Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет радиотехники и электроники

Кафедра информационных радиотехнологий

С. Ю. Васюкевич, И. Г. Давыдов, А. В. Цурко

ВЫЧИСЛИТЕЛЬНЫЕ СРЕДСТВА РАДИОСИСТЕМ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальности 1-39 01 02 «Радиоэлектронные средства»

Минск БГУИР 2016

УДК 004.4:621.396.6(076.5)
ББК 32.973.26+32.844я73
В20

Р е ц е н з е н т ы:

кафедра автоматизированных систем управления производством учреждения образования «Белорусский государственный аграрный технический университет» (протокол №11 от 02.04.2015);

доцент кафедры информационно-измерительной техники и технологий Белорусского национального технического университета, кандидат физико-математических наук, доцент А. А. Антошин

Васюкевич, С. Ю.

В20 Вычислительные средства радиосистем. Лабораторный практикум : учеб.-метод. пособие / С. Ю. Васюкевич, И. Г. Давыдов, А. В. Цурко. – Минск : БГУИР, 2016. – 128 с. : ил.

ISBN 978-985-543-183-2.

Предназначено для студентов, изучающих проектирование систем на основе микроконтроллеров. Содержит описание лабораторного оборудования, руководство по применению программного и аппаратного инструментария, необходимые теоретические сведения, методику выполнения лабораторных работ, варианты индивидуальных заданий, контрольные вопросы и библиографические ссылки на рекомендуемую для самостоятельного изучения литературу.

УДК 004.4:621.396.6(076.5)
ББК 32.973.26+32.844я73

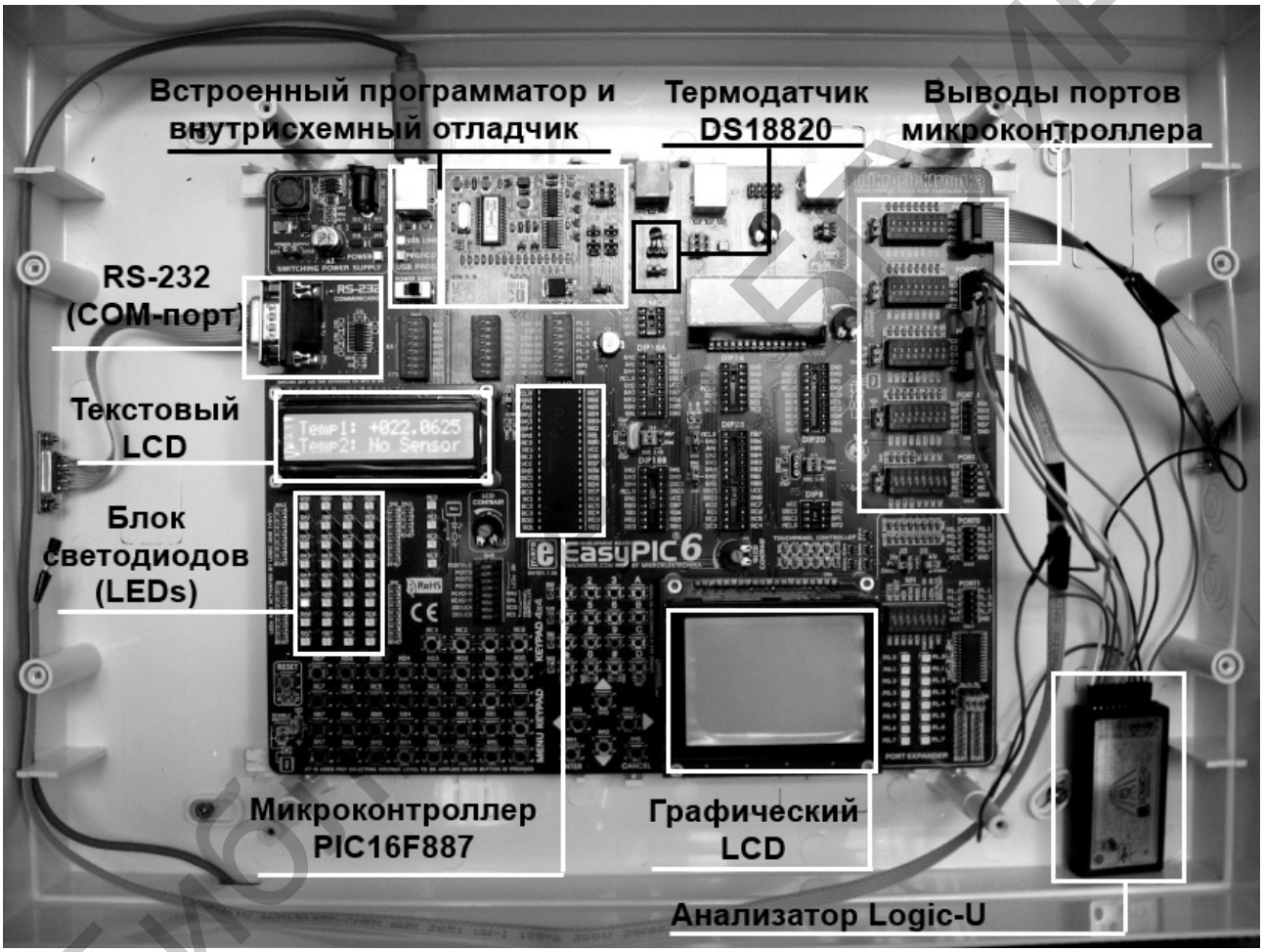
ISBN 978-985-543-183-2

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящее учебно-методическое пособие состоит из четырех лабораторных работ, в процессе выполнения которых студенты знакомятся с основами проектирования цифровых систем, возможностями и архитектурой микроконтроллеров, областью их применения в радиотехнике, приобретают навыки работы с типовыми инструментами.

Лабораторные макеты для проведения работ выполнены на базе платформы EasyPic6 производства MikroElektronika. Вид лабораторного макета изнутри представлен ниже.



Внешний вид лабораторного макета

В качестве изучаемых аппаратных средств выступает восьмибитный микроконтроллер PIC16F887, монохромные матричные жидкокристаллические дисплеи, цифровой термометр DS18B20, интерфейсы RS-232, 1-Wire. Дополнительными инструментами выступают логический анализатор Logic-U и встроенный в отладочную плату внутрисхемный отладчик ICD. В качестве программных средств используется среда разработки и отладки «MicroC PRO for PIC» v5.xx (версия 5.01 и выше), приложения для прошивки «microProg Suite for PIC», программный интерфейс логического анализатора «Saleae LLC».

