** можно переключаться между файлами проекта. В нижней части окна MikroC рекомендуется держать открытой вкладку **Messages**, в ко-

торой, в частности, появляются сообщения компилятора на этапе обработки программы. Здесь же выводится информация об ошибках.

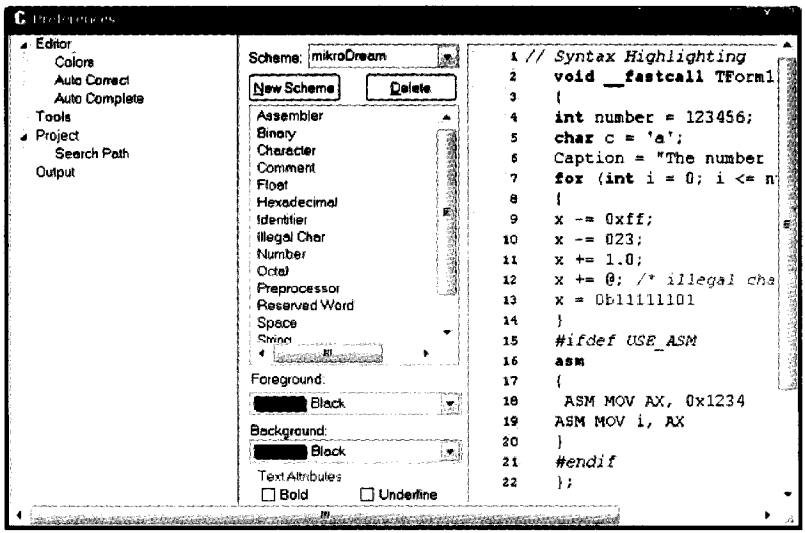


Рис. 3.2. Настройка цветовой схемы окна MikroC

В отличие от системы Flowcode, проект в MikroC состоит из довольно большого числа файлов. Кратко познакомимся с некоторыми из них. Файлу с расширением `.ppc` соответствуют свойства проекта. Его пример представлен в листинге 3.1.

Листинг 3.1. Пример файла .ppc

```
[DeviceName]
Value=P16F877A
[DeviceClock]
Value=8
[MainUnit]
Value=
[DeviceFlags]
Count=3
Value0=_LVP_OFF = $3F7F
Value1=_WDT_OFF = $3FFB
Value2=_HS_OSC = $3FFE
[BuildType]
bType=0
[ProjectFiles]
Count=1
Value0=LED_COUNT.c
[HeaderFiles]
```

Листинг 3.1. Окончание

```
Count=0
[ObjLibFiles]
Count=0
[PLDFiles]
Count=0
[EEPROMinfo]
isused=0
[SearchPath]
Count=3
Value0=C:\Program Files\Mikroelektronika\mikroC\Defs\
Value1=C:\Program Files\Mikroelektronika\mikroC\Uses\p16\
Value2=C:\IVB\3_STEP\MikroC\1\
[IncludePath]
Count=0
```

Все эти свойства устанавливаются по умолчанию или задаются пользователем при создании проекта, однако могут быть отредактированы и непосредственно в файле `.ppc`. Из листинга 3.1 видно, что здесь задается тип микроконтроллера, тактовая частота осциллятора и многое другое.

Файл с расширением `.c` (рис. 3.3) хранит С-код программы. Это — тот самый файл, который отображается в окне редактора.

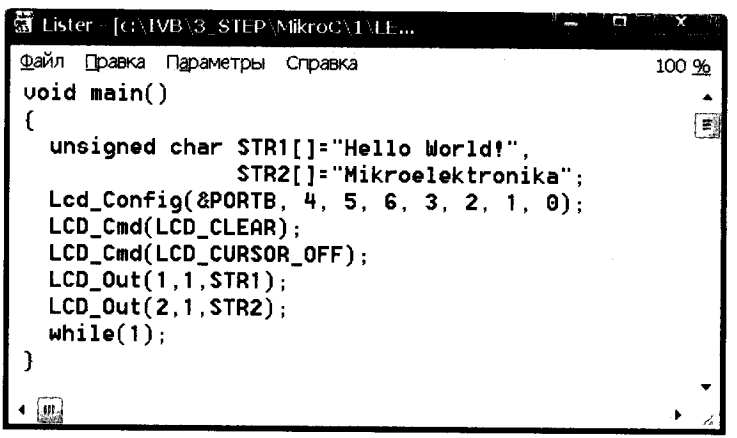


Рис. 3.3. Файл `.c`

Код по сути дублируется в файле с расширением `.cp`. Точнее, здесь сохраняется вариант предыдущей редакции текста с добавлением начальной дополнительной строки.

В файле .asm хранится ассемблерный код программы. В листинге 3.2 показан начальный фрагмент такого файла.

Листинг 3.2. Начальный фрагмент файла .asm

```
; ASM code generated by mikroVirtualMachine for PIC
; V. 8.1.0.0
; Date/Time: 06.05.2009 9:47:23
; Info: http://www.mikroe.com

; ADDRESS     OPCODE ASM
; ----------------------------------------------
$0000  $2850               GOTO   _main
$0147  $       _Delay_1us:
;delays.c,7 ::             void Delay_1us() {
;delays.c,8 ::             Delay_us(1);
$0147  $0000               NOP
$0148  $0000               NOP
;delays.c,9 ::             }
$0149  $0008               RETURN
$0126  $       _Delay_5500us:
;delays.c,31 ::            void Delay_5500us() {
;delays.c,32 ::            Delay_us(5500);
$0126  $300F               MOVLW  15
$0127  $1303               BCF    STATUS, RP1
$0128  $1283               BCF    STATUS, RP0
$0129  $00FB               MOVWF  STACK_11
$012A  $30FF               MOVLW  255
$012B  $00FA               MOVWF  STACK_10
$012C  $0BFB               DECFSZ STACK_11, F
$012D  $292F               GOTO   $+2
$012E  $2932               GOTO   $+4
$012F  $0BFA               DECFSZ STACK_10, F
$0130  $292F               GOTO   $-1
$0131  $292C               GOTO   $-5
$0132  $303E               MOVLW  62
$0133  $00FA               MOVWF  STACK_10
$0134  $0BFA               DECFSZ STACK_10, F
$0135  $2934               GOTO   $-1
;delays.c,33 ::            }
$0136  $0008               RETURN
```

Как видим, очень компактный код на языке C превращается в ассемблерный код весьма внушительных размеров.

Главным продуктом работы MikroC является исполняемый код — файл с расширением .hex. Это — бинарный файл, который записывается в память программ МК. Просматривать содержимое файла .hex в каком либо текстовом редакторе бессмысленно, поскольку в нем нет ни-

чего, кроме двоичного машинного кода. Представляет интерес, разве что, его размер, который можно сравнить с доступным объемом памяти программ МК. Остальные файлы проекта рядовому начинающему пользователю не нужны.

В главном меню программы большинство пунктов и подпунктов достаточно понятны. Так команда **File ► New** дает возможность создания нового текстового файла С-кода программы. Команда **Project ► Build** запускает процесс полной сборки программы. Пункт **Debugger** позволяет выбрать метод отладки: программный (симуляция на ПК) или аппаратный (на реальном МК с использованием отладчика mikroICD Debugger). Пункт **Run** запускает отладку программы. Здесь присутствует целый набор вариантов: пошаговое выполнение программы, установка точек прерывания и пр. Меню **Tools**, кроме всего прочего, отвечает за эмуляции некоторых устройств. На первых порах нам эти инструменты не понадобятся.

В меню **Help**, как и положено, представлены средства помощи и справочная информация. Еще раз подчеркнем, что справка в MikroC сделана очень хорошо, так что все подробности читатель может выяснить там. Далее в данной главе мы продемонстрируем возможности программирования на языке С.

Включаем светодиоды

Первая программа на С, которую мы разработаем, должна в бесконечном цикле последовательно высвечивать на светодиодах порта С двоичные числа от 0 до 7 с экспозицией в полсекунды.

Запускаем MikroC и в главном меню выбираем команду **Project ► New Project**. В результате откроется окно, показанное на рис. 3.4.

В поле **Project Name** введите имя проекта `LED_COUNT`. Во втором поле с помощью кнопки **Browse** (на рис. 3.4. не видна) найдите и установите путь к папке, в которой будет сохранен проект. В раскрывающемся списке **Device** выберите тип МК, с которым предстоит работать. Пока это — PIC16F877A. В поле **Clock** укажите тактовую частоту генератора, выраженную в мегагерцах (в нашем случае — 8 МГц).

Далее можно установить желаемую конфигурацию МК с помощью обширного списка флагов в нижней части диалогового окна **New Project**. В большинстве случаев можно ограничиться набором флагов по умолчанию, который устанавливается по нажатию кнопки **Default** (на рис. 3.4 также не видна). Как видно из описания в поле **Default settings**, в этом случае будет активизирован режим внешнего кварцевого осциллятора HS, отключится сторожевой таймер и будет запрещен режим

низковольтного программирования. Все это нас вполне устраивает, поэтому нажмите кнопку **OK** (опять-таки, на рис. 3.4 не видна), чтобы создать новый проект.

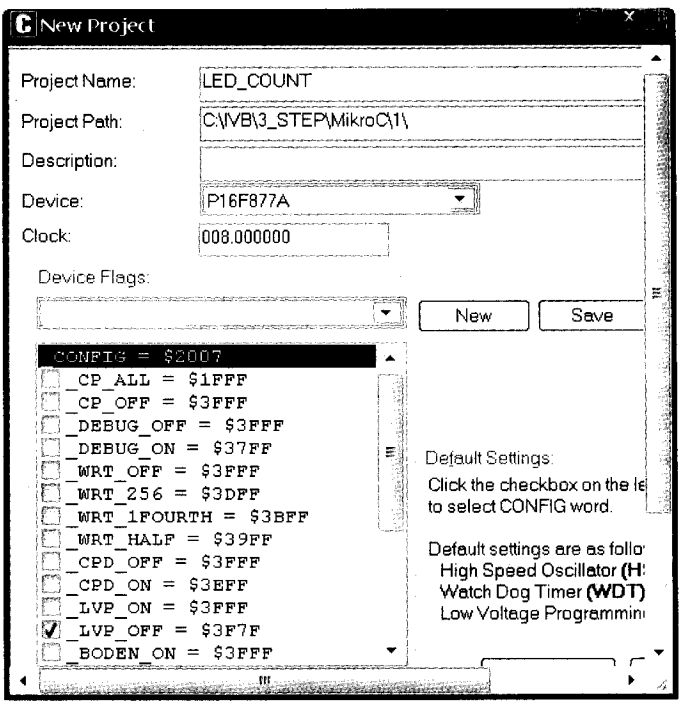


Рис. 3.4. Параметры нового проекта

Теперь мы попадаем в окно редактора исходного текста программы. Введите здесь С-код из листинга 3.3.

Листинг 3.3. Главная функция программы включения светодиодов

```
void main()
{
 unsigned short i;
 PORTC = 0;
 TRISC = 0;
 while(1)
  for(i=0; i<8; i++)
    {
     PORTC=i;
     Delay_ms(500);
    }
}
```

Здесь описана главная функция `main`. В ней объявлена беззнаковая целая переменная `i` типа `unsigned short`. Этот тип наиболее естественен при работе с портами, поскольку занимает один байт в памяти и позволяет представлять целые числа в диапазоне от 0 до 255. Следующие два выражения стандартны для использования порта, как выходного. `PORTC` и `TRISC` — это имена переменных, связанных с соответствующими регистрами. Объявлять их в программе не требуется, поскольку это уже сделано самой системой.

Далее организован бесконечный цикл `while`, внутри которого создан цикл `for` с изменением переменной `i` от 0 до 7, записью значения `i` в порт С и реализацией задержки. В данном варианте за задержку отвечает функция `Delay_ms`. Открыв справку по этой функции, можно увидеть, что она приостанавливает выполнение программы на количество миллисекунд, указанное в аргументе. Вот и все, что есть в нашей первой программе.

Компиляция программы и сборка проекта вплоть до формирования HEX-файла осуществляется по команде меню **Project ▶ Build** или с помощью комбинации клавиш <Ctrl+F9>. При наличии ошибок будет выдана соответствующая информация, и программа не откомпилируется. Средства диагностики ошибок в MikroC менее развиты по сравнению, скажем, с системами Borland C++ Builder или Microsoft Visual C++, однако при наличии определенного опыта вполне приемлемы.

Прежде чем записать программу в МК настоятельно рекомендуем протестировать ее в отладочном режиме MikroC. Для этого с помощью команды меню **Debugger ▶ Select Debugger** выберите программный симулятор Software PIC Simulator. Это означает, что работа программы моделируется на ПК. Затем выберите команду меню **Run ▶ Start Debugger**. В результате будет выделена начальная строка программы, а в рабочей области откроется окно **Watch** (Наблюдение) (рис. 3.4). Оно позволяет отслеживать текущее состояние всех объектов в программе: переменных, портов и др.

Прежде всего, следует составить список объектов, за значениями которых мы будем наблюдать. Для этого щелкните мышью в поле **Select variables from list** и отметьте интересующие элементы. Для внесения выбранного в список нажмите кнопку **Add** (Добавить). В результате объекты отобразятся в нижней части окна **Watch**. В нашем случае нас интересуют переменные `i` и `PORTC`.

Наиболее интересный вариант — проследить выполнение программы в пошаговом режиме строка за строкой по нажатию клавиши <F8>.

При этом в окне **Watch** будут отображаться текущие значения выбранных переменных. Описанный режим иллюстрирует рис. 3.6.

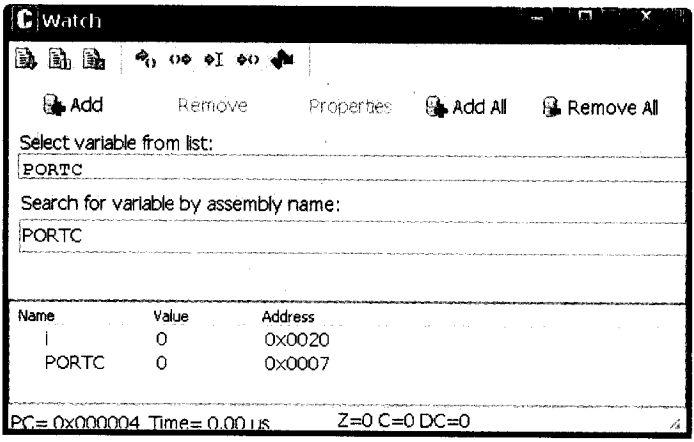


Рис. 3.5. Окно **Watch**

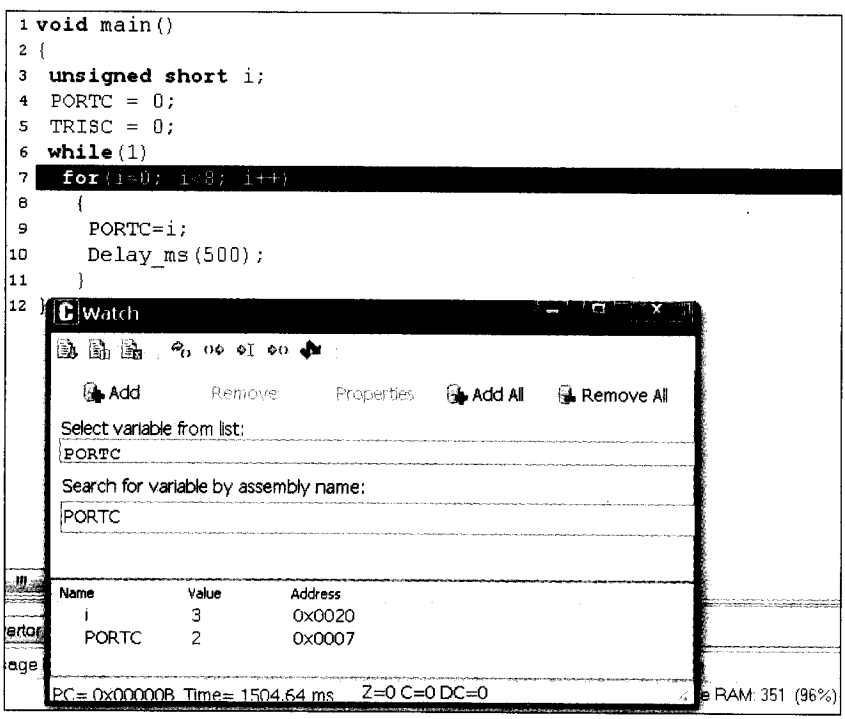


Рис. 3.6. Пошаговый режим отладки программы

Для выхода из режима отладки можно нажать комбинацию клавиш <Ctrl+F2>. В меню **Run** предусмотрены и другие средства отладки: установка точек прерывания, переход к позиции курсора и пр. С этими инструментами мы предлагаем разобраться читателю самостоятельно.

Итак, мы убедились в правильности работы программы в режиме отладки (симуляции). Загружаемый в МК HEX-файл уже готов и находится в рабочем каталоге проекта. Теперь необходимо записать программу в микроконтроллер. Этому мы уже научились в предыдущей главе. Затем можно запустить систему. Напомним, что светодиоды порта С включают, переведя третью позицию переключателя SW6 в положение "ON".

Заглянув в папку проекта, мы обнаружим там неожиданно большое число файлов: и временные файлы, и различные листинги, включая ассемблерный код программы. На самом деле, сам проект состоит всего из двух файлов: собственно проекта `LED_COUNT.ppc` и текстового файла `LED_COUNT.c`. Все остальные файлы в папке можно удалить.

Проект "Hello World!"

Этот проект зачастую идет первым, поскольку в нем программа здоровается с миром, однако мы предпочли все-таки вначале помигать лампочками, что для программирования МК вполне естественно.

Новый проект `Hello` создаем аналогично предыдущему. Текст программы, который необходимо ввести в редакторе, очень короток (листинг 3.4).

Листинг 3.4. Программа Hello

```
void main()
{
  unsigned char STR1[]="Hello World!",
                STR2[]="Mikroelektronika";
  Lcd_Config(&PORTB, 4, 5, 6, 3, 2, 1, 0);
  LCD_Cmd(LCD_CLEAR);
  LCD_Cmd(LCD_CURSOR_OFF);
  LCD_Out(1,1,STR1);
  LCD_Out(2,1,STR2);
  while(1);
}
```

Сразу напомним, что язык С различает строчные и прописные буквы. Не забывайте об этом.

Рассмотрим программу. Ее текст действительно очень компактен. В этом и заключается сила программирования на языке высокого уровня. В действительно система здесь выполняет несколько весьма трудо-

емких операций, однако для пользователя-программиста они представляются в виде функций, подготовленных разработчиками компилятора.

Вначале объявляются и инициализируются две строки: символьные массивы с текстом. Затем вызывается функция инициализации жидкокристаллического алфавитно-цифрового дисплея. В качестве первого параметра в функцию передается адрес порта, с которым работает дисплей. Мы уже знаем, что на плате EasyPIC5 он подключен к порту B. Детального объяснения остальных параметров разработчики не представили, но можно догадаться, что здесь различные линии связи распределяются по выводам порта. Остановимся на этой догадке, поскольку она оправдывает себя на практике. Попутно отметим, что предлагаемая разработчиками альтернативная функция инициализации `Lcd_Init` в нашей конфигурации вообще не работает (по-видимому, из-за не подходящего распределения выводов).

Далее следуют две функции, передающие в ЖК-дисплей управляющие команды. Команда `LCD_CLEAR` очищает дисплей, а команда `LCD_CURSOR_OFF` скрывает мигающий курсор. Информацию об этой и других функциях можно получить в справочной системе.

Основное действие выполняют два вызова функции `LCD_Out`. Первые два параметра здесь определяют номер строки (у нас их две) и номер позиции (всего 16) вывода данных. Содержимое выводимых данных определяет третий параметр — указатель на строку. Это может быть имя символьного массива, как в нашем варианте, или сама строка, т.е. набор символов, взятый в кавычки.

Вот и все! Соберите проект, запишите его в МК и запустите на выполнение. Не забудьте включить дисплей седьмой позицией переключателя SW9.

Вещественные числа

Работая с компилятором C, мы, в отличие от Flowcode, получаем возможность оперировать вещественными числами (числами с плавающей точкой). В данном разделе мы научимся отображать вещественное число на семисегментных индикаторах, и заодно лучше освоим работу с этим модулем. Задачу поставим следующим образом. У нас есть вещественное число A в диапазоне $0 <= A < 10$. Пусть это будет 8,753. Необходимо отобразить его на четырех семисегментных индикаторах, включая отображение десятичной точки.