В качестве дополнительных инструментов выступают шестнадцатеричный редактор «CI Hex Viewer», программа UART терминала «Terminal». Программные средства используются при помощи персонального компьютера под управлением ОС Windows XP или последующих версий.

В состав учебно-методического пособия входят четыре лабораторные работы.

Первая работа является ознакомительной. Рассматриваются принципы использования лабораторного макета на базе платформы EasyPIC6 для проектирования встраиваемых вычислительных систем (структура и принципиальная схема макета, архитектура микроконтроллера PIC16F887, методика использования лабораторного оборудования и выполнения лабораторных работ), программные инструменты (интерфейс и возможности программ), основы применения языка MicroC для разработки программного обеспечения под микроконтроллеры. Лабораторным заданием является чтение и запись данных на портах микроконтроллера с выполнением арифметических операций над данными.

Вторая работа посвящена особенностям подключения и вывода информации на монохромные матричные жидкокристаллические индикаторы на базе контроллеров KS0066 и KS0107, применению знакогенераторов, использованию библиотек и функций языка MicroC. Лабораторное задание заключается в настройке модулей текстового и графического дисплеев и отображении заданной информации (текст, результаты вычислений, графики).

Третья работа знакомит студентов с организацией системы прерываний в микроконтроллере, типовым интерфейсом RS-232, аппаратной и программной реализацией асинхронной передачи данных на основе модуля UART микроконтроллера PIC16F887. Лабораторное задание основано на приеме микроконтроллером текстовых команд и данных по UART, исполнении команд, формировании и передаче ответных сообщений.

Четвертая работа является обобщающей. Студенты изучают принцип работы цифрового термометра DS18B20 и полудуплексную передачу данных по интерфейсу 1-Wire. Лабораторное задание требует комбинирования полученных знаний и навыков для разработки системы мониторинга температуры с выводом значений на ЖК-дисплей и управлением по UART.

Таким образом, для каждой лабораторной работы приводятся основные теоретические сведения, дополненные ссылками на список литературы в конце данного учебно-методического пособия, приводятся методика выполнения и содержание отчета, контрольные вопросы, варианты заданий, пример решения типового задания.

Лабораторная работа №1

ОЗНАКОМЛЕНИЕ С ЛАБОРАТОРНЫМ МАКЕТОМ И ВВОДОМ-ВЫВОДОМ ДАННЫХ ЧЕРЕЗ ПОРТЫ МИКРОКОНТРОЛЛЕРА

1.1. Цели работы

1. Изучение основ программирования микроконтроллеров PIC на языке mikroC.
2. Получение навыков работы с основными аппаратными и программными инструментами.
3. Получение базовых навыков ввода-вывода данных через порты микроконтроллера.

1.2. Теоретические сведения

1.2.1. Понятие микроконтроллера

Современное положение дел в области проектирования и программирования устройств на микроконтроллерах является следствием развития технологий в области интегральных схем. Они позволяли размещать сотни тысяч транзисторов на одном чипе, что стало главной предпосылкой к появлению микропроцессоров. Первые компьютеры строились по принципу добавления к ним разного рода периферийных элементов, таких как память, устройства ввода-вывода, таймеры и прочие устройства. Дальнейшее увеличение плотности расположения элементов на схеме привело к появлению интегральных схем, содержащих и процессор, и периферию. Так был создан первый чип, содержащий микрокомпьютер, позже эта конструкция получила название «микроконтроллер» (сокращенно МК или MPU). Подробнее см. [1].

Новички обычно считают, что понятия «микроконтроллер» и «микропроцессор» абсолютно идентичны, но это не так. Отличия между ними довольно существенны. Первое и главное отличие в пользу микроконтроллера – его функциональность. Для того чтобы микропроцессор заработал, нужно обязательно добавить к нему другие компоненты (как минимум память). Даже от самой мощной вычислительной машины не будет никакой пользы, если не подключить к ней соответствующие периферийные устройства. В свою очередь, периферийные устройства подключаются к микропроцессору с помощью специальных схем. Такие принципы использовались как во времена зарождения микропроцессорной техники, так и в наши дни.

С другой стороны, микроконтроллер предназначен для того, чтобы сочетать в себе все вышеописанное. По сути ему не нужны никакие внешние компоненты, так как в него уже заложено все необходимое (вся периферия) для работы (рис. 1.1). Это помогает сэкономить время и пространство, которое необходимо для проектирования устройств.

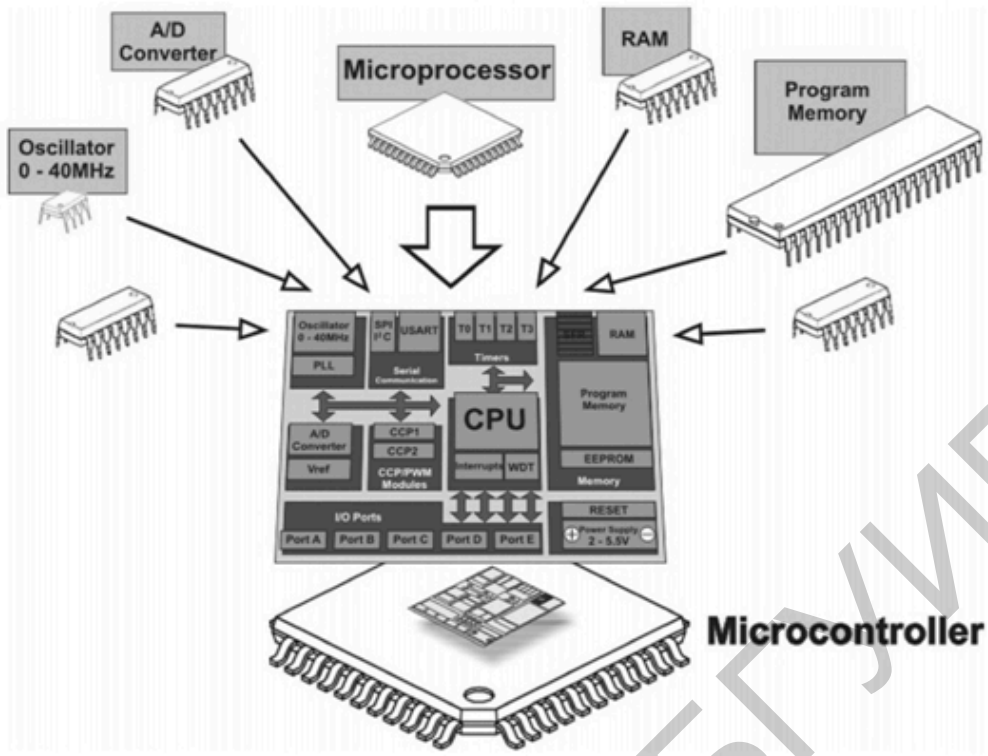


Рис. 1.1. Типичные функциональные блоки микроконтроллера

Возможности микроконтроллеров весьма широки. Для того чтобы вы поняли причины такой популярности микроконтроллеров, мы предлагаем ознакомиться с примером.

Около 10 лет назад проектирование электронного устройства, которое бы позволяло управлять работой лифтов в многоэтажном здании, было непростой задачей даже для команды экспертов. Кстати, вы когда-нибудь задумывались над тем, какие требования предъявляются к организации работы лифта? Что делать, если лифт вызывают в одно и то же время несколько человек на разных этажах? На вызов с какого этажа лифт должен приехать в первую очередь? Как решить вопросы безопасности пассажиров и оборудования? Что делать в случае отключения электричества? А что делать, если лифт начнет падать или же, наоборот, застрянет? Так вот, представьте, что после решения этих и многих других вопросов вам придется тщательно разрабатывать сложную электрическую схему с большим числом специальных элементов. В зависимости от сложности схемы и вашего опыта в данной области этот процесс может занять недели или месяцы. По завершении работы над схемой нужно разработать печатные платы и собрать устройство. Это, в свою очередь, также весьма трудоемкий процесс. Когда устройство спроектировано, собрано, а его работоспособность проверена в ходе многократных тестов и моделирования, можно испытать его на практике.

Как правило, тут наступает момент, когда «развлечение» с задачами проектирования и конструирования заканчивается и начинается другая серьезная работа, так как большинство подобных устройств никогда не

начинают работать с первого раза. Самое время приготовиться к бесконечным калибровкам, корректировкам и, разумеется, внесению улучшений в, как вам кажется, нормально работающую конструкцию одного обычного лифта.

Когда устройство наконец корректно заработает, заказчики и разработчики будут удовлетворены, а работа – оплачена, разного рода компании (например строительные) захотят заполучить разработанные чертежи и схемы, чтобы лифты и в других зданиях также работали правильно. Если повезет, разработчику могут предложить контракт на изготовление нескольких новых копий устройства. Но вот проблема: в новом здании на 4 этажа больше, чем в том, для которого разработали лифт. В таком случае непременно возникнут попытки создать универсальное устройство, которое бы идеально подходило хоть к 4-этажному дому, хоть к 40-этажному, иными словами, создать очередной «шедевр электротехники». Возможно, это даже удастся. Но что делать, если заказчик захочет что-либо изменитиь: установить камеру в лифте, добавить музыкальное сопровождение (и это, может, далеко не весь список улучшений), а управлять этим (собственно как и движением лифта) должно все то же устройство?

В любом случае разработчик не в состоянии бесконечно улучшать конструкцию своего универсального устройства, рано или поздно возможности элементной базы не позволят достичь желаемого результата. Однако вышеописанная задача по организации работы лифта получила намного более простое решение с появлением первых микроконтроллеров. С того момента, как эти компактные, мощные и дешевые устройства были изготовлены, техника начала развиваться по другому пути.

Сейчас электроника позволяет управлять, например, небольшими субмаринами, строительными кранами или вышеупомянутыми лифтами, и все это благодаря одному-единственному чипу. Микроконтроллер предоставляет широкий диапазон возможностей, и инженер сам решает, что с ними делать. Достаточно лишь сбросить существующую программу и записать свою.

Перед включением устройства его работоспособность должна быть проверена на симуляторе. Если все в порядке – можно устанавливать контроллер в готовое устройство, если нет – достаточно просто изменить программу, и не нужно постоянно изобретать и собирать какие-либо новые и сложные схемы и устройства под постоянно изменяющиеся условия.

***Обобщенный алгоритм программирования микроконтроллеров*:**

1. *Исследование* целевых функций и условий работы будущего устройства.
2. *Выбор* микроконтроллера, отвечающего предьявленным требованиям и проверка наличия всех требуемых элементов внутри микроконтроллера (число выводов, таймеров, АЦП, ШИМ и пр.).
3. *Изучение* и при возможности предварительное проектирование и изготовление оборудования, управляемого микроконтроллером и используемого для связи с периферийными устройствами.

4. *Разработка* программы для исполнения на микроконтроллере при помощи ПК и высокоуровневого языка программирования.

5. В процессе разработки активно используется моделирование, обычно в виде программ для *симуляции* работы вашей программы в реальном окружении.

6. Преобразование программы в машинный код, понятный микроконтроллеру. Эта процедура обычно называется сборкой проекта или *компиляцией*.

7. Запись машинного кода в постоянную память микроконтроллера. Вследствие определенного способа записи данных в «ранние» электромагнитные устройства памяти этот шаг называется *прошивкой*.

8. *Установка* работоспособного микроконтроллера с программой в целевом устройстве.

9. Функциональное *тестирование* и при необходимости *настройка* и *отладка* готовой продукции.

1.2.2. Основы двоичной системы счисления и булевой алгебры

Десятичная система счисления

Современные люди привыкли везде использовать 10 цифр. Например, значение числа 764 – 4 единицы, 6 десятков и 7 сотен. Это же число может быть представлено в более сложной форме: $4 + 60 + 700 = 764$. В еще более сложной форме: $4 \cdot 1 + 6 \cdot 10 + 7 \cdot 100 = 764$. Или в «более научном» виде: $4 \cdot 10^0 + 6 \cdot 10^1 + 7 \cdot 10^2 = 764$. Это означает, что форма представления числа не меняет суть, но подбирается для более простого решения задачи. Человек использует десять различных цифр (0 – 9) потому, что это верхняя граница удобного для запоминания разнообразия, а для увеличения диапазона используются позиции цифр в числе ($10^0, 10^1, 10^2$). Другими словами, мы используем десятичную систему счисления (рис. 1.2).