В первую очередь необходимо разобраться, как работает индикатор. Из рис.3.7 видно, что он содержит восемь светящихся элементов: семь

"палочек", составляющих цифру, и "кружок", обозначающий десятичную точку.

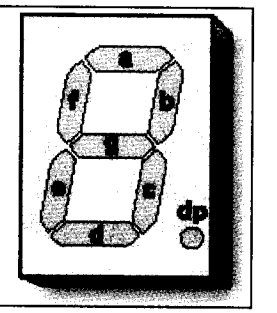


Рис. 3.7. Семисегментный индикатор

Этим элементам сопоставляется один байт, передаваемый на индикатор с выходного порта. На плате EasyPIC5 ему соответствует порт D. Семь младших разрядов сопоставляются с элементами a, b, c, d, e, f и g соответственно. Так, например, для высвечивания цифры 7 необходимо включить элементы a, b и c. Тогда в байте необходимо сформировать три единицы в младших разрядах: двоичное 111, десятичное 7, шестнадцатеричное 0x07. Для отображения цифры 8 необходимо зажечь все цифровые элементы: двоичное 1111111, десятичное 127, шестнадцатеричное 7F. Кодировка в индикаторе всех десятичных цифр представлена в табл. 3.1.

Таблица 3.1. Кодировка десятичных цифр в семисегментном индикаторе

Цифра	Двоичный код	Десятичный код	Шестнадцатеричный код
0	63	0111111	0x3F
1	6	0000110	0x06
2	91	1011011	0x5B
3	79	1001111	0x4F
4	102	1100110	0x66
5	109	1101101	0x6D
6	125	1111101	0x7D
7	7	0000111	0x07
8	127	1111111	0x7F
9	111	1101111	0x6F

Для того чтобы высветить десятичную точку после некоторой цифры, необходимо добавить в двоичный код единицу в восьмом (крайнем слева) разряде, т.е. прибавить к значению кода двоичное 10000000 (десятичное 128, шестнадцатеричное 0x80).

Теперь необходимо выделить четыре цифры, составляющих число. Для этого их целесообразно записать в массив из четырех элементов. Для нашего числа 8,753 элементы массива должны иметь значения `p[0]` = 3; `p[1]` = 5; `p[2]` = 7; `p[3]` = 8. Для получения такого массива наше число вначале необходимо превратить в целое `A` умножением исходного на 1 000 (и программным приведением к целому типу). Тогда, с учетом

особенностей целочисленного деления, описанных в главе о Flowcode, получим:

p[3] = A/1000;

p[2] = (A – p[3]*1000)/100;

p[1]= (A – p[3]*1000 – p[2]*100)/10;

p[0] = A – [3]*1000 – p[2]*100 – p[1]*10.

Рассмотрим принцип работы семисегментного индикатора. С порта данных (в нашем случае — порт D) поступает код цифры, которую необходимо отобразить в данный момент (возможно, вместе с десятичной точкой). С управляющего порта (в нашем случае — порт A) поступает код управления, показывающий, на каком из четырех индикаторов будет высвечена цифра. Код управления считывается из четырех младших разрядов порта. Один из этих разрядов должен содержать 1, остальные — 0. Так, если 1 находится в самом младшем разряде, то рабочим будет крайний справа индикатор. Цифра отображается до тех пор, пока в соответствующем разряде удерживается 1. Уместно напомнить, что перемещение единицы на один разряд влево эквивалентно умножению кодируемого числа на два, что в языке C удобно реализовать с помощью оператора сдвига <<. Дальнейшие операции прокомментируем уже в обсуждении текста программы.

Создайте новый проект с именем `Seg4Float` и введите текст из листинга 3.5.

Листинг 3.5. Программа вывода числа на семисегментные индикаторы

```
unsigned short shift,index;
unsigned short DIG[]={0x3F,0x06,0x5B,0x4F,0x66,
                      0x6D,0x7D,0x07,0x7F,0x6F};
void main()
{
  double a=8.753;
  long aa=(long)(a*1000.0);
  long aaa[4];
  unsigned short tmp;
  aaa[3]=aa/1000;
  aaa[2]=(aa-aaa[3]*1000)/100;
  aaa[1]=(aa-aaa[3]*1000-aaa[2]*100)/10;
  aaa[0]=aa-aaa[3]*1000-aaa[2]*100-aaa[1]*10;
  index = 0;
  shift = 1;
  PORTA = 0;  TRISA = 0;
  PORTD = 0;  TRISD = 0;
```

Листинг 3.5. Окончание

```
  while(1)
  {
   PORTA = 0;
   tmp = DIG[aaa[index++]];
   if(index==4) tmp+=0x80;
   PORTD=tmp;
   PORTA = shift;
   shift <<= 1;
   if (shift > 8) shift = 1;
   if (index > 3) index = 0;
  }
}
```

Массив `DIG` содержит коды цифр из табл. 3.1 в шестнадцатеричной системе счисления. В переменной `a` хранится исходное число. Переменная `aa` получена умножением `a` на 1 000. Очевидно, что для `aa` требуется тип `long` (длинное целое). Далее, в соответствии с описанной выше арифметикой, в массиве `aaa` формируются цифры числа. Затем задаются начальные значения для счетчиков. Поскольку порты A и D передают данные на индикаторы, они являются выходными. Их настройка реализована в следующих двух строках программы.

Весь дальнейший код заключен в бесконечный цикл `while`. Здесь вначале сбрасывается предыдущее значение в порте A. Счетчик `index` показывает номер разряда отображаемого числа, начиная с нулевого, при его инкрементировании. В таком случае, `aaa` с соответствующим индексом — это очередная цифра числа, а `DIG` — код этой цифры, записываемый в промежуточную переменную `tmp`.

После старшей цифры мы должны включить десятичную точку, поэтому, если инкрементированная переменная `index` стала равна 4, то к `tmp` прибавляется 0x80. Значение `tmp` выдается в порт D, а значение счетчика `shift` (при начальной величине 1) — в порт A. Далее счетчик `shift` удваивается. Если он становится больше 8, то это говорит о том, что мы отобразили все число, и для следующего цикла счетчик необходимо опять установить в 1. Аналогично, если счетчик `index` превысил 3, то его надо сбросить в 0.

Поскольку содержимое цикла выполняется очень быстро, в процессе работы системы мы не заметим "перескоков" изображения по разрядам. Будет создана полная иллюзия одновременного свечения всех индикаторов. Попробуйте добавить в цикл задержку порядка 10 мс или более. Это позволит увидеть последовательное переключение индикаторов.

Напомним, что для работы с индикаторами подтягивающие регистры порта B необходимо сконфигурировать в состояние "PullDown".

АЦП

Рассмотрим реализацию в MikroC важнейших микроконтроллерных приложений, связанных с аналого-цифровым преобразованием. Практически во всех МК встроены АЦП, причем — с одним или несколькими входными каналами. В микроконтроллере PIC16F877A, в частности, присутствует двухканальный 10-разрядный преобразователь, входы которого организованы в младших разрядах порта A. Для настройки выводов порта на аналоговый ввод устанавливаются флаги особого регистра специального назначения, что будет показано далее в программах.

В первой программе (проект ADC_on_LEDs) отобразим значение оцифрованного сигнала в двоичной системе счисления путем его высвечивания на светодиодах. Поскольку результат 10-разрядного аналого-цифрового преобразования представляет собой целое число в диапазоне от 0 до 1 023 включительно, одного восьмиразрядного порта недостаточно. Поступим следующим образом. Младшие восемь разрядов числа отобразим на светодиодах порта D, а два старших — через младшие разряды порта C. Программа выглядит весьма просто (листинг 3.6).

Листинг 3.6. Программа аналого-цифрового преобразования

```
unsigned int temp_res;

void main()
 {
  ADCON1=0x80;
  TRISA=0xFF;
  PORTD=0; TRISD=0;
  PORTC=0; TRISC=0;
  do
  {
    temp_res=ADC_Read(1);
    PORTD=temp_res;
    PORTC=temp_res >> 8;
  } while(1);
}
```

В первой исполняемой строке основной функции в регистр ADCON1 записывается значение 0x80 (устанавливается в 1 старший разряд). Это предписывает переключить входы порта A на прием информации в аналоговом виде. Следующая команда настраивает порт A на ввод: все разряды устанавливаются в лог. 1. При этом необходимо

убедиться в том, что подтягивающие регистры порта отключены. Далее следуют знакомые нам операции настройки портов D и C на вывод.

В бесконечном цикле вызывается функция `ADC_Read`, которая считывает текущую величину аналогового сигнала и оцифровывает ее. Аргумент функции задает номер канала АЦП. Как мы уже заметили, результат в одном байте не помещается, поэтому возвращаемое функцией значение заносится в переменную `temp_res` типа `int`. Затем в порт D записывается восемь младших разрядов результирующего числа. Теперь сдвигаем весь код числа на восемь разрядов вправо, и то, что осталось, записываем в порт C. А остались — два старших разряда, что и требовалось.

Не забыв установить подтягивающие резисторы порта B в состояние "PullDown", запускаем программу. Покрутив регуляторы потенциометров в правом верхнем углу платы, можно убедиться, что в крайнем левом положении ни один светодиод не горит, а при максимальном напряжении светятся все восемь лампочек на порте D и две лампочки на порте C.

Смотреть на двоичный код числа (тем более, разнесенный на два порта) явно не интересно. Кроме того, бывает необходимо увидеть не результат оцифровки в виде целого числа от 0 до 1 023, а реальную физическую величину (например, напряжение на аналоговом входе в вольтах) — своего рода обратное, цифро-аналоговое преобразование.

Создадим проект, решающий следующую задачу. Мы знаем, что на аналоговый вход может быть подано напряжение от 0 до 5 В. Вначале оцифруем входное напряжение, а затем отобразим исходную величину в вольтах на ЖК-дисплее.

Текст программы (проект `ADC_on_LCD`), опять-таки, весьма компактен (листинг 3.7).

Листинг 3.7. Отображение аналоговой величины на ЖК-дисплее

```
unsigned int adc;
char text[16];
float rf;

void main()
{
 Lcd_Config(&PORTB, 4, 5, 6, 3, 2, 1, 0);
 LCD_Cmd(LCD_CURSOR_OFF);
 LCD_Cmd(LCD_CLEAR);
 ADCON1=0x80;
 TRISA=0xFF;
 while (1)
 {
```

Листинг 3.7. Окончание

```
  adc=ADC_read(1);
  rf=adc/1023.0*5.0;
  FloatToStr(rf,text);
  LCD_Cmd(LCD_CLEAR);
  LCD_Out(1,1,text);
  Delay_ms(100);
 }
}
```

В начале главной функции находятся знакомые нам функции инициализации дисплея, отключения курсора и очистки. Далее реализовано переключение на аналоговый ввод и конфигурирование порта A на ввод. В бесконечном цикле в переменную `adc` записывается оцифрованное значение текущего напряжения на входе. Далее переменной `rf` вещественного типа присваивается отмасштабированное в диапазоне от 0 до 5 значение напряжения. Следует особо отметить, что в данном случае нам требуется не целочисленное деление, а обычное: с плавающей точкой. Поскольку `adc` — целая величина, во избежание целочисленной операции делитель необходимо представить в вещественном виде. Для этого его достаточно записать с десятичной точкой.

Теперь преобразуем вещественное число `rf` в строку символов. В языке C для этого есть специальная функция `FloatToStr`. Первый аргумент — это число, которое необходимо представить в символьном виде, а второй — строка, в которую требуется поместить результат. Для отображения строки на дисплее используется функция `LCD_Out`. Первый и второй аргументы здесь — номер строки и номер колонки, в которой выводится строка (третий аргумент).

Все здесь выглядит простым и логичным. Однако в программе присутствует информация к размышлению. Читателю предлагается самостоятельно поэкспериментировать и попытаться понять, например, почему строка должна быть длиной 16 знаков, зачем нужна очистка дисплея внутри цикла и задержка внутри цикла. В целом же, программа — весьма привлекательна, а ее результат — нагляден.

Термометр и терморегулятор

Термодатчик DS1820 подключается к микроконтроллеру по соединению типа OneWire. Вопреки названию, это — отнюдь не просто подключение провода от датчика к одному из выводов МК. Во-первых, OneWire — это название специализированного протокола передачи, поддерживаемого рядом внешних устройств. Во-вторых, данные здесь передаются не в аналоговом, как для обычного проводного соединения,

а в цифровом виде. Внутри самого датчика производится соответствующее аналого-цифровое преобразование. В MikroC разработаны функции, обслуживающие OneWire-устройства, с которыми нам и предстоит поработать.

Проект `OneWire` должен в реальном времени считывать с датчика окружающую температуру и отображать ее на ЖК-дисплее. Кроме того, во второй строке дисплея МК должен высветить одно из состояний: "Холодно", "Нормально" или "Жарко" (по-английски), — в зависимости от того, в какой диапазон попадает текущая температура.

Термодатчик может быть подключен к выводу 5 порта А или к выводу 2 порта Е. Выберем второй вариант. В таком случае перемычка рядом с датчиком на плате EasyPIC5 должна быть установлена в нижнее положение.

Код программы показан в листинге 3.8.

Листинг 3.8. Программа определения температуры

```
void main()
{
char text[4];
long temp;
  char tempo[11];
  ADCON1 = 7;
  Lcd_Config(&PORTB, 4, 5, 6, 3, 2, 1, 0);
  Lcd_Cmd(LCD_CURSOR_OFF);
  Lcd_Out(1, 1, "Temp: ");
  do {
    Ow_Reset(&PORTE,2);
    Ow_Write(&PORTE,2,0xCC);
    Ow_Write(&PORTE,2,0x44);
    Delay_us(120);
    Ow_Reset(&PORTE,2);
    Ow_Write(&PORTE,2,0xCC);
    Ow_Write(&PORTE,2,0xBE);
    temp =  Ow_Read(&PORTE,2)*5;
    LongToStr(temp,tempo);
    text[0]=tempo[8];
    text[1]=tempo[9];
    text[2]='.';
    text[3]=tempo[10];
    Lcd_Out(1,7,text);
    Lcd_Chr(1,11,223);
    Lcd_Out(1,12,"C    ");
    if(temp<290) LCD_Out(2,5,"COLDLY");
    else
     if(temp>305) LCD_Out(2,5,"HEAT  ");
```

Листинг 3.8. Окончание

```
      else LCD_Out(2,5,"NORMA ");
     Delay_ms(100);
   } while (1);
}
```

Оператор `ADCON1 = 7;` предписывает настроить МК на цифровой ввод. Вслед за этим инициализируется ЖК-дисплей, скрывается курсор, и в первой позиции первой строки выводится текст "Temp: ". Все дальнейшее выполняется в бесконечном цикле.

Работы с OneWire документирована в MikroC весьма скупо, поэтому первые семь строк после ключевого слова `do` примем как алгоритм стандартной процедуры подготовки датчика к считыванию данных. Для нас интерес представляет функция `Ow_Read`. Ее первый аргумент — адрес порта, в который передаются данные, а второй — номер вывода. Функция возвращает оцифрованное значение полученного с датчика сигнала после некоторых преобразований и форматирования. В нашем случае возвращаемое значение будет целым знаковым числом в диапазоне от –110 до 250, что соответствует диапазону температур от –55 до +125 градусов Цельсия. В программе эта величина умножается на пять и присваивается переменной `temp`. Напомним, что температура измеряется датчиком с дискретностью 0,5°. Таким образом, в переменной `temp` хранится целое число, равное удесятеренному значению температуры.

Далее в программе целое число преобразуется в строку функцией `LongToStr`. Строка для отображения числа типа `long` должна быть не менее 11 символов в длину. Символы выравниваются по правой границе строки.

Следующая задача — извлечь из строки `tempo` символы цифр и записать их в строку `text` длиной в четыре символа. Итак, `text[0]` принимает значение старшей цифры целой части температуры, `text[1]` — вторую цифру целой части. В `text[2]` записывается символ десятичной точки, а в `text[3]` — цифра дробной части температуры.

Выводим строку `text` в седьмой позиции (первые шесть — это "Temp: "), а после нее, в 11-й позиции, с помощью функции вывода символа `Lcd_Chr` — символ с кодом 223 (т.е. "°"). Далее выводим строку "C ". Пробелы после "C" необходимы для заполнения первой строки дисплея до конца во избежание появления там "мусора".

Из следующих строк программы видно, что во второй строке дисплея (в ее центре) выводится слово "COLDLY", если температура меньше 29 градусов; "HEAT", если температура выше 30,5 градусов, и "NORMA" в остальных случаях. Таким образом, мы создали прототип терморегулятора. Если вместо вывода указанных слов микроконтроллер

будет включать нагреватель или холодильник, то мы получим систему, напоминающую кондиционер. По сути, в кондиционерах действительно работают микроконтроллеры по похожей схеме.

Графический дисплей

Графический ЖК-дисплей для платы EasyPIC5 позволяет в монохромном режиме рисовать и отображать текст. Экран дисплея представляет собой матрицу размерами 128x64 пикселей. Отсчет координат ведется от левого верхнего угла экрана вправо и вниз. При выводе текста в нашем распоряжении восемь строк, а горизонтальное смещение задается в пикселях.

Вообще говоря, компилятор MikroC предоставляет в распоряжение пользователя большой набор функций для рисования. Тем не менее, ситуация с реализацией этих возможностей, мягко говоря, не вполне благополучная. По крайней мере, для некоторых функций фирменное описание не полностью соответствует реальности. Так, например, в описании указано, что функция `Glcd_Init` принимает семь аргументов, а на самом деле их восемь. Далеко не всегда понятно из описания и назначение некоторых аргументов. Странно выглядит использование функции `Glcd_Write_Text`. Похоже, что допустимо использование только заглавных латинских букв. Не удается вывести в строке восклицательный знак "!". Не допускаются пробелы и символы подчеркивания, а в качестве разделителей приходится пользоваться символом ":", который сам не выводится. Имеются и другие странности, бороться с которыми можно, только рассматривая примеры программ, прилагаемые к системе.

Тем не менее, организовать привлекательный вывод графической информации вполне возможно. Так, в листинге 3.9 показан текст программы (проект `GLCD1`), иллюстрирующей две возможности дисплея: вывод текста и рисование окружности.

Листинг 3.9. Программа рисования текста и окружности

```
void main()
{
  unsigned short i;
  char *Text;
  Glcd_Init(&PORTB, 0, 1, 2, 3, 5, 4, &PORTD);
  while(1)
  {
   Glcd_Fill(0x00);
   Text = "HELLO:WORLD";
   Delay_ms(500);
```

Листинг 3.9. Окончание

```
    Glcd_Write_Text(Text, 32,3, 1);
    Delay_ms(500);
    for (i = 1; i <= 10; i++)
     Glcd_Circle(63,32, 3*i, 1);
   }
}
```

В аргументах функции `Glcd_Init` указываются адрес*а* портов управления и данных. Что касается остальных параметров, то предоставляет любознательному читателю попробовать разобраться с ними самостоятельно. Функция `Glcd_Fill(0x00)`, по сути, очищает экран. Упомянутая выше функция `Glcd_Write_Text(Text, 32, 3, 1)` выводит текстовую строку приветствия (к сожалению без восклицательного знака) в третьей строке экрана, начиная с 32-го пикселя по горизонтали. Последний параметр (1), якобы, задает цвет, однако на самом деле, судя по всему, означает нечто другое. Впрочем, со значением 1 все работает, как надо, поэтому изменять его смысла нет.

Программа с помощью цикла `for` рисует десять концентрических окружностей различного радиуса. Казалось бы, здесь не должно возникать никаких проблем, но и тут не обошлось без сюрпризов. На экране прорисовываются не окружности, а эллипсы. Другими словами, масштаб по вертикали расходится с масштабом по горизонтали. Это, конечно, несколько огорчает, но все же, запустив программу на выполнение, мы получаем весьма привлекательную картинку. Напомним, что включение графического дисплея осуществляется переводом последней позиции на SW9 в состояние "ON".

Теперь займемся более интересной задачей... В проекте `GLCD2` заставим систему рисовать на экране графического дисплея график изменения температуры с датчика DS1820 за последние 10 секунд с ежесекундным обновлением. Текст программы представлен в листинге 3.10.