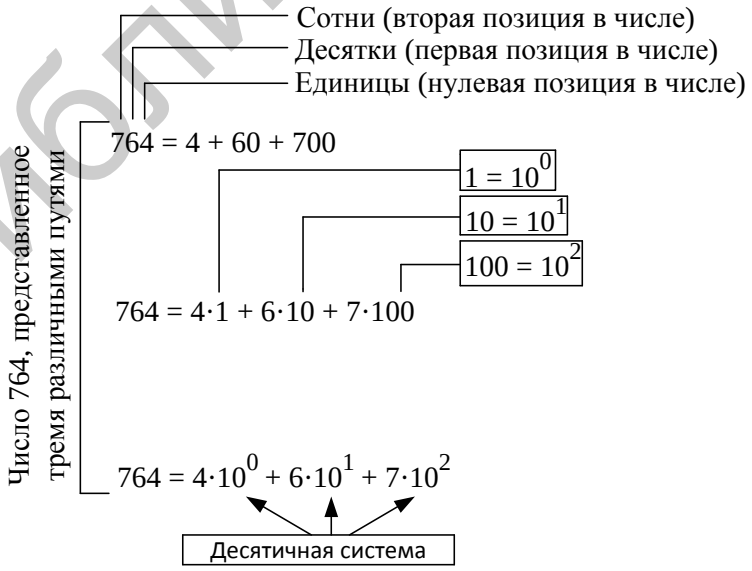


Рис. 1.2. Представление чисел в десятичной системе счисления

Двоичная система счисления

Двоичная система счисления (рис. 1.3) содержит всего две цифры: 0 и 1. При этом привычные математические операции и сравнение множеств объектов при наличии только одной характеристики – что-то есть («1»), либо ничего нет («0») – принципиально не отличаются от привычной десятичной системы счисления.

Рис. 1.3. Представление чисел в двоичной системе счисления

Очевидно, одно и то же число может быть представлено в двух (или более) системах счисления. Единственное отличие между этими системами заключается в количестве цифр, используемых для представления чисел. Одна цифра «2» используется для записи числа «два» в десятичной системе счисления, в то время как в двоичной системе для записи «двойки» потребуется две цифры «1» и «0».

За исключением строго контролируемых лабораторных условий даже самые сложные электронные приборы не могут определить разницу между двумя величинами (между двумя значениями напряжения, например), если они слишком малы (не превышают нескольких вольт). Причиной тому являются электрические шумы в реальных рабочих условиях (непредсказуемые изменения напряжения питания, изменения температуры, допустимое отклонение от номинальных значений установленных компонентов схемы и т. д.). А теперь представьте себе компьютер, который оперирует десятичными числами, представляя их следующим образом: 0 = 0В, 1 = 5В, 2 = 10В, 3 = 15В, 4 = 20В и т. д.

Гораздо проще решить вышеупомянутую проблему с помощью бинарной (двоичной) логики, где «1» это наличие напряжения, а «0» – его отсутствие. Согласитесь, гораздо проще написать нули и единицы, чем фразы «напряжение есть» или «напряжения нет». Речь идет о логических нулях «0» и единицах «1», которые электроника легко распознает и также легко выполняет все математические операции с ними. Мы имеем в виду цифровую электронику, использующую математику, в которой все числа представлены двумя цифрами, а значит, нам важны лишь два состояния: наличия напряжения и отсутствия такового.

Шестнадцатеричная система счисления

Еще на начальном этапе развития компьютерной техники стало ясно, что у пользователей могут возникнуть проблемы с обработкой двоичных чисел в уме. Поэтому была создана новая система счисления, которая использовала 16 различных символов – шестнадцатеричная система счисления. Она использовала не только десять цифр (от 0 до 9), но и шесть букв латинского алфавита (A, B, C, D, E и F). Использование подобных комбинаций может показаться странным, однако именно эта система идеально подходит для компактного представления двоичной системы (рис. 1.4).

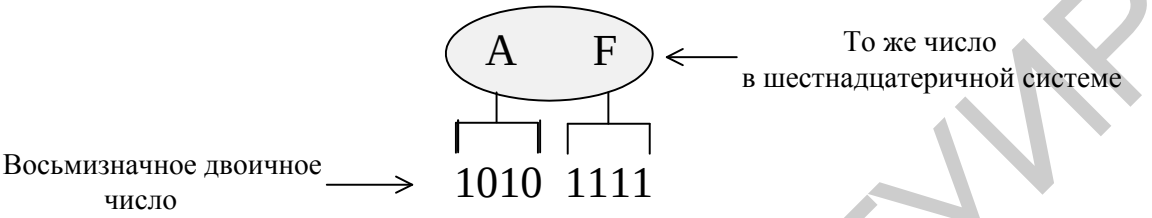


Рис. 1.4. Соответствие чисел шестнадцатеричной и двоичной систем

В двоичной системе самое большое число, которое мы можем записать в четыре разряда, это 1111. В десятичной системе счисления это число 15, а в шестнадцатеричной – это всего одна цифра F. В свою очередь F – это самое большое однозначное число в данной системе. Самое большое 8-разрядное двоичное число (а именно такими числами обычно оперируют компьютеры) – 2-разрядное число в шестнадцатеричной системе.

Двоично-десятичный код

Двоично-десятичный код (от англ. binary-coded decimal, или BCD) – это двоичный код, который используется для представления десятичных чисел. Он используется для обеспечения связи электронных схем либо для связи с периферийными устройствами, использующими десятичную систему внутри и двоичную снаружи. Он состоит из 4-значных двоичных чисел, которые представляют собой десять цифр от 0 до 9 (хотя с четырьмя разрядами можно получить и 16 цифр, но использоваться все равно будет именно 10).

Перевод чисел в другие системы счисления

Цифры в двоичном числе имеют разное значение в зависимости от того, в каком разряде они находятся. Каждая позиция может содержать либо 1, либо 0, и ее значение может быть определено путем подсчета порядкового номера позиции, считая справа от запятой. Чтобы перевести число из двоичной системы в десятичную, необходимо перемножить цифры со значениями, соответствующими их позиции/порядку, и суммировать все получившиеся произведения. Рассмотрим пример (рис. 1.5).

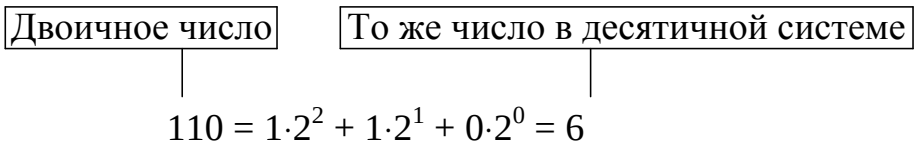


Рис. 1.5. Перевод чисел из двоичной в десятичную систему счисления

Заметим, что для представления десятичных цифр от 0 до 3 нужно использовать только два разряда, а для больших чисел соответственно нужно большее число разрядов. Вы и сами можете подсчитать, сколько чисел можно представить с помощью отведенного числа разрядов. Для этого нужно просто возвести в степень число 2. Двойка будет означать количество цифр, которое использует система счисления (обычно обозначается буквой q), а значение степени (n) – количество разрядов. К тому же если из получившегося числа вычесть единицу, то вы получите максимальное число, которое можно записать в указанное число разрядов. Например:

$2^4 - 1 = 16 - 1 = 15.$

Это значит, что используя четыре цифры (в двоичной системе), вы можете записать 16 чисел в десятичной системе (от 0 до 15).

Перевод чисел из шестнадцатеричной системы счисления в десятичную

Используя приведенные ранее примеры, можно по аналогии (при q = 16) переводить числа из шестнадцатеричной системы счисления в десятичную (рис. 1.6).

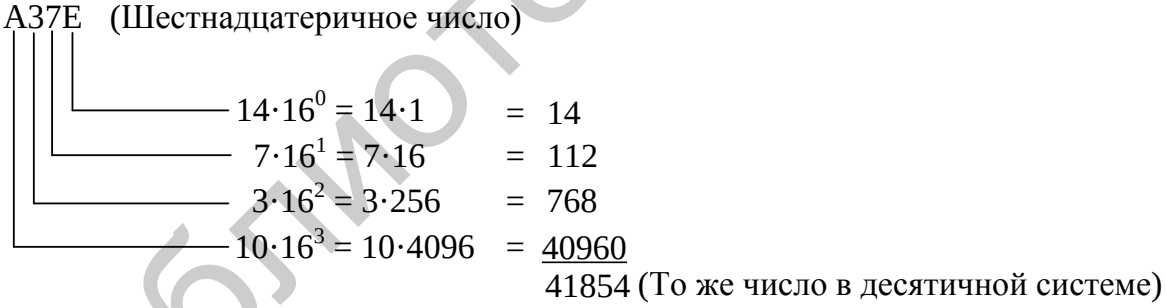


Рис. 1.6. Перевод чисел из шестнадцатеричной в десятичную систему счисления

Перевод чисел из шестнадцатеричной системы счисления в двоичную

Нет необходимости каким-либо образом изменять методику вычисления, для того чтобы перевести число из шестнадцатеричной системы счисления в двоичную. Цифры из шестнадцатеричной системы просто заменяются соответствующими цифрами двоичной системы счисления. Так как самая большая цифра шестнадцатеричной системы счисления эквивалентна числу 15 в десятичной системе, для перевода нам нужно взять 4-разрядное двоичное число (рис. 1.7).

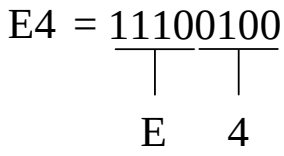


Рис. 1.7. Перевод чисел из шестнадцатеричной в двоичную систему счисления

В табл. 1.1 приведены значения чисел от 0 до 255 в трех системах счисления.

Таблица 1.1

Соответствие чисел в десятичной, двоичной и шестнадцатеричной системах счисления

Десятичная (Decimal, Dec.)	**Двоичная (Binary, Bin.)**	**Шестнадцатеричная (Hexadecimal, Hex.)**
0	0000 0000	0
1	0000 0001	1
2	0000 0010	2
3	0000 0011	3
4	0000 0100	4
5	0000 0101	5
6	0000 0110	6
7	0000 0111	7
8	0000 1000	8
9	0000 1001	9
10	0000 1010	A
11	0000 1011	B
12	0000 1100	C
13	0000 1101	D
14	0000 1110	E
15	0000 1111	F
16	0001 0000	10
17	0001 0001	11
18	0001 0010	12
19	0001 0011	13
20	0001 0100	14
21	0001 0101	15
22	0001 0110	16
23	0001 0111	17
24	0001 1000	18
25	0001 1001	19
26	0001 1010	1A
…	…	…
32	0010 0000	20
64	0100 0000	40
128	1000 0000	80
253	1111 1101	FD
254	1111 1110	FE
255	1111 1111	FF

Маркировка чисел

Шестнадцатеричная система счисления наряду с десятичной и двоичной системами считается наиболее важной для нас. Не составит труда перевести числа из шестнадцатеричной в двоичную систему. Например, что означает фраза «Необходимо отсчитать 110 товаров на сборочной линии»? В зависимости от системы счисления мы можем отсчитать 6, 110 или 272 предмета. Для того чтобы исключить недопонимание в подобных ситуациях, к числам добавляются соответствующие префиксы и суффиксы. Префикс $ или 0x, так же как и суффикс h, добавляется к шестнадцатеричным числам. Например, число 10AF будет выглядеть как $10AF, 0x10AF или 10AFh. Аналогичным образом помечаются двоичные числа: префиксом % или 0b. Если число не отмечено ни суффиксом, ни префиксом, то оно является десятичным. Однако подобный способ маркировки чисел не стандартизирован и зависит от конкретных приложений.

Бит

Согласно определению, бит – базовая единица измерения информации. Однако значение такого определения не вполне понятно, так как нет единственного и однозначного определения информации. Бит – это в первую очередь двоичная цифра. Подобно тому как в десятичной системе нам прежде всего важно, в каком разряде находится цифра, значение бита зависит от его позиции в двоичном числе. Поэтому нет смысла говорить о единицах, десятках и т. п. в двоичной системе, так как цифры указывают сначала на нулевой бит (первый, считая справа), затем на первый бит (второй справа) и т. д. А поскольку в двоичной системе используются только две цифры, то и значение бита может быть либо 0, либо 1.

Иногда можно встретить бит, имеющий значение 4, 16 или 64. Это значит, что он представлен в десятичной системе счисления. Просто мы настолько привыкли к использованию десятичных чисел, что подобное представление стало для нас общим. Корректнее было бы сказать, например: «Значение шестого бита любого двоичного числа эквивалентно десятичному числу 64».

Байт

Байт состоит из восьми сгруппированных битов. Если мы считаем, что бит – это цифра, то логично было бы предположить, что байт – это число. Над ним можно выполнять любые математические операции, как и над любым десятичным числом. Подобно цифрам в любом числе цифры в байте тоже имеют разные значения. Самое большое значение имеет крайний левый бит – старший бит (от англ. most significant bit, или MSB). Соответственно крайний правый бит называется младшим (от англ. least significant bit, или LSB). Так как восемь единиц и нулей могут образовать 256 комбинаций, самое большое десятичное число, которое мы можем записать в один байт, это 255 (одна комбинация представляет собой набор нулей).

Если условно разделить байт на две части, то в зависимости от половины регистра (левой или правой) мы говорим о высоком или низком полубайте соответственно (рис. 1.8).