Листинг 3.10. Программа рисования графика изменения температуры

```
void main()
{
  int i;
  long temp,mas[12];
  for(i=0;i<12;i++) mas[i]=50;
  ADCON1 = 7;
  Glcd_Init(&PORTB, 0, 1, 2, 3, 5, 4, &PORTD);
  do {
    Ow_Reset(&PORTE,2);
    Ow_Write(&PORTE,2,0xCC);
    Ow_Write(&PORTE,2,0x44);
```

Листинг 3.10. Окончание

```
    Delay_us(120);
    Ow_Reset(&PORTE,2);
    Ow_Write(&PORTE,2,0xCC);
    Ow_Write(&PORTE,2,0xBE);
    temp =  Ow_Read(&PORTE,2);
    for(i=0;i<11;i++) mas[i]=mas[i+1];
    mas[11]=100-temp*3;
    Glcd_Fill(0x00);
    for(i=1;i<12;i++)
      Glcd_Line((i-1)*10,mas[i-1],i*10,mas[i],1);
    Delay_ms(1000);
  } while (1);
}
```

Пусть график представляет собой ломаную линию, построенную по 12 точкам. Каждая точка — значение температуры. Для этого нам понадобится массив `mas` из 12 чисел. Вначале запишем в него постоянное произвольное значение (начальный участок графика нас не интересует).

В бесконечном цикле в переменную `temp` считывается очередное значение с датчика, которое сохраняется в последнем (одиннадцатом) элементе массива. Предварительно все остальные значения в массиве сдвигаются вниз по индексу. На место нулевого записывается значение первого, на место первого — значение второго и т.д. В операторе `mas[11]=100-temp*3` учтено, что вертикальная координата "перевернута", т.е. ось направлена сверху вниз. Уровень отсчета 100 и масштабный множитель 3 подобраны эмпирически для наиболее наглядного отображения. В цикле каждый раз с помощью функции `Glcd_Line` очищается экран. Во внутреннем цикле `for` строится ломаная линия. Требуемая частота перерисовки изображения достигается за счет задержки.

Запустите программу на выполнение. На экране отобразится график изменения температуры в чистом виде. Зажмите в пальцах термодатчик, и график пойдет вверх. Отпустите датчик — отобразится уменьшение температуры. Отметим, что все будет выглядеть более или менее нормально для диапазона изменения температур 27..33 градуса. При других условиях график должен выйти за пределы экрана. Чтобы убедиться в этом, поднесите к термодатчику, например, горящую спичку.

Сенсорный экран

Взаимодействие МК с сенсорным экраном (touch-панелью) визуально выглядит весьма впечатляюще. Мы продемонстрируем его на

примере, поставляемом с компилятором MikroC. Но вначале опишем процесс работы программы.

Прежде всего необходимо отключить подтягивающие резисторы порта A и семисегментные индикаторы. Также выключаются светодиоды (по крайней мере, для порта A). Должны быть включены четыре младших и самая старшая позиции переключателя SW9. При запуске микроконтроллера активизируется процедура калибровки. На экране ЖК-дисплея появится приглашение указать левый нижний угол рабочей области. Необходимо стилусом или просто не остро заточенным карандашом несильно, но уверенно коснуться панели в требуемой точке. Затем аналогичную процедуру следует проделать для правого верхнего угла.

В верхней части экрана дисплея высветится кнопка **CLEAR** для очистки рабочей области, а также три кнопки для задания толщины прорисовываемых линий (толстая, средняя и тонкая). Справа от них отображена кнопка, переключающаяся при каждом касании стилуса из состояния **ERASE** (стирание) в состояние **WRITE** (запись). Исходным будет состояние **WRITE**. В нем стилус оставляет на рабочей области след в виде линии. В состоянии **ERASE** стилус выполнят роль ластика выбранной толщины, которым можно стирать пиксели на экране.

В отличие от предыдущих примеров, программа взаимодействия с touch-панелью довольно большая. Приводим ее практически в том виде, в каком она дана в фирменном примере (листинг 3.11).

Листинг 3.11. Программа взаимодействия с сенсорным экраном

```
char write_erase, pen_size;
unsigned int x_coord, y_coord;
long x_coord128, y_coord64;       // Масштабируемая позиция XY
// Калибровочные константы:
int cal_x_min, cal_y_min, cal_x_max, cal_y_max;
char write_msg[] = "WRITE";       // Сообщения меню дисплея
char clear_msg[] = "CLEAR";
char erase_msg[] = "ERASE";
// Пороговое значение для обнаружения нажатия
const unsigned int ADC_THRESHOLD = 900;

// Если есть нажатие на touch-панель, возвращается 1.
// В противном случае возвращается 0
char PressDetect()
{
  unsigned adc_rd;
  char result;
  // Обнаружение нажатия
```

Листинг 3.11. Продолжение

```
  // DRIVEA = 0 (LEFT drive off, RIGHT drive off, TOP drive on)
  PORTC.F0 = 0;
  PORTC.F1 = 0;          // DRIVEB = 0 (BOTTOM drive off)
  Delay_us(3500);
  adc_rd = ADC_read(1);  // Чтение RA1 (READ-Y)
  result = (adc_rd > ADC_THRESHOLD);  // Если обнаружена лог. 1
    //Учет дребезга контактов, повтор обнаружения через 2 мс
  Delay_ms(2);
  adc_rd = ADC_read(1);
  result = result & (adc_rd > ADC_THRESHOLD);
  return result;
}

unsigned int GetX() {
  unsigned int result;
  // Чтение X
  // DRIVEA = 1 (LEFT drive on, RIGHT drive on, TOP drive off)
  PORTC.F0 = 1;
  PORTC.F1 = 0;          // DRIVEB = 0 (BOTTOM drive off)
  Delay_ms(5);
  result = ADC_read(0); // Чтение X с RA0 (НИЗ)
  return result;
}

unsigned int GetY() {
  unsigned int result;
  //Чтение Y
  // DRIVEA = 0 (LEFT drive off, RIGHT drive off, TOP drive on)
  PORTC.F0 = 0;
  PORTC.F1 = 1;          // DRIVEB = 1 (BOTTOM drive on)
  Delay_ms(5);
  result = ADC_read(1); // Чтение Y с RA1 (ЛЕВО)
  return result;
}

void Calibrate() {
  Glcd_Dot(0,63,1);
  Glcd_Write_Text("TOUCH BOTTOM LEFT",10,3,1);
  while (!PressDetect());
  // Получение калибровочных констант (чтение и компенсация
  // нелинейности Touch-панели)
  cal_x_min = GetX() - 10;
  cal_y_min = GetY() - 10;
  Delay_ms(1000);
  Glcd_Fill(0);
  Glcd_Dot(127,0,1);
```

Листинг 3.11. Продолжение

```
  Glcd_Write_Text("TOUCH UPPER RIGHT",10,4,1);
  while (!PressDetect()) ;
  // Получение калибровочных констант (чтение и компенсация
  // нелинейности Touch-панели)
  cal_x_max = GetX() + 5;
  cal_y_max = GetY() + 5;
  Delay_ms(1000);
}

void main() {
  ADCON1 = 0x07;
  PORTA  = 0x00;
  TRISA  = 0x03;              // RA0 и RA1 - аналоговые входы
  PORTC  = 0 ;
  TRISC  = 0 ;                // PORTC - выход
  Glcd_Init(&PORTB, 0, 1, 2, 3, 5, 4, &PORTD);
  Glcd_Fill(0);
  Glcd_Set_Font(FontSystem5x8, 5, 8, 32);
  Glcd_Write_Text("CALIBRATION", 24, 3, 1);
  Delay_ms(1500);
  Glcd_Fill(0);
  Calibrate();
  Glcd_Fill(0);
  Glcd_Write_Text("WRITE ON SCREEN", 24, 4, 1) ;
  Delay_ms(1000);
  Glcd_Fill(0);
  Glcd_Fill(0);
  Glcd_V_Line(0,7,0,1);
  Glcd_Write_Text(clear_msg,1,0,0);
  Glcd_V_Line(0,7,97,1);
  Glcd_Write_Text(erase_msg,98,0,0);
  //Прорисовка меню:
  Glcd_Rectangle(41,0,52,9,1);
  Glcd_Box(45,3,48,6,1);
  Glcd_Rectangle(63,0,70,7,1);
  Glcd_Box(66,3,67,4,1);
  Glcd_Rectangle(80,0,86,6,1);
  Glcd_Dot(83,3,1);
  write_erase = 1;
  pen_size = 1;
  while (1) {
    if (PressDetect()) {
      // После обнаружения нажатия считываем координату X-Y
      // и преобразовываем для пространства 128x64
      x_coord = GetX() - cal_x_min;
      y_coord = GetY() - cal_y_min;
```

Листинг 3.11. Продолжение

```
x_coord128 = (x_coord * 128l) / (cal_x_max - cal_x_min);
y_coord64 = (64 -(y_coord*64) / (cal_y_max - cal_y_min));
if ((x_coord128 < 0) || (x_coord128 > 127))
   continue;
if ((y_coord64 < 0) || (y_coord64 > 63))
   continue;
// Если нажата кнопка "CLEAR"
if ((x_coord128 < 31) && (y_coord64 < 8)) {
   Glcd_Fill(0);
   //Pen Menu:
   Glcd_Rectangle(41,0,52,9,1);
   Glcd_Box(45,3,48,6,1);
   Glcd_Rectangle(63,0,70,7,1);
   Glcd_Box(66,3,67,4,1);
   Glcd_Rectangle(80,0,86,6,1);
   Glcd_Dot(83,3,1);
   Glcd_V_Line(0,7,0,1);
   Glcd_Write_Text(clear_msg,1,0,0);
   Glcd_V_Line(0,7,97,1);
   if (write_erase)
     Glcd_Write_Text(erase_msg,98,0,0);
   else
     Glcd_Write_Text(write_msg,98,0,0);
   }
//Если нажата кнопка "WRITE/ERASE"
if ((x_coord128 > 96) && (y_coord64 < 8)) {
   if (write_erase) {
     write_erase = 0;
     Glcd_Write_Text(write_msg,98,0,0);
     Delay_ms(500);
     }
   else {
     write_erase = 1;
     Glcd_Write_Text(erase_msg,98,0,0);
     Delay_ms(500);
     }
   }
//Если выбран размер пера
if ((x_coord128 >= 41) &&
    (x_coord128 <= 52) &&
    (y_coord64 <= 9))
   pen_size = 3;
if ((x_coord128 >= 63) &&
    (x_coord128 <= 70) &&
    (y_coord64 <= 7))
   pen_size = 2;
```

Листинг 3.11. Окончание

```
      if ((x_coord128 >= 80) &&
          (x_coord128 <= 86) &&
          (y_coord64 <= 6))
         pen_size = 1;
      if (y_coord64 < 11)
         continue;
      switch (pen_size) {
         case 1 : Glcd_Dot(x_coord128, y_coord64, write_erase);
                  break;
         case 2 : Glcd_Box(x_coord128, y_coord64,
                           x_coord128 + 1, y_coord64 + 1,
                           write_erase);
                  break;
         case 3 : Glcd_Box(x_coord128-1, y_coord64-1,
                           x_coord128 + 2, y_coord64 + 2,
                           write_erase);
                  break;
      }
   }
}
```

Несмотря на внушительные размеры С-кода программы, ее содержание вполне понятно. По этой причине разъяснять его суть мы не будем, надеясь, что читателю уже вполне по силам разобраться с ним самостоятельно (тем более, что в программе переведены комментарии). Заметим только, что здесь использованы уже знакомые нам функции работы с АЦП, графическим дисплеем, а также ряд новых функций, информацию о которых с той или иной степенью успеха можно найти в справке.

Клавиатура PS/2

В системе Flowcode мы создали проект, позволяющий вводить данные в МК с помощью 12-кнопочной клавиатуры. Возможности такого ввода, естественно, не велики. В MikroC разработаны средства для использования полноценной компьютерной клавиатуры при ее подключении к разъему PS/2.

Основным рабочим инструментом здесь является следующая функция:

```
Ps2_Key_Read(unsigned short *value, unsigned short *special,
             unsigned short *pressed).
```

Она возвращает значение 1, если символ был успешно считан с клавиатуры, и 0 в противном случае. Код прочитанного символа записывается по адресу, указанному в качестве первого аргумента. По адресу, указанному в виде второго аргумента, записывается 0, если была нажата обычная алфавитно-цифровая клавиша, и 1 в случае нажатия специальной клавиши (например, <F1>). Наконец, по адресу третьего аргумента размещается 1, если в данный момент клавиша нажата, и 0, если клавиша отпущена.

В проекте PS_2 (листинг 3.12) опрашивается клавиатура, и при нажатии клавиши соответствующий символ высвечивается на ЖК-дисплее.

Листинг 3.12. Взаимодействие с клавиатурой через порт PS/2

```
void main()
{
 unsigned short keydata=0,special=0,down=0,i=1;
 CMCON=0x07;
 INTCON=0;
 Ps2_Init(&PORTC);
 Delay_ms(100);
 Lcd_Config(&PORTB,4,5,6,3,2,1,0);
 LCD_Cmd(LCD_CLEAR);
 LCD_Cmd(LCD_CURSOR_OFF);
 do
 {
  if (Ps2_Key_Read(&keydata,&special,&down)&& down)
   Lcd_Chr(1,i++,keydata);
  if(i>16) i=1;
  Delay_ms(10);
 } while (1);
}
```

Во второй и третьей строках главной функции отключается так называемая аналоговый компаратор и запрещаются все прерывания — необходимые установки конфигурации. В программе инициализируется клавиатура и дисплей, после чего организуется бесконечный цикл. Если в цикле обнаружено нажатие клавиши, то символ считывается и передается на ЖК-дисплей, где он выводится в очередной позиции первой строки. Если строка полностью записана, то вывод возвращается к начальной позиции.

Для успешной работы проекта подтягивающие резисторы порта С необходимо установить в положение “PullUp”. Необходимо также выключить все светодиоды, семисегментные индикаторы и источники на-

пряжения для аналоговых входов. Четыре старших позиции SW6 должны быть включены.

Обмен данными по интерфейсу RS232

Микроконтроллеры PIC поддерживают разнообразные коммуникационные интерфейсы. С одним из них: OneWire — мы уже познакомились. Очень распространена и передача данных через уже знакомый нам по предыдущей главе интерфейс RS232. В данном контексте RS232 следует рассматривать просто как кабельное соединение, а за фактическую организацию приема и передачи данных отвечает встроенный в МК аппаратный модуль USART. Под аббревиатурой USART следует понимать "универсальный последовательный асинхронный приемопередатчик". Именно им мы и воспользуемся в следующей программе. Данные через USART передаются байтами, хотя каждый байт передается последовательно, бит за битом. По этой причине главной характеристикой USART является скорость передачи, измеряемая, например, в бодах.

Применив только что опробованные функции работы с клавиатурой, создадим проект, который по RS232 передает символы, нажимаемые на подключенной к МК клавиатуре, в ПК. Компьютер соединяется с платой EasyPIC5 нуль-модемным кабелем. В качестве программы для приема и отображения информации будем использовать уже знакомый нам HyperTerminal. Текст программы (проект `RS232_Key`) представлен в листинге 3.13.

Листинг 3.13. Программа для обмена данными по интерфейсу RS232

```
void main()
{
  unsigned short keydata=0,special=0,down=0;
  CMCON  = 0x07;
  INTCON =  0;
  Usart_Init(19200);
  Ps2_Init(&PORTC);
  Delay_ms(100);
  do
  {
   if (Ps2_Key_Read(&keydata,&special,&down))
    {
     if (down && (keydata == 13))    // Enter
      {
        Usart_Write('\r');
        Usart_Write('\n');
      }
      else if (down && !special && keydata)
```

Листинг 3.13. Окончание

```
        Usart_Write(keydata);
    }
    Delay_ms(10);
  } while (1);
}
```

Новыми для нас являются функции `Usart_Init` и `Usart_Write`. Первая из них инициализирует приемопередатчик на работу со скоростью 19 200 бод, а вторая выдает символ (в данном случае — в интерфейс RS232), который по нуль-модемному кабелю поступает в ПК. Передаваемый символ считывается с клавиатуры, как в предыдущем проекте. Если клавиатура передала символ с кодом 13 (нажатие клавиши <Enter>), то на ПК будет отправлено два специальных символа: `\r` и `\n`. Читатель должен помнить, что это — символы возврата каретки и перехода на новую строку. Таким образом, следующие символы будут выведены с начальной позиции в новой строке. Где они будут выведены? В окне программы HyperTerminal. Имейте в виду, что по умолчанию терминал настроен на низкую скорость приема данных. Измените ее на 19 200. И еще одно... Кроме установок переключателей и перемычек,, необходимых для работы с клавиатурой, следует включить младшие позиции SW7 и SW8.

Следующий проект `PC_PIC_PC` реализует двухстороннюю связь между МК и ПК. На прилагаемом к книге компакт-диске находится программа для ПК `COM_PIC.exe` и библиотека к ней `SerialGate.dll`. Для успешной работы оба файла должны находиться в одном каталоге на диске компьютера. Эта программа может принимать данные в виде целого однобайтового числа и передавать такие числа по интерфейсу RS232.

В микроконтроллер же необходимо записать программу, представленную в листинге 3.14.

Листинг 3.14. Программа для двухсторонней связи МК с ПК

```
void main()
{
 unsigned short i;
 USART_init(9600);
 PORTB = 0;
 TRISB = 0;
 while (1)
  {
   if (USART_Data_Ready())
    {
     i = USART_Read();
```

Листинг 3.14. Окончание

```
      USART_Write(i);
      PORTB=i;
     }
   }
}
```

USART инициализируется для работы на скорости 9 600 бод. Порт B настраивается на вывод. Организуется бесконечный цикл. Система ждет получения очередного байта извне. Как только это происходит, функция `USART_Data_Ready` возвращает 1. При этом в переменную `i` считывается значение полученного байта, которое тут же отправляется обратно в USART и далее через RS232 — в ПК. Одновременно на светодиодах порта B высвечивается двоичный код полученного байта.

Теперь запустите программу `COM_PIC.exe` на ПК. Появится окно, показанное на рис. 3.8.

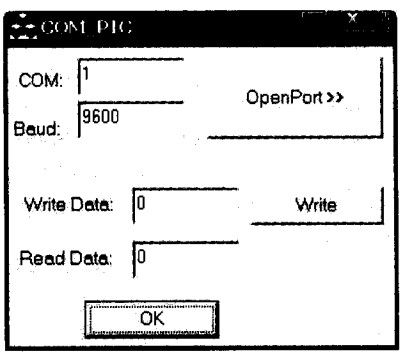


Рис. 3.8. Программа COM_PIC

Здесь следует выбрать COM-порт, к которому подключен нуль-модемный кабель (по умолчанию COM1) и скорость передачи данных (по умолчанию 9 600 бод). Нажмите кнопку **OpenPort**. Если с кабелем и портом все в порядке, то система подтвердит успешное открытие порта. Теперь в поле **Write Data** можно ввести любое число в диапазоне от 1 до 255 и нажать кнопку **Write**. На плате EasyPIC5 светодиоды порта B просигнализируют о приеме байта. Байт вернется обратно в компьютер, и в поле **Read Data** отобразится посланное число.

Звук

Микроконтроллер можно генерировать звуковые сигналы. В качестве устройства излучения звука используется так называемый пьезодинамик — небольшая "таблетка", подключаемая к "земле" и выходному выводу одного из портов. Этот аксессуар продается в магазинах электронных компонентов. Разумеется, звук, излучаемый такой "таблеткой", — далеко не HiFi, и отличается очень неравномерной частотной характеристикой. Тем не менее, в качестве инструмента звуковой индикации тех или иных событий пъезо-динамик находит применение, например, в охранных системах.

Звуковая библиотека MikroC содержит функцию Sound_Init, инициализирующую работу со звуком. Он принимает два аргумента: адрес порта, на который будет выдаваться звуковой сигнал, и номер вывода этого порта, с которого сигнал считывается. Собственно звуковой сигнал генерирует функция Sound_Play. Первый ее параметр — частота звука, измеряемая в герцах, а второй — длительность звучания сигнала, измеряемая в миллисекундах.

В листинге 3.15 дан текст программы (проект Sound), взятый из русской справки по MikroC (см. прилагаемый к книге компакт-диск).

Листинг 3.15. Программа генерирования звукового сигнала

```
unsigned adc_value;

void main()
{
  PORTB  = 0;
  TRISB  = 0;
  INTCON = 0;
  ADCON1 = 0x82;
  TRISA  = 0xFF;
  Sound_Init(&PORTB, 0);
  do
 {
    adc_value = Adc_Read(2);
    Sound_Play(adc_value, 100);
  } while (1);
}
```

Порт В устанавливается на вывод, запрещаются все прерывания, Четвертое выражение в главной функции предписывает работать со вторым каналов АЦП. Зачем? Ответ будет дан чуть позже. Порт А устанавливается на ввод. Звуковой сигнал будет выдаваться на нулевой вывод порта В.