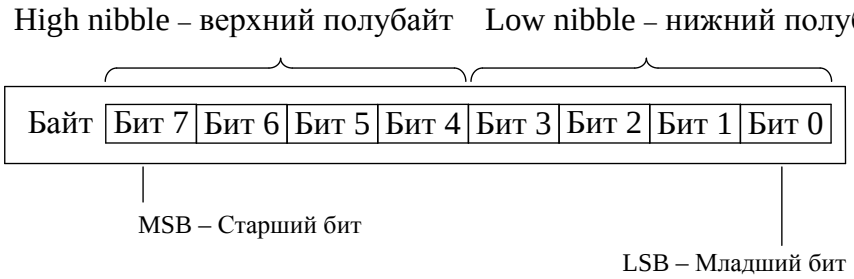


Рис. 1.8. Маркировка битов в байте

Булева алгебра и логические уравнения

Джордж Буль (1815–1864 гг.) – английский логик и математик, самоучка, который никогда не посещал колледж. В 1847 г. он опубликовал «Математический анализ логики», а в 1854 г. – «Исследование законов мышления». Эти работы, в которых он создал алгебру логики (известная как булева алгебра), связали математику и логику.

Английские философы и математики Бертран Рассел (1872–1970 гг.) и Альфред Норт Уайтхед (1862–1947 гг.) опубликовали в 1910 г. первый том «Начала математики» из трех. Благодаря этой книге булева логика превратилась в мощную систему символьной логики. В своей магистерской диссертации в 1938 г. студент MIT Клод Е. Шеннон показал, что исчисление при помощи символической логики может быть реализовано на аппаратной базе рэлейных схем. Он назвал ее алгеброй рэлейных схем и показал, как эти логические уравнения позволяют создавать схемы, выполняющие такие операции, как сложение и вычитание двоичных чисел. В этой книге термины «Булева алгебра» и «Рэлейная алгебра» означают одно и то же.

Булевы теоремы

Теоремы булевой алгебры для операций с одной переменной представлены в табл. 1.2.

Таблица 1.2

Булевы теоремы с одной переменной

Название теоремы	**Для ИЛИ**	**Для И**
Тождество	x | 0 = x x | 1 = 1	x & 1 = x x & 0 = 0
Дополнение	x | ~x = 1	x & ~x = 0
Идемпотентность	x | x = x	x & x = x
Возведение в степень	~~x = x	–

Теоремы с одной переменной

В табл. 1.2 показаны булевы теоремы, содержащие в себе только одну логическую переменную. Все эти выражения легко доказать, показав, что каждое выражение справедливо для значений x = 0 и x = 1.

Принцип двойственности

Обратите внимание, что в табл. 1.2 теоремы для И отличаются от теорем для ИЛИ, в которых «|» меняется на «&», нули на единицы, а единицы на нули. Это общий принцип двойственности, который справедлив для любой булевой теоремы или тождества. Он гласит, что любая такая теорема или тождество остаются верным, если все символы «|» и «&» меняются местами и все нули и единицы меняются местами. В будущем мы столкнемся с принципом двойственности и в теоремах с двумя и тремя переменными.

Теоремы с двумя и тремя переменными

В табл. 1.3 показан список теорем булевой алгебры с двумя и тремя переменными. Они поделены на пять групп, рассмотрим каждую из них отдельно. Обратите внимание, что в каждой группе версия b является двойственной версии a.

Таблица 1.3

Булевы теоремы с двумя и тремя переменными

Название теоремы	**Пример**	**Формулировка**
Коммутативность	(1a) (1b)	x | y = y | x x & y = y & x
Ассоциативность	(2a) (2b)	x | (y | z) = (x | y) | z x & (y & z) = (x & y) & z
Дистрибутивность	(3a) (3b)	x | (y & z) = (x | y) & (x | z) x & (y | z) = (x & y) | (x & z)
Единство	(4a) (4b)	(x & y) | (~x & y) = y (x | y) & (~x | y) = y
Абсорбция	(5a) (5b)	x | (x & y) = x x & (x | y) = x
Абсорбция (вторая форма)	(6a) (6b)	x | (~x & y) = x | y x & (~x | y) = x & y

Коммутативность

Коммутативность (1a) и (1b) в табл. 1.3 довольно очевидна. Она верна, так как выходное значение z в двух средних строках таблицы истинности для коммутативного закона одинаковы. То есть значение z независимо от порядка x и y.

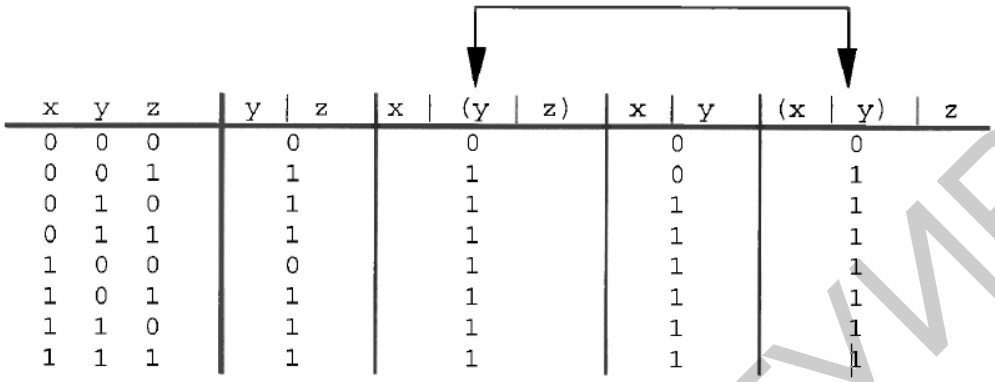
Ассоциативность

Ассоциативность (2a) и (2b) в табл. 1.3 утверждает, что для трех (и более) входных переменных порядок, в котором выполняются операции И/ИЛИ, не имеет значения. Мы всегда можем доказать любую из этих теорем, составив

таблицу истинности и проверив, что этот закон справедлив для всех позиций. На рис. 1.9 показана ассоциативность (2а).

Таким образом, x | (y | z) = (x | y) | z.

Для доказательства этой и других теорем из табл. 1.3 можно использовать достаточно полезный графический метод, называемый диаграммами Венна.