Далее организуется бесконечный цикл, в котором оцифровывается входной аналоговый сигнал, и полученный уровень (от 0 до 1 024 — 10-разрядная оцифровка) передается в качестве первого параметра в функцию Sound_Play. Таким образом, вращая ручку потенциометра АЦП, мы меняем частоту звука от 0 до 1 024 Гц. Для того чтобы услышать эту "музыку", необходимо подключить пьезо-динамик к "земле" и выводу 0 порта В.

Внимание! Не пытайтесь заменить пьезо-динамик обычным динамиком или телефонным наушником. Наша "таблетка", по сути, является конденсатором с очень большим входным сопротивлением. Попытка

подключить на выход порта низкоомную нагрузку может просто вывести из строя микроконтроллер.

Фирменный пример от Mikroelektronika предлагает нечто более интересное, чем простое звучание какого-то тона. В листинге 3.16 представлен текст проекта Sound_Var из дистрибутива MikroC. Вообще говоря, ничего принципиально нового здесь для нас нет. Просто с помощью все той же функции Sound_Play создаются функции более высокого уровня Tone и Melody, позволяющие воспроизвести более разнообразные звуковые фрагменты.

Листинг 3.16. Пример генерирования звука

```
void Tone1() {
  Sound_Play(659, 250);  // Частота 659 Гц, длительность 250 мс
}
void Tone2() {
  Sound_Play(698, 250);  // Частота 698 Гц, длительность 250 мс
}
void Tone3() {
  Sound_Play(784, 250);  // Частота 784 Гц, длительность 250 мс
}

void Melody() {            // Мелодия "Yellow house"
  Tone1(); Tone2(); Tone3(); Tone3();
  Tone1(); Tone2(); Tone3(); Tone3();
  Tone1(); Tone2(); Tone3();
  Tone1(); Tone2(); Tone3(); Tone3();
  Tone1(); Tone2(); Tone3();
  Tone3(); Tone3(); Tone2(); Tone2(); Tone1();
}

void ToneA() {
  Sound_Play( 880, 50);
}
void ToneC() {
  Sound_Play(1046, 50);
}
void ToneE() {
  Sound_Play(1318, 50);
}

void Melody2() {
  unsigned short i;
  for (i = 9; i > 0; i--) {
    ToneA(); ToneC(); ToneE();
  }
}
```

Листинг 3.18. Окончание

```
void main() {
  ADCON1 = 0x07;    // Конфигурируем все ан. выводы как цифровые
  TRISB  = 0xF0;    // RB7..RB4 - входы, RB3 - выход
  Sound_Init(&PORTB, 3);
  Sound_Play(1000, 500);
  while (1) {
    if (Button(&PORTB,7,1,1))    // RB7 играет Tone1
      Tone1();
    while (PORTB & 0x80) ;       // Ожидаем отпускания кнопки
    if (Button(&PORTB,6,1,1))    // RB6 играет Tone2
      Tone2();
    while (PORTB & 0x40) ;       // Ожидаем отпускания кнопки
    if (Button(&PORTB,5,1,1))    // RB5 играет Melody2
      Melody2();
    while (PORTB & 0x20) ;       // Ожидаем отпускания кнопки
    if (Button(&PORTB,4,1,1))    // RB4 играет Melody
      Melody();
    while (PORTB & 0x10) ;       // Ожидаем отпускания кнопки
  }
}
```

Единственное, что здесь требует некоторого разъяснения, — это функция `Button`. Не вдаваясь в детали, ограничимся тем, что эта функция проверяет, нажата ли кнопка, связанная определенным выводом заданного порта с учетом эффекта дребезга контактов. При нажатии одной из кнопок 4–7 порта B воспроизводится некоторая звуковая последовательность, заданная функциями `Melody` и `Tone`. Цикл вида `while (PORTB & 0x10);` повторяется до тех пор, пока не будет отпущена ранее нажатая кнопка.

Прерывания

В данном разделе поговорим о прерываниях более подробно. Прерывание определим как событие приостановки выполнения текущей программы с последующим переходом к особой подпрограмме. Как только подпрограмма обработки прерывания выполнена, возобновляется основная программа с точки, где она была прервана.

Основные источники прерываний в микроконтроллерах:

- переполнения счетного буфера таймера;
- изменение состояния определенного разряда определенного порта (в подавляющем большинстве случаев — разряд 0 порта B);
- завершение обслуживания (чтения или записи) памяти данных EEPROM;

- любое изменение состояния определенного порта (опять-таки, в подавляющем большинстве случаев речь идет о порте B).

Мы рассмотрим программирование первых двух типов прерываний, начав со второго, которое принято обозначать как INT/RB0.

Для разрешения использования INT/RB0 необходимо установить в 1 старший (GIE) разряд регистра специальных функций INTCON. Этим мы "разрешаем разрешить" любые прерывания. Для того чтобы разрешить конкретно прерывание INT/RB0, необходимо установить в 1 разряд 4 (INTE) того же регистра.

Теперь, как только на выводе RB0 появится сигнал, будет вызвана подпрограмма обработки прерывания. На время ее работы все прерывания блокируются обнулением старшего разряда регистра INTCON (это происходит автоматически). Одновременно устанавливается в 1 так называемый флаг запроса на прерывание (разряд 1, INTF). Перед выходом из подпрограммы обработки данный разряд необходимо опять обнулить. Это автоматически не происходит, что должно быть предусмотрено в подпрограмме. После возврата в основную программу разряд GIE автоматически восстанавливается в 1, что дает возможность отслеживать последующие прерывания. Подпрограмма обработки прерываний любого типа в MikroC должна быть оформлена в виде функции типа `void` с предопределенным именем `interrupt`.

Проект `INT_RB0` реализует следующий алгоритм. После запуска программа начинает выполнять бесконечный пустой цикл. При нажатии на плате EasyPIC5 кнопки, связанной с выводом 0 порта B, возникает прерывание, в результата чего содержимое порта D инвертируется, а счетчик количества прерываний увеличивается на 1. Начальное состояние счетчика равно нулю, исходный код на порте D — 0x0F. Содержимое счетчика отображается светодиодам порта C в двоичном виде. Также отображается и код порта D. Реализация алгоритма представлена в листинге 3.17.

Листинг 3.17. Обработка прерывания INT/RB0

```
unsigned short count=0;

void interrupt()
{
  PORTD=~PORTD;
  PORTC=++count;
  INTCON.INTF = 0;
}

void main(){
```

Листинг 3.17. Окончание

```
  TRISB = 0xFF;
  TRISD = 0;
  PORTD = 0x0F;
  TRISC = 0;
  PORTC = 0x00;
  INTCON = 0x90;
  while(1);
}
```

В обработчике прерываний операция `PORTD=~PORTD` означает, что новое значение получается из старого путем инвертирования, когда все единицы превращаются в нули и наоборот. Конструкция `INTCON.INTF` определяет конкретный разряд регистра специальных функций. В главной функции порт B конфигурируется на ввод. Подтягивающие резисторы необходимо установить в состояние "PullDown". Порты C и D назначаются выходными. Все присваиваемые значения должны быть понятны из приведенного выше описания функционирования прерывания.

Прерывание от таймера наступает в момент переполнения его буфера (регистр TMR0). Значения буфера увеличивается на единицу в каждом цикле работы системы. В микроконтроллере PIC16F877A буфер — восьмиразрядный. Таким образом, если в обработчике прерываний каждый раз устанавливать TMR0 в некоторое значение от 0 до 255, то этим значением мы регулируем частоту следования прерываний. Кроме того, для работы таймера можно использовать так называемый предделитель частоты. Его назначение — передавать на таймер не каждый такт, а каждый второй, каждый четвертый и т.д. вплоть до каждого 256-го такта.

Работой таймера управляет регистр специальных функций OPTION (в MikroC обозначается как `OPTION_REG`). Три младших разряда этого регистра устанавливают значение предделителя от 1/2 тактовой частоты при 000 до 1/256 при 111. Старший разряд рекомендуется устанавливать в 1. Остальные разряды по умолчанию сброшены в 0, что соответствует штатным условиям работы с таймером 0.

Проект `Tmr0` включает и гасит светодиоды порта B с заданной частотой, используя прерывания от таймера 0. Соответствующая программа дана в листинге 3.18.

Листинг 3.18. Обработка прерывания от таймера 0

```
unsigned cnt;

void interrupt()
{
```

Листинг 3.18. Окончание

```
 cnt++;
 TMR0   = 0x00;
 INTCON = 0x20;
}

void main()
{
 OPTION_REG = 0x87;
 TRISB = 0;
 PORTB = 0xFF;
 TMR0  = 0x00;
 INTCON = 0xA0;
 cnt = 0;
 do
  {
   if (cnt == 32)
    {
      PORTB = ~PORTB;
      cnt = 0;
    }
  } while(1);
}
```

Начнем с главной функции... Значение 0x87 для `OPTION_REG` задает установку старшего разряда в 1 и коэффициент деления частоты 1/256. Порт B объявляется выходным с начальным состоянием 0xFF, т.е. все светодиоды включены. Регистр таймера устанавливается в 0. Состояние INTCON соответствует разрешению прерываний именно от таймера 0. Счетчик `cnt` обнуляется. Если состояние порта B будет изменяться при каждом прерывании от таймера, то мигание будет слишком частым. По этой причине мы создали счетчик `cnt`, который инкрементируется каждым прерыванием, однако порт B модифицируется только по достижении `cnt` значения 32. После этого счетчик опять обнуляется. Все это помещено в бесконечный цикл.

Функция обработки прерывания увеличивает счетчик на единицу и обнуляет регистр таймера. Последнее в данном случае можно было бы и не делать, поскольку переполнение само переводит регистр в ноль. Данная операция была добавлено исключительно для иллюстративных целей. Необходимость такого действия возникает, если регистр таймера необходимо устанавливать не в ноль. Наконец, требуется восстановить в 1 значение шестого разряда регистра INTCON — разрешение прерываний от таймера 0.

После запуска МК в работу все светодиоды порта В начнут попеременно включаться и гаснуть. В данном варианте проекта параметры подобраны так, что при частоте осциллятора 8 МГц цикл включения/гашения составит примерно одну секунду.

В современных микроконтроллерах доступно несколько таймеров. Изучить работу с другими таймерами можно по справочной литературе или по примерам, подготовленным, в частности, компанией Mikroelektronika. Мы же рассмотрим только сторожевой таймер (Watch Dog Timer — WDT).

Назначение сторожевого таймера — вывод МК из внештатной ситуации, когда рабочая программа по тем или иным причинам зацикливается. Что в этом случае можно сделать? Да только одно: перезапустить систему. Итак, если разрешено прерывание от строжевого таймера, то по переполнению его регистра произойдет полный сброс системы и ее повторный запуск "с нуля". Подчеркнем, что перезапуск требуется именно при зацикливании. Если программа функционирует нормально, то в ней необходимо предусмотреть сброс регистра сторожевого таймера прежде, чем он будет переполнен.

Следующий проект `WatchDog_Timer` целиком взят из примеров к MikroC. В программе, показанной в листинге 3.19, сброс сторожевого таймера не предусмотрен, поэтому он периодически переполняется, что приводит к перезапуску программы. Уточним, что здесь не требуется никакой функции-обработчика. Все происходит автоматически.

Листинг 3.19. Реализация сброса от сторожевого таймера

```
void main() {
  OPTION_REG = 0x0E; // Назначаем предделитель сторожевому
                     // таймеру, коэффициент 1:64
  asm CLRWDT;        // Ассебмлерная команда - обнуление
                     // сторожевого таймера
  PORTB = 0x0F;      // Инициализация порта B
  TRISB = 0;         // Конфигурируем порт B как выход
  Delay_ms(300);     // Ожидаем 0,3 с
  PORTB = 0xF0;      // Изменяем значение для порта B
  while (1);         // Бесконечный цикл. Сторожевой таймер
                     // сбросит PIC
}
```

В программе в регистре OPTION устанавливается разряд 3. Это означает, что предделитель назначается не таймеру 0, а сторожевому таймеру. Коэффициент деления устанавливается в 1/64 (младшие разряды содержат 110). Обратите внимание, что шкала делителя здесь отличается от шкалы для таймера 0. Там 110 соответствует коэффициенту 1/128.

Далее исключительно в учебных целях вставлена строка ассемблерного кода, начинающаяся с инструкции `asm`. Сама команда ассемблера проста: `CLRWDT` — обнуление регистра сторожевого таймера. Устанавливаются в единицы четыре младших разряда порта B. Порт B определяет как выходной. Выдерживается пауза в 300 мс. Устанавливаются четыре старших разряда порта.

После включения микроконтроллера начнут поочередно инвертироваться светодиоды на старших и младших разрядах порта B. Казалось бы, — обычная работа программы, однако, следует помнить, что здесь все время происходит перезапуск программы, как по нажатию красной кнопки **RESET** на плате EasyPIC5.

ШИМ

Эта аббревиатура расшифровывается как "широтно-импульсная модуляция" (PWM) и обозначает процесс аппаратного (т.е. независимо от выполнения программы) формирования микроконтроллером последовательности прямоугольных импульсов.

Для инициализации генератора используется функция `PWM1_Init`, аргументом которой является частота следования импульсов. Диапазон допустимых частот зависит от типа МК. Так, для PIC16F877A нижняя граница частоты составляет 490 Гц. Функция `PWM1_Change_Duty` своим аргументом задает скважность сигнала. Значение аргумента может меняться в диапазоне от 0 до 255. В частности, значение 127 задает скважность 50%, когда длительность импульса равна длительности паузы. Генератор активизируется вызовом функции `PWM1_Start`, а останавливается с помощью функции `PWM1_Stop`. Последовательность импульсов выдается на вывод 2 порта C. Программа проекта `Pwm` представлена в листинге 3.20.

Листинг 3.20. Реализация ШИМ

```
void main()
{
  PORTB=0;
  TRISB=0;
  PWM1_Init(490);
  PWM1_Change_Duty(127);
  PWM1_Start();
  while (1) PORTB.F0=PORTC.F2;
}
```

Выражение присваивания в цикле `while` передает сигнал с вывода 2 порта C на вывод 0 порта B. Мы предполагаем, что к этому выводу

подключен пьезо-динамик. В этом случае мы услышим монохроматичный звук. Если же пьезо-динамик подключить к порту С, то цикл `while` вообще можно сделать пустым. Мы избрали вариант с перенаправлением сигнала, прежде всего, для демонстрации работы с отдельными разрядами портов.

Память внутренняя и внешняя

Последний раздел этой главы мы посвятим работе с внутренней и внешней памятью МК. Сразу же отметим, что реализация работы с памятью в MikroC, на наш взгляд, далеко не безупречна. Во-первых, не в полной мере унифицированы функции для различных типов МК. Во-вторых, функции работают не всегда так, как описано в справочной системе. И вообще, обнаруживаются некоторые странности, на которых акцентировать внимание мы не будем, однако читатель должен быть готов к встрече с ними. Тем не менее, представленные ниже примеры вполне работоспособны

К внутренней памяти, аппаратно реализованной в МК, относится оперативная Flash-память и долгосрочная (энергонезависимая) память EEPROM.

Проект `Flash_Write` демонстрирует запись данных в Flash-память и чтение из этой памяти (листинг 3.21).

Листинг 3.21. Работа с Flash-памятью

```
unsigned int i,addr,dataout,datain[5][4]={{0,1,2,3},{4,5,6,7},
                    {8,9,10,11},{12,13,14,15},{16,17,18,19}};
void main()
{
 PORTB = 0;
 TRISB = 0;
 addr = 0x00;
 Delay_ms(200);
 for (i = 0; i < 5; i++)
  {
   Delay_ms(20);
   Flash_Write(addr+i*4, datain[i]);
  }
 addr = 0x00;
 for (i = 0; i < 20; i++)
  {
   dataout = Flash_Read(addr++);
   PORTB = dataout;
   Delay_ms(500);
  }
```

Функция Flash_Write в качестве первого аргумента принимает адрес Flash-памяти, начиная с которого будет записана текущая порция данных. Адрес указывается в собственной "системе отсчета" безотносительно нумерации байтов в общей памяти МК. Второй параметр является указателем на записываемую порцию данных уже в стандартной памяти программ. В представленной программе мы записываем в Flash-память двухмерный массив из пяти строк и четырех столбцов. Запись осуществляется построчно в цикле. Поскольку строка содержит четыре элемента, приращение адреса записи составляет 4. Как известно, согласно синтаксису языка С, datain[i] является начальным адресом i-й строки.

Функция Flash_Read считывает данные по адресу, указанному в ее аргументе. В нашем случае чтение производится в цикле поэлементно, поэтому приращение адреса составляет 1. Прочитанное значение отображается двоичным кодом на светодиодах порта B. Одна из вышеупомянутых странностей заключается в том, что попытка поэлементной записи не приводит к желаемому результату.

Проект Eeprom_Write демонстрирует работу с памятью EEPROM (листинг 3.22). Здесь операции подобны тем, которые были реализованы в предыдущей программе.

Листинг 3.22. Работа с памятью EEPROM

```
void main()
{
 int i;
 PORTB = 0;
 TRISB = 0;
 for (i = 0; i < 32; i++)
  EEprom_Write(i,i);
 Delay_ms(50);
 for (i = 0; i < 32; i++)
  {
   PORTB = EEPROM_Read(i);
   Delay_ms(200);
  }
}
```

Впрочем, между функциями EEprom_Write и Flash_Write существует одно значительное отличие. При записи в EEPROM второй аргумент — это не адрес записываемой порции данных, а само значение. Все остальное в данной программе понятно и комментариев не требует.

Стоит, пожалуй, отметить, что допустимое количество операций перезаписи памяти EEPROM на порядки меньше, чем для памяти про-

грамм (Flash). В связи с этим злоупотреблять манипуляциями с памятью EEPROM не рекомендуется.

Если необходимо обрабатывать и хранить большие объемы данных, одной внутренней памятью микроконтроллера не обойтись. В MikroC представлены средства для работы с внешней памятью, в частности, — с картами памяти MMC и Compact Flash (CF). Мы рассмотрим последний вариант. Сразу отметим, что успешная работа с CF возможна на МК серии PIC18 и, возможно, — старших серий. По этой причине два представленных ниже проекта ориентированы на PIC18F4550. Адаптер CF является дополнительным модулем и подключается к портам B и D.

Использование CF возможно в двух вариантах. В первом карта памяти представляет собой просто хранилище байтов информации, практически не структурированное. Единственной структурной единицей здесь является блок емкостью 512 байт. При этом блок — неделимая единица, т.е. при записи необходимо заполнить все 512 байт блока. Во втором варианте CF представляет собой полноценную файловую систему с таблицей размещения файлов FAT16.

Проект `Cf-sector` работает с секторами памяти карты. В программе, представленной в листинге 3.23, в сектор под номером 591 в его начальные байты записываются символы слова "Mikroelektronika". Остальные байты сектора заполняются символом "7". Информация о готовности карты и успешном чтении данных отправляется через USART и RS232 терминальной программе на ПК.

Листинг 3.23. Работа с картой CF

```
void Send_String(char *ostr)
{
  unsigned short i=0;
  while (ostr[i]) USART_Write(ostr[i++]);
}

void main()
{
 char text[]="Mikroelektronika";
 unsigned int i;
 Usart_Init(19200);
 Delay_ms(50);
 CF_Init(&PORTB,&PORTD);
 while (CF_Detect() == 0);
 Send_String("\n\rCF Ready");
 Delay_ms(10);
 CF_Write_Init(591,1);
 Delay_ms(10);
 for (i=0; i<512; i++)
```

Листинг 3.23. Окончание

```
  if(i<16) CF_Write_Byte(text[i]);
  else CF_Write_Byte('7');
 Delay_ms(10);
 CF_Read_Init(591,1);
 Send_String("\n\r");
 for (i=0; i<32; i++)
  {
    Usart_Write((char)CF_Read_Byte());
    Delay_ms(10);
  }
}
```

Адаптер карты на плате EasyPIC5 следует подключать к портам B и D. Работа с CF инициализируется с помощью функции `CF_Init`, аргументами которой являются адреса портов. Строка

```
while (CF_Detect() == 0);
```

организует цикл до момента поступления подтверждения об обнаружении карты функцией `CF_Detect`. Функция `Send_String` передает строку "\n\rCF Ready" на терминал, начиная с новой строки, с первой позиции. Функция `CF_Write_Init(591,1)` подготавливает CF к записи в сектор 591, начиная с первого байта. Собственно запись на карту реализует функция `CF_Write_Byte`. Из названия функции видно, что она записывает байт, передаваемый ей в качестве аргумента. По окончании записи карта инициализируется на чтение. Считываются первые 32 байта сектора, после чего прочитанные символы передаются в ПК. Понятно, что вначале мы там увидим слово "Mikroelektronika", а сразу за ним — 32 символа "7".

Чтение и запись происходят не мгновенно. По этой причине задержки `Delay`, которые присутствуют в программе, оказываются действительно необходимыми.

Для работы с файловой системой на карте Compact Flash в MikroC разработано большое количество функций, которые обеспечивают полный набор операций с файлами и каталогами. Знакомство с этими инструментами заняло бы десятки страниц, и потому рассматривать из здесь мы не будем. Предоставляем читателю самому поработать со справочной системой. Ограничимся лишь рассмотрением программы из проекта `Cf_FAT16_test`, любезно предоставленной автору компанией Mikroelektronika (листинг 3.24). При этом отметим еще одну странность. Хотя тактовая частота осциллятора на плате EasyPIC5 составляет 8 Гц, в свойствах проекта она должна быть установлена равной 48 Гц!

Листинг 3.24. Работа с файловой системой FAT16

```
/*
 * Project name:
     Cf_Fat16_Test (демонстрация использования библиотеки
                    Cf_Fat16)
 * Copyright:
     (c) MikroElektronika, 2005-2008
 * Описание:
     Этот проект состоит из нескольких блоков, демонстрирующих
     различные аспекты использования библиотеки Cf_Fat16:
     - создание нового файла и запись в него;
     - открытие существующего файла и его перезапись;
     - открытие существующего файла и добавление к нему данных;
     - открытие файлы и чтение данных из него (передача на
       терминал USART);
     - создание и модификация одновременно нескольких файлов.
 * Тестовая конфигурация:
     MK:               PIC18F4550
     Плата разработки: EasyPIC5
     Осциллятор:       HS, 08.000 МГц
     Внешние модули:   карта CF на портах B и D
     SW:               mikroC v8.0
 * Примечания:
     - Перед работой с этим примером убедитесь, что карта CF
       правильно отформатирована (FAT16 или просто FAT)!
     - Эта библиотека - только для микроконтроллера PIC18!
     - Этот пример предполагает, что карта CF будет подключена
       до сброса, иначе отобразится сообщение FAT_TXT!!!
 */

 #include "built_in.h"

char
 FAT_TXT[20] = "FAT16 not found",
 file_contents[50] = "XX CF FAT16 library by Anton Rieckert\n";
char
 filename[14] = "MIKRO00xTXT";           // Имена файлов
unsigned short
 tmp, caracter, loop, loop2;
unsigned long
 i, size;
char Buffer[512];

//-------------- Записывает строку в USART
void I_Write_Str(char *ostr) {
  unsigned short i;
  i = 0;
```

Листинг 3.24. Продолжение

```
  while (ostr[i]) {
    USART_Write(ostr[i++]);
  }
  USART_Write(0x0A);
}

//-------------- Создает новый файл и записывает в него данные
void M_Create_New_File() {
  filename[7] = 'A';
  Cf_Fat_Assign(&filename, 0xA0);    // Не обнаружив файл,
                                        создает его
  Cf_Fat_Rewrite();                 // Очищает файл и начинает
                                    // записывать новые данные
  for(loop = 1; loop <= 99; loop++) {  // Нужны 5 файлов на
                                       // карте MMC
    file_contents[0] = loop / 10 + 48;
    file_contents[1] = loop % 10 + 48;
    Cf_Fat_Write(file_contents, 38); // Запись данных
                                     // в назначенный файл
    Usart_Write('.');
  }
}

//---- Создает несколько новых файлов и записывает в них данные
void M_Create_Multiple_Files() {
  for(loop2 = 'B'; loop2 <= 'Z'; loop2++) {
    usart_write(loop2);
    filename[7] = loop2;               // Установка имени файла
    Cf_Fat_Assign(&filename, 0xA0);    // Ищем существующий файл
                                       // или создаем новый
    Cf_Fat_Rewrite();                 // Очищает файл и начинает
                                      // записывать новые данные
    for(loop = 1; loop <= 44; loop++) {
      file_contents[0] = loop / 10 + 48;
      file_contents[1] = loop % 10 + 48;
      Cf_Fat_Write(file_contents, 38); // Запись данных
                                       // в назначенный файл
    }
  }
}

//-------------- Открывает и перезаписывает существующий файл
void M_Open_File_Rewrite() {
  filename[7] = 'G';
  Cf_Fat_Assign(&filename, 0);
  Cf_Fat_Rewrite();
```

Листинг 3.24. Продолжение

```
  for(loop = 1; loop <= 55; loop++) {
    file_contents[0] = loop / 10 + 64;
    file_contents[1] = loop % 10 + 64;
    Cf_Fat_Write(file_contents, 38);    // Запись данных
                                        // в назначенный файл
  }
}

//-------------- Открывает существующий файл и добавляет в него
//               данные (и изменяет дату и время модификации)
void M_Open_File_Append() {
     filename[7] = 'F';
     Cf_Fat_Assign(&filename, 0);
     Cf_Fat_Set_File_Date(2005,6,21,10,35,0);
     Cf_Fat_Append();   // Подготовка файла к добавлению данных
     // Запись данных в назначенный файл
     Cf_Fat_Write(" for mikroElektronika 2005\n", 27);
}

//-------------- Удаляет файл. Если файл не существует, то он
//               будет вначале создан, а затем удален.
void M_Delete_File() {
  filename[7] = 'B';
  Cf_Fat_Assign(filename, 0);
  Cf_Fat_Delete();
}

//-------------- Открывает существующий файл, считаывает из
//               него данные и выдает их в USART
void M_Open_File_Read() {
  filename[7] = 'F';
  Cf_Fat_Assign(&filename, 0);
  Cf_Fat_Reset(&size);              // Для чтения файла процедура
                                    // возвращает его размер
  for (i = 1; i <= size; i++) {
    Cf_Fat_Read(&caracter);
    Usart_Write(caracter);          // Запись данных в USART
  }
}

//------------- Проверяет, существует ли файл. Если существует,
//              передает его дату создания и размер через USART
void M_Test_File_Exist() {
  unsigned long fsize;
  unsigned int year;
  unsigned short month, day, hour, minute;
```

Листинг 3.24. Продолжение

```
  unsigned char outstr[12];

  filename[7] = 'F';        //Эта строка должна быть не
      //закомментирована, чтобы искать файл, который СУЩЕСТВУЕТ
//  filename[7] = 'B';        //Эта строка должна быть не
   //закомментирована, чтобы искать файл, который НЕ СУЩЕСТВУЕТ
  if (Cf_Fat_Assign(filename, 0)) {
    //--- Файл найден. Извлекаем его дату
    Cf_Fat_Get_File_Date(&year, &month, &day, &hour, &minute);
    WordToStr(year, outstr);
    I_Write_Str(outstr);
    ByteToStr(month, outstr);
    I_Write_Str(outstr);
    WordToStr(day, outstr);
    I_Write_Str(outstr);
    WordToStr(hour, outstr);
    I_Write_Str(outstr);
    WordToStr(minute, outstr);
    I_Write_Str(outstr);
    //--- get file size
    fsize = Cf_Fat_Get_File_Size();
    LongToStr((signed long)fsize, outstr);
    I_Write_Str(outstr);
  }
  else {
    //--- Файл не найден. Извещаем об этом
    Usart_Write(0x55);
    Delay_ms(1000);
    Usart_Write(0x55);
  }
}

//-------------- Пытается создать файл подкачки с размером хотя
//               бы 100 секторов (подробности см. в справке)
void M_Create_Swap_File() {
  unsigned int i;
  for(i=0; i<512; i++)
    Buffer[i] = i;
  // Подробности о следующей функции см. в справке
  size = Cf_Fat_Get_Swap_File(5000, "mikroE.txt", 0x20);
  if (size) {
    LongToStr((signed long)size, fat_txt);
    I_Write_Str(fat_txt);
    for(i=0; i<5000; i++) {
      Cf_Write_Sector(size++, Buffer);
      Usart_Write('.');
```

Листинг 3.24. Окончание

```
    }
  }
}

//-------------- Главная функция. Снимите комментарии с тех
//               функций, которые хотите проверить
void main() {
    ADCON1 = 0x0F;           // Настройка аналоговых линий на
                             // цифровой ввод-вывод
    Usart_Init(19200);       // Настройка USART на чтение файлов
    Delay_ms(100);
    // Если необходимо форматирование, используйте вместо
    // функции init быстрое форматирование fat16
    if(!Cf_Fat_Init(&PORTB, &PORTD)) {     // Инициализация
                                           // библиотеки FAT
      //--- Начало проверки
      Usart_Write('s');
      M_Create_New_File();
      M_Create_Multiple_Files();
      M_Open_File_Rewrite();
      M_Open_File_Append();
      M_Open_File_Read();
      M_Delete_File();
      M_Test_File_Exist();
      M_Create_Swap_File();
      //--- Окончание проверки
      Usart_Write('e');
    }
    else {
     I_Write_Str(FAT_TXT);
    }
}
```

Глава 4

MPLAB и программирование на ассемблере

Итак, мы подошли к наиболее эффективному, но и наиболее сложному подходу к разработке проектов для микроконтроллеров: программированию на языке низкого уровня ассемблере. В основе такого подхода лежит использование команд МК, представленных в главе 1. Наиболее популярный инструмент программирования в данном случае — пакет MPLAB от компании Microchip. Его подробное описание заняло бы не один десяток страниц, однако наша цель — ознакомиться с азами программировании, и потому мы, конечно же, не будем давать здесь исчерпывающий разбор возможностей MPLAB. Затронем лишь некоторые примеры, сопроводив их необходимыми пояснениями, чего для начинающих будет вполне достаточно.

Первая программа. Включаем светодиоды

В проекте `111` реализуем простую идею: включим четыре из восьми светодиодов на порте B микроконтроллера. Прежде всего, запустите программу MPLAB и выберите в главном меню команду **Project ▶ Project Wizard**. Появится окно приветствия, в котором следует просто нажать кнопку **Далее**. В следующем окне, показанном на рис. 4.1, выберите в раскрывающемся списке тип микроконтроллера. В нашем случае это — все тот же PIC16F877A.

В следующем окне (рис. 4.2) необходимо выбрать рабочий компилятор. В MPLAB при стандартной инсталляции устанавливается несколько компиляторов, в том числе, и для языка C. Нас интересует компилятор MPASM, который предлагается по умолчанию.

Нажмите кнопку **Далее**, чтобы перейти в окно выбора имени проекта и папки его размещения. Допустим, проект назван `111`, а размещается на диске `C`, в каталоге `TMP`. В таком случае окно мастера должно соответствовать рис. 4.3.

В следующем окне (рис. 4.4) можно сразу же присоединить к проекту заранее подготовленный файл (например, текстовый файл с ассемблерным кодом программы).

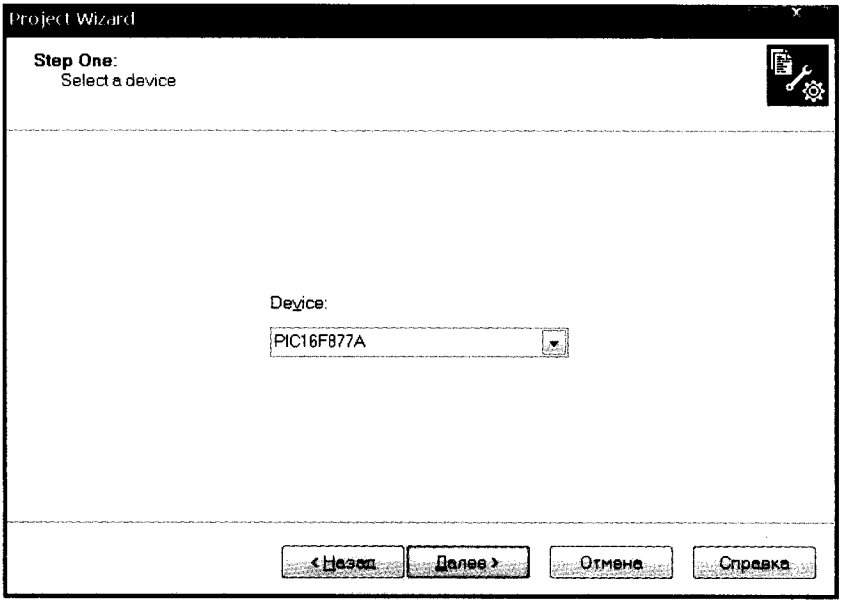


Рис. 4.1. Первое окно мастера проектов MPLAB

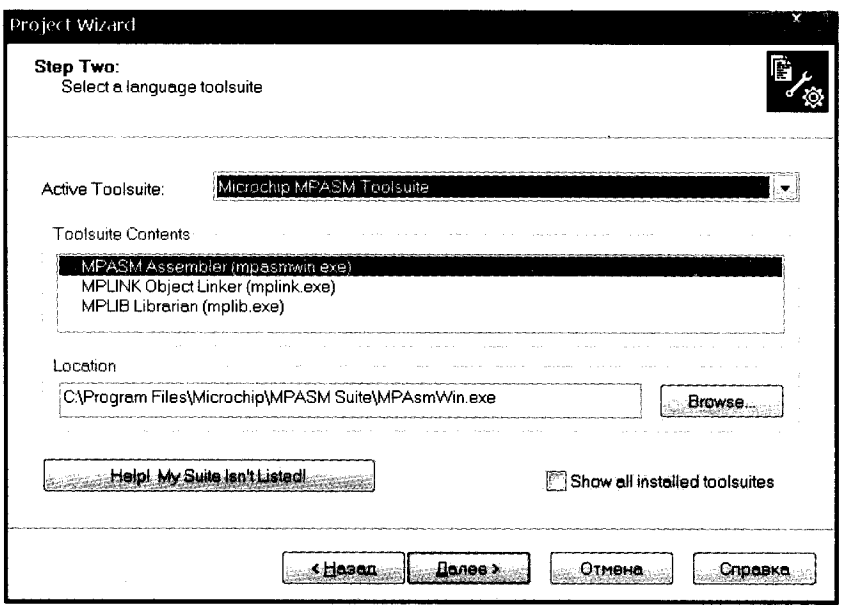


Рис. 4.2. Выбор типа компилятора

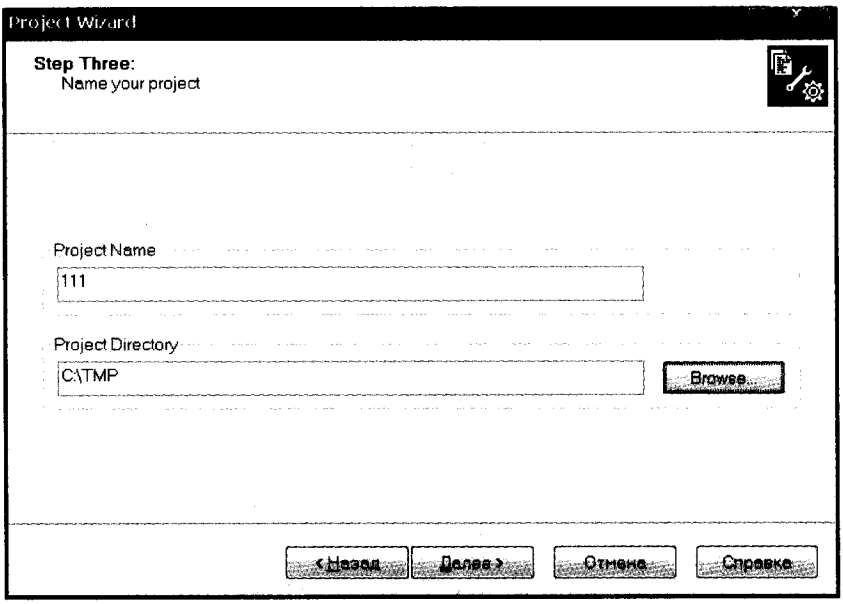


Рис. 4.3. Ввод имени проекта и папки его размещения

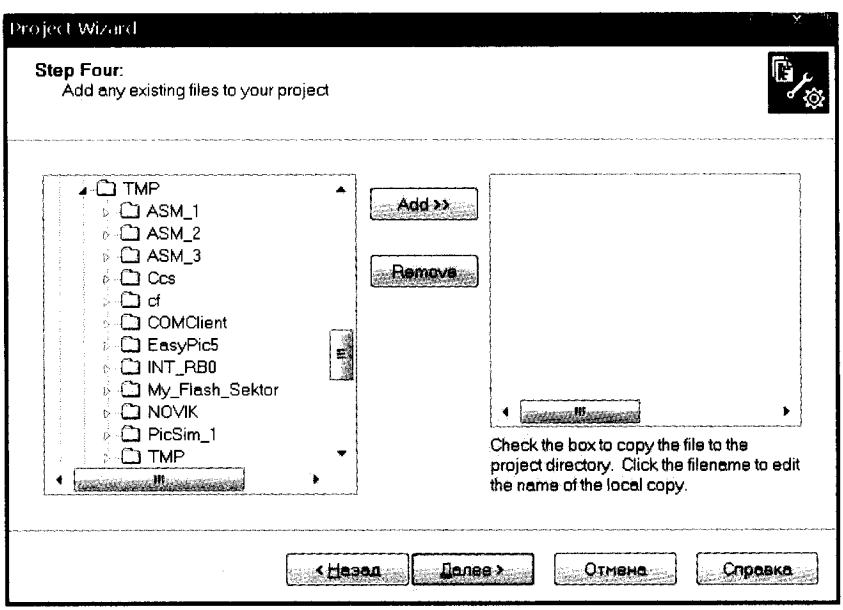


Рис. 4.4. Присоединение к проекту файлов

Заранее подготовленного файла у нас нет, поэтому мы переходим к следующему этапу. В последнем окне мастера просто представлена сводка предыдущих настроек проекта. Нажмите кнопку **Готово**, чтобы вернуться в главное окно MPLAB.

Теперь необходимо создать текст ассемблерного кода. В главном меню выберите команду **File ► New**. На экране появится пустое окно текстового редактора. Введите в нем текст, показанный на рис. 4.5.

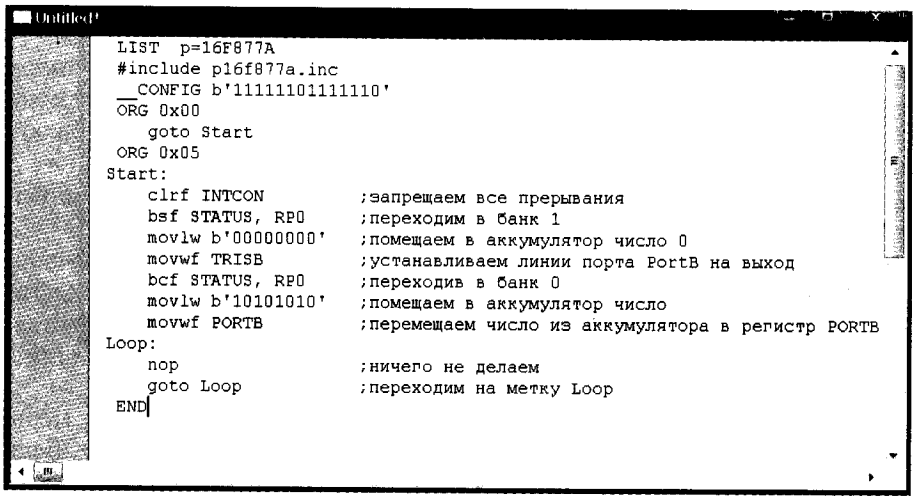


Рис. 4.5. Текст ассемблерной программы

Комментарии (текст, расположенный в строке справа от точки с запятой ";") вводить, конечно же, не обязательно, однако они полезны для понимания сути программного кода. Обратим внимание, что ассемблер не различает регистра символов. Применение прописных и строчных букв в данном примере обусловлено исключительно желанием повысить наглядность кода.

Далее стандартным образом (через команду меню **File ► Save as**) сохраните файл в требуемой папке под именем `111.asm` (рис. 4.6). Обратите внимание, что после сохранения вид текста в редакторе изменился. Теперь различные конструкции в программе обозначены разными цветами. Ассемблер различает команды, операнды, комментарии и т.д.

Файл `111.asm` необходимо присоединить к проекту. Для этого выберите команду меню **Project ►Add Files to Project**. Откроется стандартное диалоговое окно выбора файла. После присоединения файла имеет смысл сохранить проект с помощью команды меню **Project ► Save Project**.