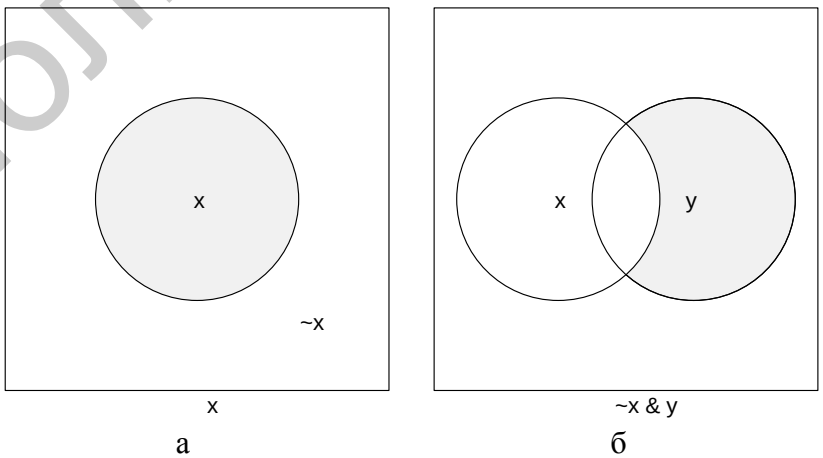
x	y	z	y | z	x | (y | z)	x | y	(x | y) | z
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Рис. 1.9. Таблица истинности для ассоциативного закона

Диаграммы Венна

Диаграммы Венна представляют собой графическое изображение булевых функций. Изображенная на рис. 1.10, а диаграмма Венна представляет логическую переменную x как круг в единичном квадрате. Область внутри круга считается «верным» значением переменной, т. е. x = 1. Область, лежащая за пределами круга x, представляет значения x = 0 или ~x.

Если у нас есть две переменные x и y, то каждая из них представляется кругом. Область внутри круга x представляет собой «верные» значения x, а область внутри круга y представляет собой «верные» значения y. Область, лежащая за пределами круга x, но в пределах круга y, представляет собой булеву функцию ~x & y, как показано на рис. 1.10, б.



а б

Рис. 1.10. Диаграммы Венна для логической переменной x и булевой функции ~x & y

Три логические переменные x, y и z будут иметь свои круги. Диаграммы Венна для булевых функций (x & y), (x & z), (y & z) и (x & y & z) показаны на

рис. 1.11. Обратите внимание, что в каждом случае закрашенная область представляет собой ту часть, в которой переменные одновременно являются «верными». Это значит, что обе (или все) переменные должны быть «верны», для того чтобы операция И была «верной».

Ассоциативность (2b) в табл. 1.3 подтверждается диаграммами Венна на рис. 1.11. Если вы примените И к x и области (y & z) на рис. 1.11, в, то получите ту же область (показанную на рис. 1.11, г), которую вы получили бы, если бы применили И к z и области (x & y) с рис 1.11, а.

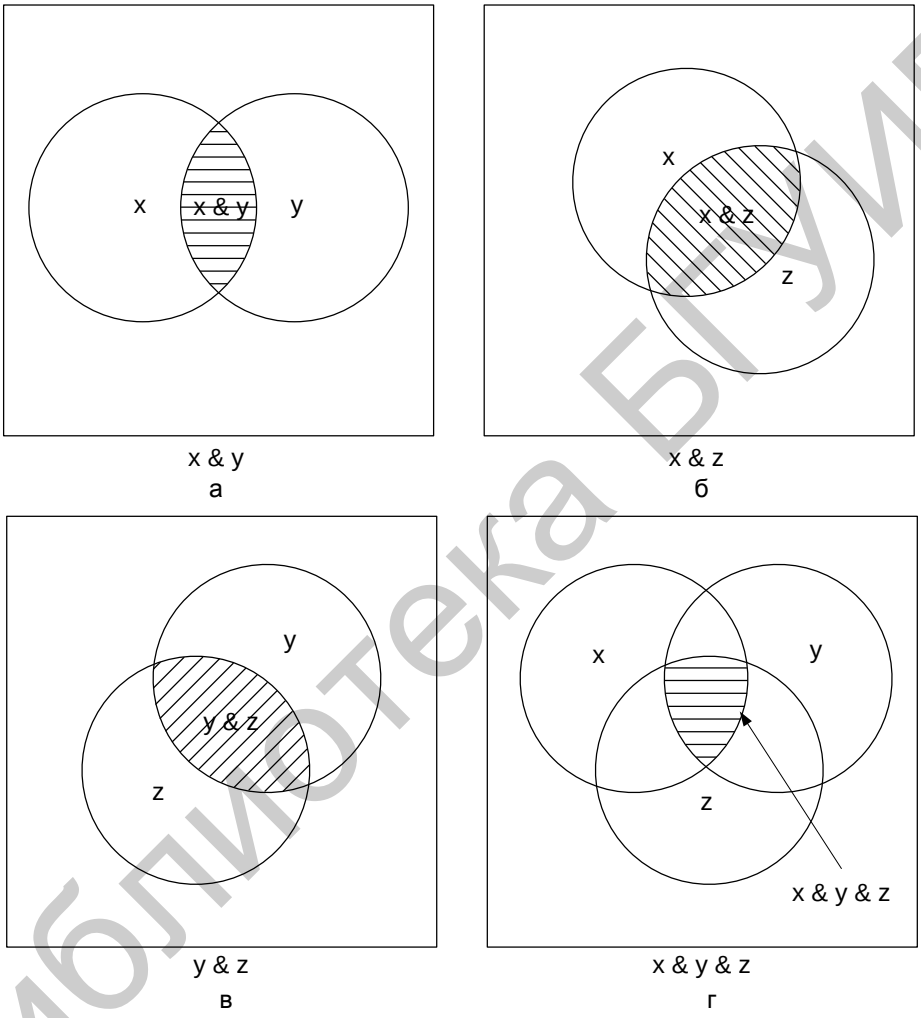


Рис. 1.11. Диаграммы Венна для операций И

Диаграммы Венна для операций ИЛИ показаны на рис. 1.12. Учтите, что в этом случае закрашенная область покрывает все логические переменные, так как оператор ИЛИ «верен», если хоть одна переменная верна. Ассоциативность (2a) в табл. 1.3, которую мы доказали посредством таблицы истинности на рис. 1.9, также справедлива и для диаграмм Венна на рис. 1.12. Не важно, добавим мы, применив операцию ИЛИ, x к (y | z) с рис. 1.12, в, или же мы добавим z к (x | y) с рис. 1.12, а – мы в любом случае получим полностью закрашенную область, как показано на рис. 1.12, г.

Дистрибутивность

Дистрибутивность (3a) и (3b) в табл. 1.3 можно также доказать посредством диаграмм Венна. На рис. 1.13, а показано, что если мы применим операцию ИЛИ к x и (y & z) с рис. 1.12, в, то мы получим ту же закрашенную область, как если бы мы применили операцию И к (x | y) с рис. 1.12, а и к (x | z) с рис. 1.12, б.