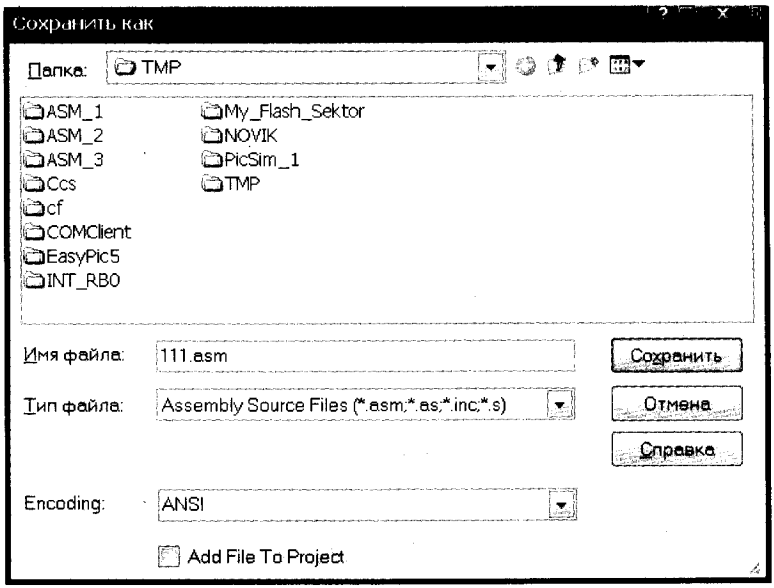


Рис. 4.6. Сохранение файла с ассемблерным кодом

Наступил важный момент: сборка проекта. Выберите команду меню **Project ▶ Build All** или нажмите комбинацию клавиш <Ctrl+F10>. Если все сделано правильно, то проект будет успешно скомпилирован и скомпонован. В результате в окне **Output** в последней строке появится нечто похожее на "BUILD SUCCEEDED: Thu May 21 10:57:36 2009".

На данном этапе закроем MPLAB, подтвердив запросы на все сохранения, и посмотрим, что у нас получилось. Просмотрите содержимое папки проекта. Как видим, в ней появилось довольно много разных файлов. Здесь есть и сам текст программы `111.asm`, и конечный продукт `111.hex` (исполняемый файл, знакомый нам по предыдущим главам). Запишите HEX-файл в микроконтроллер, чтобы убедиться, что цель достигнута: светодиоды светятся (четыре из восьми, через один, на порте B).

В папке присутствует главный файл проекта с расширением `.mcp`, связанный с приложением MPLAB. Присутствуют тут и другие файлы, которые для начинающих пользователей особой важности не представляют.

А теперь приступаем к самому главному: изучению ассемблерной программы. Прежде всего, отметим, что основными конструкциями исходного кода могут быть директивы ассемблера, метки, команды и комментарии. При этом имеет значение порядок их следования в строках.

Метки (например, `Start:`) и директивы (например, `#include p16f877a.inc`) рекомендуется размещать с первой (крайней слева) позиции в строке. Команды располагаются со второй или более правой позиции, а комментарии — в любой позиции, начиная с символа ";" и до конца строки. Нарушение этих правил в MASM вызывает предупреждение, а в ряде других компиляторов вообще недопустимо. Напомним также, что ассемблер не чувствителен к регистру символов, т.е. не различает прописные и строчные буквы. А теперь — к делу!

В первой строке нашей программы указана директива `LIST` и ее параметр `p=16F877A`. Тут все ясно: задается тип микроконтроллера. Параметр в данном случае один. Если бы их было несколько, то они отделялись бы запятыми. Например, в конце первой строки можно добавить: `,r=HEX`. Это будет означать, что в числовых константах по умолчанию используется шестнадцатеричная система счисления.

Следующая строка — тоже директива. Она подключает к тексту программы все, что содержится в указанном файле. Если мы заглянем в файл `p16f877a.inc` (находится в одной из подпапок программного каталога `Microchip`), то увидим в нем фрагмент, представленный в листинге 4.1.

Листинг 4.1. Начало файла `p16f877a.inc`

```
      LIST
; P16F877A.INC  Standard Header File, Version 1.00
; Microchip Technology, Inc.
      NOLIST
; This header file defines configurations, registers, and other
; useful bits of information for the PIC16F877A
; microcontroller. These names are taken to match the data
; sheets as closely as possible. Note that the processor must
; be selected before this file is included. The processor may
; be selected the following ways:
;   1. Command line switch:
;            C:\ MPASM MYFILE.ASM /PIC16F877A
;   2. LIST directive in the source file
;            LIST   P=PIC16F877A
;   3. Processor Type entry in the MPASM full-screen interface
;==========================================================
;       Revision History
;==========================================================
;Rev:    Date:     Reason:
;1.03   11/17/05 Added the ADCON1 bit ADCS2.
;1.02  05/28/02 Corrected values for _CP_ALL and _CP_OFF in
;Configuration Bits section.
;1.01   09/13/01 Added the PIR2 bit CMIF and the PIE2 bit CMIE
```

Листинг 4.1. Окончание

```
;1.00   04/19/01 Initial Release (BD - generated from
; PIC16F877.inc)
;==========================================================
;       Verify Processor
;==========================================================
        IFNDEF __16F877A
        MESSG "Processor-header file mismatch.  Verify selected
processor."
        ENDIF
;==========================================================
;       Register Definitions
;==========================================================
W                               EQU     H'0000'
F                               EQU     H'0001'
;----- Register Files------------------------------------------
INDF                            EQU     H'0000'
TMR0                            EQU     H'0001'
PCL                             EQU     H'0002'
STATUS                          EQU     H'0003'
FSR                             EQU     H'0004'
PORTA                           EQU     H'0005'
PORTB                           EQU     H'0006'
PORTC                           EQU     H'0007'
PORTD                           EQU     H'0008'
PORTE                           EQU     H'0009'
PCLATH                          EQU     H'000A'
```

Здесь, в частности, присутствуют конструкции типа `PORTB EQU H'0006'`. Такая конструкция позволяет использовать вместо адреса его псевдоним. В данном случае адрес регистра порта B заменяется именем `PORTB`, что более удобно при написании и чтении текста программы.

В третьей строке программы опять находится директива `__CONFIG`. Разговор о ней — большой и серьезный. С помощью этой директивы выполняется настройка МК. Для этого задаются значения (0 или 1) так называемых разрядов конфигурации. Таких разрядов всего 14. Их обозначения представлены на рис. 4.7.

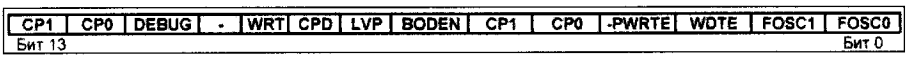


Рис. 4.7. Разряды конфигурации

Далее мы приводим описание этих разрядов по официальной документации к микроконтроллеру:

- разряды 13–12 (CP1:CP0) — защита памяти программ;
- разряд 11 (DEBUG) — включения внутрисхемной отладки;

- 1 — внутрисхемная отладка отключена; выводы RB6 и RB7 работают как каналы ввода-вывода;
- 0 — внутрисхемная отладка включена; выводы RB6 и RB7 используются отладчиком;

- разряд 10 — не реализован (всегда "1");
- разряд 9 (WRT) — разрешение записи в Flash-память программ:
  - 1 — запись в Flash-память программ через регистры управления EECON разрешена;
  - 0 — запись в Flash-память программ через регистры управления EECON запрещена;
- разряд 8 (CPD) — защита памяти данных EEPROM;
  - 1 — защита памяти данных выключена;
  - 0 — защита памяти данных включена;
- разряд 7 (LVP) — разрешение низковольтного программирования:
  - 1 — вывод RB3/PGM работает как PGM; режим низковольтного программирования включен;
  - 0 — вывод RB3/PGM работает как цифровой порт ввода-вывода; вывод HV используется для программирования МК;
- разряд 6 (BODEN) — разрешение сброса по снижению напряжения питания:
  - 1 —сброс разрешен;
  - 0 — сброс запрещен;
- разряды 5–4:
  - 11 — защита памяти программ выключена;
  - 10 — защита памяти программ с адресами 1F00h–1FFFh;
  - 01 — защита памяти программ с адресами 1000h–1FFFh;
  - 00 — защита памяти программ с адресами 0000h–1FFFh;
- разряд 3 (PWRTE) — разрешения работы таймера включения питания:
  - 1 — таймер выключен;
  - 0 — таймер включен;
- разряд 2 (WDTE) — разрешение работы сторожевого таймера:
  - 1 — сторожевой таймер включен;
  - 0 — сторожевой таймер выключен;
- разряды 1–0 (FOSC1: FOSC0) — выбор режима тактового генератора:
  - 11 — RC;
  - 10 — HS;
  - 01 — XT;
  - 00 — LP.

Наверное, не все здесь понятно. Ничего страшного! Начинающему и не требуется знать сразу все. По мере необходимости читатель сможет подробнее разобраться с разрядами конфигурации самостоятельно, используя дополнительные источники информации.

Аргумент директивы `b'11111101111110'` представляет собой 14-разрядное двоичное число, разряды которого соответствуют разрядам конфигурации. В частности, два младших разряда в комбинации `10` соответствуют режиму генератора HS, как это и должно быть для нашей системы на плате EasyPIC5. Нетрудно разобраться и со всеми настройками, представленными в данном варианте. Аргумент директивы можно записать более компактно, прейдя от двоичной системы к шестнадцатеричной. В этом случай третья строка выглядела бы как `__CONFIG 0x3F7E`.

Следующая строка `ORG 0x00` — тоже директива ассемблера. Понимать ее следует так: "Следующая после директивы команда (а это будет именно команда) должна быть размещена по нулевому адресу памяти программ". Сама команда `goto Start` предписывает перейти на метку `Start`. Метки адресов не имеют или, можно сказать, что метка имеет тот же адрес, что и следующая за ней или в этой же строке команда. У нас перед меткой расположена директива `ORG 0x05`. Это значит, что команда `clrf INTCON` должна быть размещена по адресу 5. Зачем и почему? Дело в том, что по адресу 4 записывается так называемый вектор прерываний. Если в программе используется механизм прерываний, то этот вектор трогать нельзя. Потому мы его и обошли. Правда, в данной программе этого можно было бы и не делать, поскольку мы прерываниями здесь не пользуемся.

Команда `clrf` очищает (обнуляет) целиком весь регистр, адрес которого указан в операнде (см., например, описание команды в главе 1). В файле `p16f877a.inc` адрес закодирован именем `INTCON`. Не вдаваясь в детали, поясним, что обнуление этого регистра запрещает все прерывания. Сделать это необходимо уже потому, что у нас включен сторожевой таймер, который может генерировать прерывания.

Порты ввода-вывода, используемые в программе (точнее, выводы портов), должны быть настроены либо на ввод, либо на вывод. Для этого необходимо установить соответствующие разряды регистров TRIS. Однако для получения доступа, например, к регистру TRISB необходимо перейти из банка памяти 0 в банк 1. Для этого устанавливается в 1 разряд RP0 регистра STATUS. Данную задачу в нашей программе реализует команда `bsf STATUS, RP0`. После перехода в банк 0 можно занести во все разряды регистра TRISB нули.

Напрямую записать константу в регистр специального назначения нельзя. Операция разбивается на два этапа. Вначале константа записывается в рабочий регистр w (аккумулятор): `movlw b'00000000'`. Затем содержимое аккумулятора копируется в требуемый регистр: `movwf TRISB`. В результате порт B настроен на вывод.

Возвращаемся в банк 0, обнулив разряд RP0 регистра STATUS: `bcf STATUS, RP0`. Здесь уместно взглянуть на мнемонику (имена) команд. Присутствие символов `bs` можно расшифровать, как "bit set": "установить разряд" (в 1), — а символы `bc` расшифровываются как "bit clear": "очистить разряд" (обнулить).

Следующие две команды (`movlw b'10101010'` и `movwf PORTB`), как легко понять, заносят комбинацию нулей и единиц в порт B. Именно в этот момент включаются светодиоды, подключенные к тем выводам порта B, на которых установились единицы.

Три следующие строки переводят программу в бесконечный цикл. Вначале размещается метка `Loop` (можно указать с двоеточием, а можно и без). В теле цикла выполняется команда `nop`, которая ничего не делает. Команда `goto Loop` предает управление назад на метку. Тело цикла могло быть вообще пустым, т.е. команду `nop` можно убрать. Небольшая разница заключается в том, что команда `nop` занимает во времени один машинный цикл (четыре такта работы тактового генератора).

И последнее, что должно присутствовать в конце каждой программы, — директива ассемблера `END`.

Программирование задержек

Усложним задачу. Пусть светодиоды порта B не просто горят, а мигают с некоторым интервалом. Другими словами, некоторые группа светодиодов должны попеременно включаться и гаснуть. Для этого нам необходимо научиться программировать паузы в работе программы. Идея здесь достаточно проста. В требуемой точке программы мы организовываем цикл или несколько вложенных друг в друга циклов, которые ничего не выполняют, а только "тратят время зря". Циклы могут быть действительно пустыми или содержать необходимое количество команд `nop`. Текст программы (проект `222`) представлен в листинге 4.2.

Листинг 4.2. Программа с использованием задержек

```
LIST  p=16F877A
 #include p16f877a.inc
 __CONFIG b'11111101111110'
      COUNT1 EQU 0x0E
      COUNT2 EQU 0x0F
```

Листинг 4.2. Окончание

```
      ORG 0x00
      GOTO Start
      ORG 0x05
Start:
      CLRF INTCON          ;запрещаем все прерывания
      BSF STATUS, RP0      ;переходим в банк 1
      MOVLW b'00000000'    ;помещаем в аккумулятор число 0
      MOVWF TRISB   ;устанавливаем линии порта PortB на вывод
      BCF STATUS, RP0      ;переходим в банк 0
Loop;
      MOVLW b'10101010'    ;помещаем в аккумулятор число
      MOVWF PORTB   ;перемещаем число из аккумулятора в PORTB
      NOP                  ;ничего не делаем
      CALL DELAY_D
      MOVLW b'01010101'    ;помещаем в аккумулятор число
      MOVWF PORTB   ;перемещаем число из аккумулятора в PORTB
      NOP           ;ничего не делаем
      CALL DELAY_D
      GOTO Loop            ;переходим на метку Loop
DELAY_D
      MOVLW .255
      MOVWF COUNT2
      MOVWF COUNT1
loop_1:
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      DECFSZ COUNT2,1
      GOTO loop_1
      DECFSZ COUNT1,1
      GOTO loop_1
      RETURN
 END
```

В начале программы нам все знакомо за исключением двух строк: `COUNT1 EQU 0x0E` и `COUNT2 EQU 0x0F`. Здесь адреса*м* присваиваются имена все с той же целью: упростить чтение программы.

После конфигурирования порта B на вывод организуется бесконечный цикл с возвратом на метку `Loop`, в котором и реализованы все основные действия программы. В этом цикле в порт B через аккумулятор заносится двоичный код из чередующихся нулей и единиц, который сразу же высвечивается светодиодами. Следующая команда `nop` совсем не обязательна и поставлена просто для отделения одной группы действий от другой.

Далее следует команда `CALL DELAY_D`. По сути, здесь вызывается подпрограмма с именем `DELAY_D`, которая ничего не делает, а просто "впустую тратит время". Другими словами, программа реализует паузу некоторой длительности. После паузы в аккумулятор, а затем — и в порт B записывается набор нулей и единиц, инвертированный относительно первоначального. И опять вызывается подпрограмма задержки. Все это зацикливается между меткой `Loop` и оператором `GOTO Loop`. Таким образом, цель достигнута: светящиеся группы светодиодов чередуются во времени.

Метку `DELAY_D` следует рассматривать как начало тела подпрограммы. Здесь вначале в аккумулятор, а затем — в переменные `COUNT1` и `COUNT2` записывается число 255. Точка перед константой в операнде означает, что число представлено в десятичной системе счисления. Вместо этого в операнде мы могли бы записать `0xFF` или, например, `b'11111111'`.

Далее установлена метка `Loop_1` и следует 16 команд `nop`. Эта последовательность займет 16 машинных циклов или 64 такта тактового генератора. При частоте осциллятора 8 МГц данная цепочка будет выполняться 64/8000000 с = 8 мкс. По истечении этого времени сработает команда `DECFSZ COUNT2,1`, которая уменьшает на единицу счетчик `COUNT2`. Команда устроена таким образом, что при достижении счетчиком нуля счетчик адреса команд принудительно увеличится не на 1, а на 2. Это означает, что до обнуления счетчика `COUNT2` вслед за командой будет выполняться следующая по программе команда: `GOTO Loop_1`.

Мы 255 раз выполним цикл `Loop_1` – `GOTO Loop_1`. Очевидно, что на это уйдет 8*255 мкс = 2,04 мс. Впрочем, это не совсем так, поскольку и сам переход занимает время в несколько тактов. Но мы пока что это учитывать не будем, поскольку высокая точность расчета времени нам не требуется. Далее можно увидеть, что внутренний цикл оказался помещенным во внешний цикл, где инкрементируется уже счетчик

COUNT1. Выполнение внешнего цикла займет 2,04*255 мс = 0,5202 с. По окончании внешнего цикла по команде RETURN происходит выход из подпрограммы в основную программу. Итак, мы получили подпрограмму задержки длительностью около половины секунды. В результате наши светодиоды будут мигать с частотой около 1 Гц, в чем можно убедиться, запустив программу на исполнение на плате EasyPIC5.

Для скрупулезного расчета времени необходимо, как уже говорилось, учесть время исполнения команды GOTO (выполняется не за один, а за два машинных цикла, т.е. восемь тактов). Заметим, что во многих практических задачах использования микроконтроллеров (например, в прецизионных измерениях), такой учет крайне необходим.

Отладка программы

Как и многие системы высокоуровневого программирования, пакет MPLAB предоставляет средства отладки. Мы рассмотрим здесь только пошаговое выполнение программы в режиме эмуляции с отслеживанием изменений переменных.

Представленный ниже текст программы проекта 333 (листинг 4.3) был взят нами из статьи, опубликованной в сети Internet.

Листинг 4.3. Пример из Internet-статьи

```
        LIST  p=16F877
        include<p16f877a.inc>
        __CONFIG _HS_OSC & _WDT_OFF
;=================================================
; Инициализация переменных в памяти данных
;=================================================
DelH                    equ     0x20
DelM                    equ     0x21
DelL                    equ     0x22
;=================================================
; Начало программы
;=================================================
        ORG 0x00
        goto Start
        ORG 0x05
Start:
        clrf INTCON             ;запрещаем все прерывания
;=================================================
; Настраиваем линии порта PORTB на выход
;=================================================
        bsf STATUS, RP0         ;переходим в банк 1
        movlw B'00000000'       ;помещаем в аккумулятор число 0
        movwf TRISB   ;устанавливаем линии порта PortB на вывод
```

Листинг 4.3. Окончание

```
        bcf STATUS, RP0      ;переходим в банк 0
;=============================================
; Переходим к основному циклу программы
;=============================================
        clrf PORTB           ;очищаем регистр порта PORTB
        nop                  ;ничего не делаем
        nop
        nop
Loop:
        movlw .2             ;заносим в аккумулятор число 2
        movwf DelH           ;копируем содержимое аккумулятора в
                             ;  регистр DelH
        movlw .2             ;заносим в аккумулятор число 2
        movwf DelM           ;копируем содержимое аккумулятора в
                             ;  регистр DelM
        movlw .5             ;заносим в аккумулятор число 5
        movwf DelL           ;копируем содержимое аккумулятора в
                             ;  регистр DelL
        nop
Delay_L:
        decfsz DelL,1        ;уменьшаем содержимое регистра DelL
                             ;  и сохраняем результат в регистре
                             ;  DelL, если содержимое регистра
                             ;  равно 0, то пропускаем команду
        goto Delay_L         ;переходим на метку Delay_L
Delay_M:
        decfsz DelM,1        ;уменьшаем содержимое регистра DelM
                             ;  и сохраняем результат в регистре
                             ;  DelM, если содержимое регистра
                             ; равно 0, то пропускаем команду
        goto Delay_L         ;переходим на метку Delay_L
Delay_H:
        decfsz DelH,1        ;уменьшаем содержимое регистра DelH
                             ;  и сохраняем результат в регистре
                             ;  DelH, если содержимое регистра
                             ;  равно 0, то пропускаем команду
        goto Delay_L         ;переходим на метку Delay_L
        incf PORTB,1         ;увеличиваем содержимое регистра
                             ;  PORTB и сохраняем
                             ;  результат в регистре PORTB
        goto Loop            ;переходим на метку Loop
        End
```

Скомпилировав проект и загрузив исполняемый файл в микроконтроллер, можно увидеть, что светодиодами порта B последовательно во времени высвечиваются коды двоичных чисел от 1 до 255 в бесконеч-

ном цикле. Индикация числа меняется с некоторой задержкой, которая организована здесь несколько иначе, чем в предыдущем примере (об этом — чуть позже).

Обратите внимание, что во второй строке директива `include` построена не так, как раньше: нет символа #, а имя файла взято в угловые скобки <>. Такие "вольности" в некоторых конструкциях допустимы, хотя это, пожалуй, — не совсем правильно. В третьей строке директива конфигурации тоже выглядит иначе, чем в предыдущей программе. Здесь вместо двоичного кода, соответствующего разрядам конфигурации, помещен текст `_HS_OSC & _WDT_OFF`. Смысл написанного понятен. Здесь объединены (операцией `&`) две опции: выбирается режим осциллятора HS и отключается сторожевой таймер. Надо понимать так, что остальные разряды конфигурации остаются со значениями, заданными по умолчанию. Таким образом, зная символические названия системных констант типа `_WDT_OFF`, можно разрабатывать конфигурационные директивы.

В программе задаются три переменные: `DelH`, `DelM` и `DelL`, — которым в начале цикла `Loop` присваиваются десятичные значения 2, 2 и 5 соответственно. Далее от метки `Delay_L` до перехода `goto Delay_L` расположен цикл, формирующий задержку. Читателю предлагается внимательно изучить этот блок программы и убедиться, что здесь на самом деле — три вложенных друг в друга цикла. В общих чертах здесь все происходит примерно так же, как и в предыдущем примере, однако тройное вложение позволяет, в принципе, организовать более длительные паузы и без многократного использования команды `nop`. Кроме того, здесь цикл задержки находится в сам*ой* основной программе, а не оформлен подпрограммой, как в предыдущем варианте.

По завершении тройного цикла инкрементируется содержимое регистра порта B, и светодиоды высвечивают очередное двоичное число. Если дождаться того, что в регистре PORTB сформируется число 255, то в результате переполнения в нем установится 0, и все начнется сначала.

Теперь организуем обещанное пошаговое выполнение программы. В главном окне MPLAB с открытым проектом `333` выберите пункт меню **Debugger ► Select Tool ► MPLAB SIM**. Это означает, что отладка будет осуществлена в симуляторе MPLAB. Перекомпилируйте проект (<Ctrl+F10>) и еще раз откройте меню **Debugger**. Как видим, его состав изменился. В нем появились разнообразные возможности, среди которых нам необходимо выбрать пошаговое исполнение **Step Into** (клавиша <F7>). В текстовом редакторе с программой появится зеленая стрелка,

указывающая на текущую команду. Очередное нажатие <F7> приводит к переходу к следующей инструкции.

Выберите пункт меню **View ▶ Watch**. Откроется окно, которое служит для отображения текущих значений объектов программы. Допустим, мы хотим отслеживать изменения переменной `DelL`. Выберите из раскрывающегося списка **Add Symbol** требуемое имя и нажмите кнопку **Add Symbol**. В результате переменная `DelL` появляется в списке просматриваемых объектов (рис. 4.8).

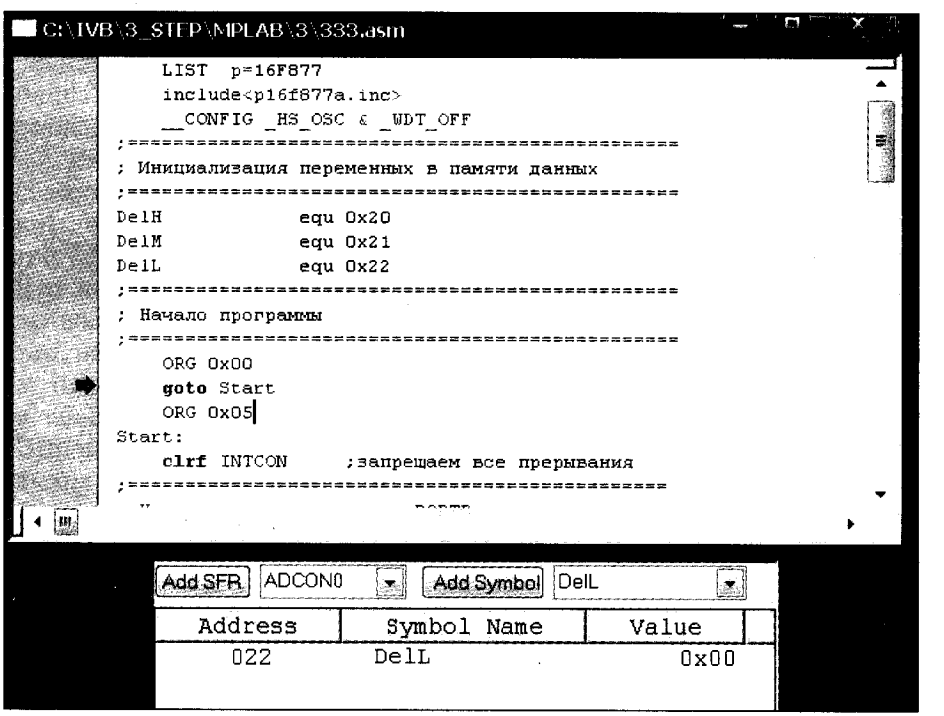


Рис. 4.8. Просмотр значения переменной `DelL`

Как видим, адрес переменной — `022`, а ее значение — `0x00`. Нажимая последовательно клавишу <F7>, мы доберемся до команд, которые будут изменять значение переменной `DelL`. В этом и заключается процесс отслеживания изменений в объектах программы. Кроме переменных, можно наблюдать и за содержимым регистров специальных функций. Для этого следует выбрать необходимый элемент в раскрывающемся списке **Add SFR** и нажать кнопку **Add SFR**. С остальными возможностями отладки читатель может ознакомиться самостоятельно.

Управление счетчиком команд

В следующем проекте (444) мы выведем на семисегментный индикатор произвольную цифру. Для этого будет использован прием, который доступен только в низкоуровневом ассемблерном программировании: управление счетчиком команд (листинг 4.4).

Листинг 4.4. Вывод цифры на семисегментный индикатор

```
LIST  p=16F877A
 #include p16f877a.inc
 __CONFIG b'11111101111110'
 ORG 0x00
       goto Start
 ORG 0x05
Start:
    clrf INTCON          ;запрещаем все прерывания
    bsf STATUS, RP0      ;переходим в банк 1
    movlw b'00000000'    ;помещаем в аккумулятор число 0
    movwf TRISD          ;устанавливаем линии порта D на вывод
    bcf STATUS, RP0      ;переходим в банк 0
Loop:
    MOVLW 0x09
CALL SEG
    MOVWF PORTD
    goto Loop            ;переходим на метку Loop
SEG
    ANDLW 0x0F
    ADDWF PCL,1
    RETLW 0x3F
    RETLW 0x06
    RETLW 0x5B
    RETLW 0x4F
    RETLW 0x66
    RETLW 0x6D
    RETLW 0x7D
    RETLW 0x07
    RETLW 0x7F
    RETLW 0x6F

 END
```

Работа с семисегментным индикатором на плате EasyPIC5 реализована через порт D, поэтому в начале программы устанавливаем его на вывод. Все, что написано до начала цикла Loop, нам уже знакомо.

В цикле Loop – goto Loop вначале в аккумулятор записывается число 9 — та самая цифра, которую мы хотим отобразить на индикато-

ре. Можно заменить ее любой другой десятичной цифрой. Далее вызывается подпрограмма SEG, и в порт D записывается значение, сформированное в аккумуляторе в результате работы подпрограммы.

Что же выполняет подпрограмма SEG? Это — самое главное. Прежде всего, заметим, что любая десятичная цифра от 0 до 9 занимает в двоичном коде не более четырех младших разрядов. По этой причине, во избежание случайных ошибок (например, в команде MOVLW 0x09 вместо 0x09 может оказаться 0x19), принудительно обнуляются четыре старших разряда аккумулятора. Младшие разряды должны оставаться неизменными. Для этого служит команда логического поразрядного перемножения аккумулятора с константой 0x0F, т.е. с двоичным кодом 00001111. Результат необходимо сохранить в аккумуляторе. Все это реализует команда ANDLW 0x0F.

Инструкция ADDWF PCL,1 складывает содержимое регистра PCL с содержимым аккумулятора и сохраняет результат в PCL. Теперь вспомним, что PCL — это младший байт счетчика команд (старший байт не трогаем, поскольку не собираемся далеко уходить от текущего адреса). Что же получилось? Адрес команды, которая будет выполнена немедленно, принудительно увеличился на содержимое аккумулятора. Если там, например, 0, то следующей выполнится команда, размещенная сразу после ADDWF PCL,1, т.е. RETLW 0x3F. Если в аккумуляторе 9 (как в нашем примере), то необходимо отсчитать вниз девять инструкций. Выполнится команда RETLW 0x6F. Инструкция RETLW предписывает выход из подпрограммы с записью значения операнда в аккумулятор. Все команды RETLW в нашей подпрограмме имеют операнды: соответствующие комбинации разрядов, высвечивающих требуемые сегменты на индикаторе.

Стоит напомнить, что для реализации проекта на плате EasyPIC5 необходимо включить все или только необходимые переключатели из четырех старших позиций на SW6; подтягивающие резисторы порта B включить и установить в состояние "Pull-down"; включить одну или обе перемычки входов АЦП (j15, j16).

Работа с массивами

Работа с массивами в ассемблерной программе организована неожиданным с точки зрения высокоуровневого программирования способом. Прежде всего, уточним, что речь может идти о двух типов массивов: в памяти программ (ПЗУ, ROM) и в памяти данных (регистровой памяти, ОЗУ, RAM).

Поскольку программа для микроконтроллера прошивается в ПЗУ, если в ней предусмотрено использование массивов, то они должны быть созданы (и, самое главное, инициализированы) непосредственно в программе. Вот здесь на помощь опять приходит программное изменение счетчика команд. Каким бы странным это ни показалось, следующая подпрограмма сама по себе представляет массив из шести элементов, инициализированных числами 7, 3, 1, 2, 255 и 255 (последние два представлены в шестнадцатеричной системе счисления).

```
MasROM
      addwf   PCL,1
      retlw  7
      retlw  3
      retlw  1
      retlw  2
      retlw  0xff
      retlw  0xff
```

Если в главной программе перед вызовом подпрограммы `MasROM` в аккумулятор записать, например, 3, то в теле подпрограммы счетчик адреса PCL увеличится на три и после команды `addwf PCL,1` выполнится команда `retlw 2`. При этом произойдет выход из подпрограммы с записью в аккумулятор числа 2. Таким способом можно считать в аккумулятор любой элемент массива.

Для работы с массивом в регистровой памяти обычно используется так называемая косвенная адресация. Существует специальный регистр косвенной адресации (индексный регистр) FSR. Если в нем содержится некоторый ненулевой адрес, то изменение числа в аккумуляторе приводит к немедленному копированию этого числа в регистр ОЗУ, адрес которого указан в FSR.

Теперь рассмотрим проект `555`, в котором массив создается в памяти программ, копируется в память ОЗУ (т.е. в массив в памяти данных), один из его элементов считывается из ОЗУ в порт B, и его значение высвечивается на светодиодах (листинг 4.5).

Листинг 4.5. Использование массива в памяти программ

```
      list p=16f877a
      include<p16f877a.inc>
      __CONFIG b'11111101111110'
    org     0x00
MasRAM       equ    0x20          ;Резервирование памяти в ОЗУ
   tmp       equ    MasRAM +6
counter      equ    tmp+1
      clrf INTCON          ;запрещаем все прерывания
      bsf STATUS, RP0      ;переходим в банк 1
```

Листинг 4.5. Окончание

```
        movlw b'00000000'   ;помещаем в аккумулятор число 0
        movwf TRISB  ;устанавливаем линии порта PortB на вывод
        bcf STATUS, RP0     ;переходив в банк 0
        movlw MasRAM ;запись в аккумулятор адрес начала массива
        movwf FSR           ;подготовились к косвенной адресации
        clrf   tmp          ;очистка индекса элемента массива
        movlw 6             ;готовим счетчик цикла counter
        movwf counter
Loop                        ;основной цикл
        movf tmp,0
        call MasROM          ;выбрали из ROM очередной элемент
        movwf 0             ;переслали его косвенно в RAM
        incf tmp,1          ;увеличили индекс элемента в ROM
        incf FSR,1          ;увеличили индекс элемента в RAM
        decfsz counter,1    ;уменьшили и проверили на ноль
                            ;счетчик цикла
        goto Loop
        movf MasRAM+2,0     ;записали в аккумулятор значение
                            ; элемента массива в RAM
        movwf PORTB
MasROM                      ;Подпрограмма - массив
                            ; {7,3,1,2,0x0f,0xff}
        addwf   PCL,1
        retlw  7
        retlw  3
        retlw  1
        retlw  2
        retlw  0xff
        retlw  0xff

        end
```

Прокомментируем новые элементы программы... Команда `MasRAM equ 0x20` сохраняет в переменной `MasRAM` адрес ОЗУ, начиная с которого будет размещен массив. Это — первый из доступных адресов регистров общего назначения в микроконтроллере PIC16F877A. Переменная `tmp` размещается по адресу `MasRAM+6` (первый свободный адрес после шести элементов массива). Таким образом, `tmp` будет у нас индексом элементов массива. Переменная `counter` размещается в следующем байте с адресом `tmp+1`.

Далее в FSR необходимо записать начальный адрес (`MasRAM`). Как мы знаем, напрямую это сделать нельзя. Приходится идти по пути "в два приема", через аккумулятор. Напоследок обнуляется индекс, и счетчик `counter` инициализируется значением 6.

Основной цикл Loop – goto Loop выполнится шесть раз. Как следует из комментариев в каждой строке, здесь массив копируется из ПЗУ в ОЗУ с использованием уже рассмотренной выше подпрограммы MasROM. После основного цикла массив в RAM скопирован и с ним можно работать. Две последние инструкции в программе записывают число из адреса MasRAM+2 в аккумулятор и копируют его оттуда в порт B. В результате светодиоды высветят 1. Если бы мы изменили MasRAM+2 на MasRAM+4 (или MasRAM+5), то загорелись бы все светодиоды порта B.

Работа с АЦП

При работе с АЦП используются два управляющих регистра специальных функций (ADCON0, ADCON1) и два регистра для хранения результата (ADRESL, ADRESH). Поскольку в PIC16F877A АЦП выдает 10-разрядный код результата, одного регистра для размещения результата преобразования недостаточно. Младшие восемь разрядов результата помещаются в ADRESL, а два оставшихся — в ADRESH. Остальные разряды могут размещаться в двух младших или в двух старших разрядах регистра: так называемое правое или левое выравнивание.

Описание разрядов регистра ADCON1 представлено на рис. 4.9. Описание разрядов регистра ADCON0 представлено на рис. 4.10.

Проект 666 (листинг 4.6) выполняет аналого-цифровое преобразование входного сигнала и отображает двоичный код результата через порты B (младшие разряды) и D (два старших разряда).

Листинг 4.6. Отображение результата аналого-цифрового преобразования

```
LIST  p=16F877A
 #include p16f877a.inc
 __CONFIG 0x03F72
 ORG 0x00
       goto Start
 ORG 0x05
Start:
       COUNT1 EQU 0x0E
       bsf STATUS, RP0          ;переходим в банк 1
       movlw b'10000000'
       movwf ADCON1
       movlw b'00000000'        ;помещаем в аккумулятор число 0
       movwf TRISB      ;устанавливаем линии порта B на вывод
       movwf TRISD      ;устанавливаем линии порта D на вывод
       bcf STATUS, RP0          ;переходив в банк 0
Loop_0:
       movlw b'10010001'
```

Листинг 4.6. Окончание

```
        movwf ADCON0
        MOVLW .255
        MOVWF COUNT1
loop_1:
        DECFSZ COUNT1,1
        GOTO loop_1
        bsf ADCON0, 2
Loop_2:
        btfsc ADCON0, 2
        GOTO Loop_2
        bsf STATUS, RP0       ;переходим в банк 1
        movf ADRESL, 0
        bcf STATUS, RP0       ;переходив в банк 0
        movwf PORTD
        movf ADRESH, 0
        movwf PORTB
        GOTO Loop_0
        END
```

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/-DONE	-	ADON
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

биты 7-6: **ADCS1:ADCS0:** Выбор источника тактового сигнала
00 = Fosc/2
01 = Fosc/8
10 = Fosc/32
11 = FRC (внутренний RC генератор модуля АЦП)

биты 5-3: **CHS2:CHS0:** Выбор аналогового канала
000 = канал 0, (RA0/AN0)
001 = канал 1, (RA1/AN1)
010 = канал 2, (RA2/AN2)
011 = канал 3, (RA3/AN3)
100 = канал 4, (RA5/AN4)
101 = канал 5, (RE0/AN5)(1)
110 = канал 6, (RE1/AN6)(1)
111 = канал 7, (RE2/AN7)(1)

бит 2. **GO/-DONE:** Бит статуса модуля АЦП
Если ADON=1
1 = модуль АЦП выполняет преобразование (установка бита вызывает начало преобразования)
0 = состояние ожидания (аппаратно сбрасывается по завершению преобразования)

бит 1. **Не используется:** читается как '0'

бит 0. **ADON:** Бит включения модуля АЦП
1 = модуль АЦП включен
0 = модуль АЦП выключен и не потребляет тока

Примечание 1. Эти каналы не реализованы в микроконтроллерах PIC16F873/ PIC16F876.

Рис. 4.9. Описание разрядов регистра ADCON1

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
-n – значение после POR
-x – неизвестное значение после POR

Бит 7: **ADFM:** Формат сохранения 10-разрядного результата
1 = правое выравнивание. 6 старших бит ADRESH читаются как '0'
0 = левое выравнивание. 6 младших бит ADRESL читаются как '0'

Биты 6-4: **Не используются:** читаются как '0'

Биты 3-0: **PCFG3:PCFG0:** Управляющие биты настройки каналов АЦП

PCGF3: PCGF0	AN7[1] RE2	AN6[1] RE1	AN5[1] RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF+}	V_{REF-}	Кан./ V_{REF}[2]
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}	8/0
0001	A	A	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	7/1
0010	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}	5/0
0011	D	D	D	A	V_{REF+}	A	A	A	RA3	V_{SS}	4/1
0100	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}	3/0
0101	D	D	D	D	V_{REF+}	D	A	A	RA3	V_{SS}	2/1
011x	D	D	D	D	D	D	D	D	V_{DD}	V_{SS}	0/0
1000	A	A	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}	6/0
1010	D	D	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	5/1
1011	D	D	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	4/2
1100	D	D	D	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	3/2
1101	D	D	D	D	V_{REF+}	V_{REF-}	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}	1/0
1111	D	D	D	D	V_{REF+}	V_{REF-}	D	A	RA3	RA2	1/2

A = аналоговый вход D = цифровой канал ввода/вывода

Примечания:
1. Эти каналы не реализованы в микроконтроллерах PIC16F873/ PIC16F876.
2. В этом столбце указывается число аналоговых каналов, доступных для выполнения преобразования, и число входов источника опорного напряжения.

Рис. 4.10. Описание разрядов регистра ADCON0

Определяется переменная `COUNT1` — счетчик для организации цикла задержки. Переходим в банк 1 и там в самый старший разряд регистра ADCON1 записываем 1, а во все остальные разряды — 0. В соответствии с описанием мы тем самым устанавливаем правое выравнивание для старших разрядов, аналоговый ввод назначаем через порт A, и диапазон изменения входного аналогового напряжения устанавливаем от 0 до напряжения питания.

Далее, в цикле `Loop_0 – goto Loop_0` вначале конфигурируем разряды регистра ADCON1 двоичным числом `10010001`. Обращаемся к описанию регистра и видим, что при этом выбирается частота источника тактового сигнала для АЦП, равная 1/32 от тактовой частоты осциллятора, канал аналогового ввода 2 (AR2), включается режим ожидания и включается сам АЦП.

В цикле `loop_1 – goto loop_1` с помощью счетчика `COUNT1` устроена небольшая задержка, необходимая для зарядки конденсатора на

входе АЦП. Команда `bsf ADCON0, 2` устанавливает в 1 разряд 2 регистра ADCON0. Тем самым отключается режим ожидания, и модуль непосредственно приступает к преобразованию.

Преобразование занимает некоторое время, в течение которого указанный разряд остается в состоянии 1. Как только преобразование завершено, разряд обнуляется. Цикл `Loop_2 – goto Loop_2` работает в ожидании сброса. Здесь задействована команда `btfsc ADCON0, 2`. Она анализирует состояние разряда, и, если его значение равно 1, то ничего не происходит, если же разряд равен 0, то пропускается следующая команда, т.е. мы выходим из цикла.

Теперь в регистре ADRESL (размещен в банке 1) находятся младшие разряды результата. Пересылаем их через аккумулятор в порт D. Два старших разряда результата — в регистре ADRESH (размещен в банке 0). Пересылаем их в порт B. Все это зацикливается, позволяя вращать регулятор потенциометра АЦП на плате EasyPIC5 и наблюдать за изменениями в состоянии светодиодов.

Сравнение С и ассемблера

В завершение нашего путешествия по миру программирования микроконтроллеров сравним эффективность разработки на С и ассемблере. Заодно познакомимся и с работой таймера МК.

Начнем с ассемблера... Пусть наш проект (`777`) мигает светодиодами на порте B под управлением таймера. Рассмотрим работу таймера TMR1. Он оснащен 16-разрядным счетчиком, состоящим из регистров TMR1L (младшие восемь разрядов) и TMR1H (старшие восемь разрядов). На каждом машинном цикле (четыре такта осциллятора) счетчик инкрементируется. При достижении счетчиком числа 0xFF на следующем цикле происходит переполнение таймера: счетчик сбрасывается в 0, и устанавливается в 1 разряд TMR1IF регистра PIR1 (банк 0). Это событие можно рассматривать как прерывание от таймера 1. Режим работы таймера 1 определяется состоянием регистра T1CON, разъяснения по которому показано на рис. 4.11.

Представленной информации достаточно для понимания текста программы (листинг 4.7).

Листинг 4.7. Использование таймера 1

```
LIST  p=16F877A
 #include p16f877a.inc
  __CONFIG b'11111101111110'
ORG 0x00
    goto Start
```

Листинг 4.7. Окончание

```
 ORG 0x05
Start:
    BSF  STATUS,RP0
    MOVLW b'00000000'
    MOVWF TRISB
    BCF  STATUS,RP0
    MOVLW b'11110000'
    MOVWF PORTB
    MOVLW .49
    MOVWF T1CON
    BSF T1CON,TMR1ON
Loop:
    BTFSS PIR1, TMR1IF
    GOTO Loop
    BCF PIR1,TMR1IF
    COMF PORTB,1
    GOTO Loop

 END
```

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	T1CKPS	T1CKPS	T1OSCE	-T1SYNC	TMR1CS	TMR1ON
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

биты 7-6: **Не реализованы**: читаются как '0'

биты 5-4: **T1CKPS1:T1CKPS0**: Выбор коэффициента деления предделителя TMR1
11 = 1:8
10 = 1:4
01 = 1:2
00 = 1:1

бит 3: **T1OSCEN:** Включение тактового генератора TMR1
1 = генератор включен
0 = генератор выключен (инвертирующий элемент и резистивная обратная связь выключены для уменьшения тока потребления)

бит 2: **-T1SYNC:** Синхронизация внешнего тактового сигнала
TMR1CS = 1
1 = не синхронизировать внешний тактовый
0 = синхронизировать внешний тактовый

TMR1CS = 0
Значение бита игнорируется

бит 1: **TMR1CS:** Выбор источника тактового сигнала
1 = внешний источник с вывода T1OSO/T1CKI (активным является передний фронт сигнала)
0 = внутренний источник Fosc/4

бит 0: **TMR1ON:** Включение модуля TMR1
1 = включен
0 = выключен

Рис. 4.11. Разряды регистра T1CON

Здесь (после стандартных операций) в порт B записывается двоичный код 11110000. Десятичное число 49 (двоичное 00110001), записанное в регистр T1CON, задает значение предделителя 1:8. Команда BSF T1CON,TMR1ON устанавливает в 1 разряд TMR1ON, тем самым, запуская таймер.

Далее организован бесконечный цикл Loop – GOTO Loop. В нем проверяется значение разряда TMR1IF. Пока он не равен 1 (нет переполнения), происходит переход на метку Loop. При переполнении разряд TMR1IF обнуляется, а значение порта B инвертируется.

Запустив программу на плате EasyPIC5, мы увидим "перемигивание" светодиодов с интервалом примерно четыре раза в секунду. Посмотрим на размер исполняемой программы (файл 777.hex). Он составляет 159 байт.

Теперь создадим аналогичный проект Timer1_01 в системе MikroC. Соответствующий C-код представлен в листинге 4.8.

Листинг 4.8. Программа на языке C, аналогичная коду из листинга 4.7

```
void interrupt() {
  PIR1.TMR1IF = 0;            // обнуление TMR1IF
  TMR1H = 0x00;
  TMR1L = 0x00;
  PORTB  = ~PORTB;
}

void main() {
  PORTB = 0xF0;               // инициализация PORTB
  TRISB = 0;                  // PORTB на выход
  T1CON = 49;                 // установка режима
  PIE1.TMR1IE  = 1;           // разрешение прерываний от Timer1
  INTCON = 0xC0;              // установка GIE, PEIE
M: goto M;
}
```

Читатель уже стал достаточно опытен для того, чтобы разобраться в программе без наших комментариев. Скомпилировав, "прошив" и запустив программу, мы убедимся, что система делает то же самое, что и ассемблерный проект. Проверим размер HEX-файла. Он составляет 300 байт, что существенно больше, чем в предыдущем варианте. Это наглядно демонстрирует экономичность программирования на ассемблере. А главное, не так уж это оказалось и сложно!

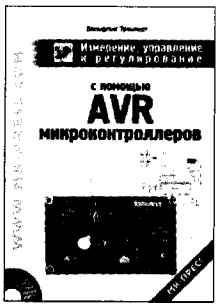
На этой оптимистичной ноте мы хотим поздравить читателя с прохождением трех главных шагов в программировании микроконтроллеров и пожелать им успехов в дальнейшем самостоятельном освоении этого интересного дела!

Содержимое прилагаемого к книге компакт-диска

Прилагаемый к книге компакт-диск содержит следующие папки:

- `Flowcode` — дистрибутив программы Flowcode;
- `MikroC` — установочный файл среды программирования MikroC, а также справка по ней на русском языке;
- `MPLAB` — дистрибутив пакета MPLAB 7.50;
- `PC_PIC_PC` — программа и DLL-файл системы обмена данными между персональным компьютером и микроконтроллером;
- `Projects` — исходные тексты рассмотренных в книге проектов и программ для Flowcode, MikroC и MPLAB.

Издательство "МК-Пресс" представляет



Вольфганг Трамперт

Измерение, управление и регулирование с помощью AVR микроконтроллеров (+CD)

ISBN 966-8806-14-X
208 стр., мягкая обложка

Книга описывает особенности применения AVR-микроконтроллеров в технике измерения, управления и регулирования. При этом основной акцент поставлен на измерении напряжения, выводе и отображении результатов измерений, а также на регулировании аналоговых напряжений. Изложенный материал дает возможность поэтапно проследить весь процесс разработки устройства, понять, почему программное и аппаратное обеспечение скомпоновано именно таким, а не каким-либо другим образом, и суметь в случае необходимости выполнить самостоятельную разработку.

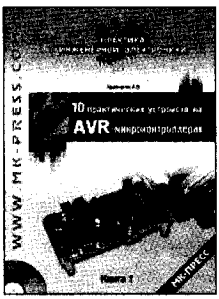
Дитер Кохц

Измерение, управление и регулирование с помощью PIC-микроконтроллеров (+CD)

ISBN 966-8806-15-8
304 стр., мягкая обложка

Книга посвящена применению PIC-микроконтроллеров семейства PIC16C5X, а также PIC16C71 и PIC16F84 в схемах измерения, управления и регулирования. Рассмотрены следующие примеры: программируемый счетчик-частотомер, измерение температуры, реле времени, гигрометр с реле, температурное реле для регулирования нагрева, управление шаговым электродвигателем, регулирование уровня заполнения.

Издательство "МК-Пресс" представляет



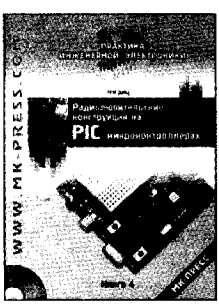
Кравченко А.В.

10 практических устройств на AVR-микроконтроллерах. Книга 1 (+CD)

ISBN 978-966-8806-41-4
224 стр., мягкая обложка

Данная книга открывает серию сборников с практическими примерами применения микроконтроллеров. В ней рассмотрены десять завершенных устройств на базе микрокон-троллеров AVR, которые можно легко собрать в домашних условиях и применять в быту или профессиональной деятельности: генератор световых эффектов; счетчик событий; музыкальный звонок; индикатор уровня звука; повышающий преобразователь, схема управления шаговым двигателем; цифровой термометр и др.

Благодаря подробному анализу аппаратной и программной части устройств, книга будет интересна и полезна как начинающим, так и опытным радиолюбителям, желающим изучить методы эффективного применения микроконтроллеров.



Заец Н.И.

Радиолюбительские конструкции на PIC-микроконтроллерах (+CD). Книга 4

ISBN 966-8806-42-1
336 стр., мягкая обложка

Данная книга — практическое пособие по освоению микроконтроллеров PIC компании Microchip и другой современной элементной базы, наподобие индикаторов, выполненных по COG-технологии. Рассмотрены алгоритмы работы, схемы и программы для различных полезных устройств: многофункциональных часов, отображающих текущее время и температуру воздуха; автомобильных часов, фиксирующих время в пути и сообщающих о поломке реле-регулятора; автомата включения освещения; цифрового устройства для блока питания с установкой защиты по току и напряжению; специализированных термометров и др. Для начинающих дана глава о наладке устройств на микроконтроллерах. Книга предназначена для широкого круга радиолюбителей, а также будет полезна студентам, изучающим микроконтроллеры.

Книги издательства "МК-Пресс" можно заказать
по e-mail: **info@mk-press.com**

Посетите наш Internet-магазин: **http://www.mk-press.com**

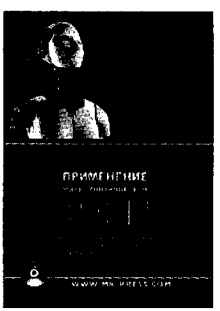
Издательство "МК-Пресс" представляет

Тим Уилмсхерст

Разработка встроенных систем с помощью микроконтроллеров PIC. Принципы и практические примеры (+CD)

ISBN 978-5-903383-61-0
544 стр., мягкая обложка

Благодаря полезным примерам и иллюстрациям, эта книга дает глубокие познания в сфере проектирования систем с помощью микроконтроллеров PIC, а также — программирования этих устройств на ассемблере и С. Подробно рассмотрены микроконтроллеры 16F84A, 16F873A и 18F242. Даны примеры реальных проектов, включая модель робота, выполненного в виде транспортного средства с автономным управлением. Дополнительно рассматриваются такие вопросы повышенной сложности, как применение устройств в сетевой среде и построение операционных систем реального времени.



Барри Брэй

Применение микроконтроллеров PIC18. Архитектура, программирование и построение интерфейсов с применением С и ассемблера (+CD)

ISBN 978-5-7931-0516-3
576 стр., мягкая обложка

Сегодня микроконтроллеры используются повсеместно в автомобилях, бытовой технике, промышленном и медицинском оборудовании и т.п. Этот учебник дает всестороннее представление об архитектуре, программировании и построении интерфейсов этого современного чуда. На примере семейства микроконтроллеров PIC18 производства Microchip в книге объясняется архитектура, программирование и построение интерфейсов. Семейство PIC18 выбрано не случайно, поскольку оно относится к самым современным восьмиразрядным микроконтроллерам. Изложенный в книге материал также применим как к более ранним версиям микроконтроллеров Microchip, так и к аналогичным устройствам других производителей. Он рассчитан на опытных практиков и радиолюбителей, интересующихся микроконтроллерами.

Книги издательства "МК-Пресс" можно заказать
по e-mail: **info@mk-press.com**
Посетите наш Internet-магазин: **http://www.mk-press.com**

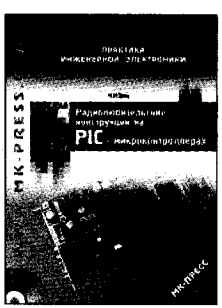
Издательство "МК-Пресс" представляет

Лусио ди Джасио

Программирование на С микроконтроллеров PIC24 (+CD)

ISBN 978-5-7931-0529-3
336 стр., мягкая обложка

Лусио ди Джасио, эксперт из компании Microchip, предлагает свой уникальный взгляд на революционную технологию PIC24, проводя читателя от основ 16-разрядной архитектуры до сложных программных разработок средствами языка С, включая реализацию многозадачности с помощью прерываний PIC24, управление ЖК-дисплеями, формирование звуковых и видеосигналов, доступ к запоминающим устройствам большой емкости и др. Вне всякого сомнения, эта книга будет полезна как опытным PIC-разработчикам, так и новичкам в мире встроенных систем.



Заец Н.И.

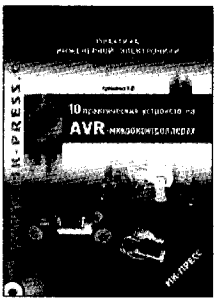
Радиолюбительские конструкции на PIC-микроконтроллерах (+CD). Книга 1

ISBN 978-5-7931-0518-7
304 стр., мягкая обложка

Это издание переработано с целью исправления замеченных ошибок и уменьшения объема за счет переноса статей, не пользующихся спросом, на прилагаемый компакт-диск. В книге представлено 20 описаний радиолюбительских устройств различного назначения, выполненные на микроконтроллере PIC16F84. Начинающие радиолюбители, не знакомые с программированием микроконтроллеров, смогут без труда воспроизвести любое устройство. Радиолюбители, имеющие опыт программирования, могут изменить программы и печатные платы под свои цели. Для этого в книге даны алгоритмы и исходные тексты программ с подробными комментариями. Автор также делится опытом работы с ассемблером MPLAB и различными средствами программирования. Рассматриваются типичные ошибки при наладке устройств на микроконтроллерах. Книга предназначена для широкого круга радиолюбителей, а также может быть полезна студентам, изучающим программирование микроконтроллеров.

Книги издательства "МК-Пресс" можно заказать
по e-mail: **info@mk-press.com**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Издательство "МК-Пресс" представляет



Кравченко А.В.

10 практических устройств на AVR-микроконтроллерах. Книга 2 (+CD)

ISBN 978-5-7931-0532-3
320 стр., мягкая обложка

Это вторая книга из серии сборников с практическими примерами применения микроконтроллеров AVR. В ней подробно рассмотрены десять завершенных устройств на базе микроконтроллеров AVR, которые можно легко собрать в домашних условиях и применять в быту или профессиональной деятельности: генератор световых эффектов с помощью АЦП; схема управления шаговым двигателем; автомат освещения шкафа-купе; схема управления вентилятором; простой термометр; световой эффект "Призма"; микроконтроллерный генератор; робот, который двигается на свет и умеет обходить преграды; система радиоуправления моделью автомобиля; схема цифрового управления паяльником. Каждому устройству посвящена отдельная глава, где подробно описаны все этапы создания микроконтроллерной модели и программ, начиная со структуры и блок-схемы, и заканчивая самой программой и рабочим кодом.

Оуэн Бишоп

Настольная книга разработчика роботов (+CD)

ISBN 978-5-7931-0559-0
176 стр., мягкая обложка

Эта книга представляет собой справочное руководство для тех, кто хочет научиться проектировать и конструировать роботов. Благодаря представленным в ней пошаговым инструкциям, вы быстро освоите методики создания забавных и захватывающих роботов. На основании своего обширного практического опыта автор открывает важные аспекты программирования, электроники и механики, характерные для робототехники. Поскольку все проекты основаны на использовании всемирно популярных микроконтроллеров PIC, методики программирования осваиваются быстро и безболезненно.

Книги издательства "МК-Пресс" можно заказать
по e-mail: **info@mk-press.com**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Издательство "МК-Пресс" представляет

Крид Хадлстон

Проектирование интеллектуальных датчиков с помощью Microchip dsPIC (+CD)

ISBN 978-966-8806-38-4
320 стр., мягкая обложка

На страницах этой книги раскрыты способы применения популярных цифровых контроллеров сигналов Microchip dsPIC, в которых вычислительный потенциал мощных цифровых процессоров сигналов удачно объединен с возможностями микроконтроллеров PIC. Рассматриваются вопросы не только программирования, но и проектирования электронного оборудования. Таким образом, читатель получает полное представление о процессе создания интерфейса для трех конкретных типов датчиков: температуры, давления/нагрузки и расхода. Книга раскрывает реальные проблемы, возникающие в повседневной работе разработчиков, и показывает решения, позволяющие реализовать все сильные стороны интеллектуальных датчиков.

Авраменко Ю.Ф.

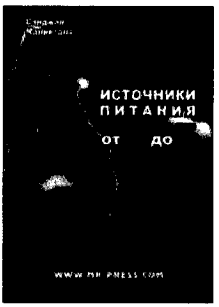
Качественный звук — сегодня это просто

ISBN 966-8806-27-1
286 стр., мягкая обложка

В книге максимально подробно приведены все рекомендации разработчиков – инженеров NSC, как правильно построить усилительный тракт на основе мощных ОУ. Современный подход, основанный на рекомендациях инженеров AD и TI, к топологии печатной платы, к выбору «правильных» пассивных компонентов для звуковоспроизводящего тракта поможет реализовать основной принцип: как можно меньше ухудшить качество записи. Большое количество примеров построения качественных УМЗЧ будет наглядным пособием для реализации собственной конструкции в короткие сроки с небольшими материальными затратами и главное, с предсказуемым результатом.

Книги издательства "МК-Пресс" можно заказать
по e-mail: **info@mk-press.co**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Издательство "МК-Пресс" представляет

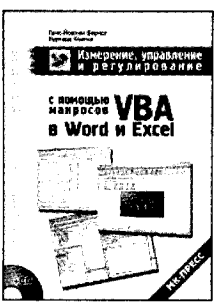


Санджая Маниктала

Импульсные источники питания от A до Z (+CD)

ISBN 978-5-903383-59-7
544 стр., мягкая обложка

Эта книга основывается на десятилетнем авторском опыте проектирования источников питания. Здесь читатель найдет наглядное и доступное введение в курс "Источники питания"; изложение основ без пугающего математического анализа; полную и, в то же время, уникальную по своей простоте методику проектирования импульсных преобразователей и их магнитных компонентов; подробный расчет всех видов потерь в импульсных источниках питания; описание основных схемотехнических решений импульсных источников; исчерпывающее исследование аспектов контроля и измерения паразитных электромагнитных излучений, связанных с работой импульсных преобразователей.



Г.-Й. Берндт, Б. Каинка

Измерение, управление и регулирование с помощью макросов VBA в Word и Excel (+CD)

ISBN 978-5-7931-0504-0
256 стр., мягкая обложка

Эта книга представляет новый подход, согласно которому весь диапазон задач измерения, управления и регулирования реализуется средствами популярного программного пакета Microsoft Office. Хотя это звучит необычно, с помощью приложений Word и Excel можно получить прямой доступ к аппаратному обеспечению, что делает их универсальными и простыми в использовании инструментами.

В книге показано, как с помощью макросов VBA реализовать управление цифровыми мультиметрами, релейными картами и ПК-интерфейсами, организовать взаимодействие с микроконтроллерными системами и многое другое на основе стандартного последовательного интерфейса RS232.

Книги издательства "МК-Пресс" можно заказать
по e-mail: **info@mk-press.com**
Посетите наш Internet-магазин: **http://www.mk-press.com**

ББК 32.973-04
УДК 004.312
І20

Іванов В. Б.

І20 Програмування мікроконтролерів для початківців. Візуальне проектування, мова С, асемблер. – К.: "МК-Пресс", СПб.: "КОРОНА-ВЕК", 2016. – 176с., іл.

ISBN 978-5-7931-0559-0 ("КОРОНА-ВЕК")
ISBN 978-966-8806-65-0 ("МК-Пресс")

У цій книзі автор проводить читача шляхом опанування програмуванням мікроконтролерів від простого до складного. Почавши з короткого опису архітектури та системи команд мікроконтролерів PIC, він переходить до візуального проектування у середовищі Flowcode, яке дозволяє отримати робочий код без будь-яких навичок програмування на асемблері чи мові високого рівня. В останніх двох розділах дані приклади програмування мікроконтролерів PIC за допомогою мови С у середовищі MikroC та асемблера у середовищі MPLAB.

Для освоєння матеріалу книги не треба ніяких спеціальних знань, що спрощує процес вивчення програмування мікроконтролерів для тих, хто починає працювати у цьому напрямку "з нуля".

ББК 32.973-04

Головний редактор: Ю. О. Шпак

ISBN 978-5-7931-0559-0 ("КОРОНА-ВЕК")
ISBN 978-966-8806-65-0 ("МК-Пресс")