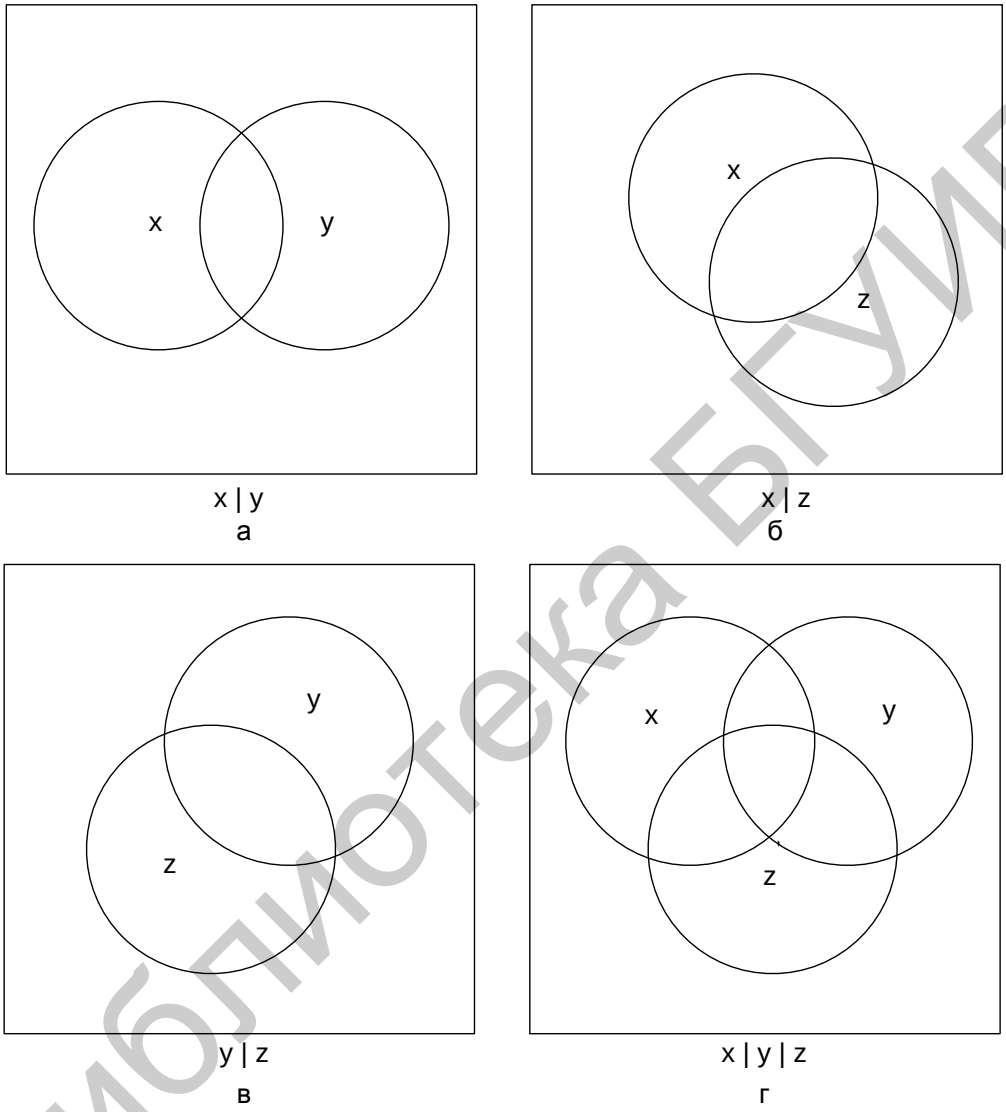


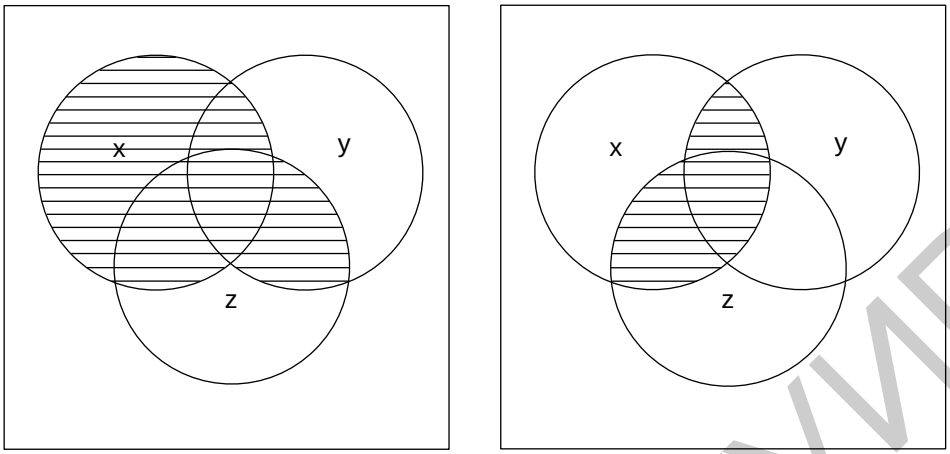
Рис. 1.12. Диаграммы Венна для операций ИЛИ

Аналогично закрашенную область на рис. 1.13, б можно получить, применив операцию И к x и (y | z) с рис. 1.12, в. Также ее можно получить, применив ИЛИ к областям (x & y) с рис. 1.12, а и (x & z) с рис. 1.12, б. Это и является доказательством дистрибутивности (3b) в табл. 1.3.

Обратите внимание, что дистрибутивность (3b) в табл. 1.3 похожа на дистрибутивность в обычной алгебре: $x \cdot (y + z) = x \cdot y + x \cdot z$.

В этом случае умножение x распределяется по различным дополнительным членам. В булевой алгебре не только операции И

распределены по различным членам ИЛИ, как в (3b) табл. 1.3, но и операции ИЛИ также распределены по различным членам операции И, как в (3a) табл. 1.3. Аналогичное выражение в обычной алгебре не верно.



а б

Рис. 1.13. Диаграммы Венна, иллюстрирующие дистрибутивность

Единство

Теорему единства (4a) в табл. 1.3 можно доказать несколькими способами. Заметьте, что если мы добавим (ИЛИ) закрашенную область (x & y) с рис. 1.11, а к закрашенной области (~x & y) с рис. 1.10, б, то мы получим просто y.

Мы также можем получить (4a) в табл. 1.3 из уже известных нам теорем. Можно записать:

(x & y) | (~x & y) = (y & x) | (y & ~x)	табл. 1.3 (1b)
= y & (x | ~x)	табл. 1.3 (3b)
= y & 1	табл. 1.2
= y	табл. 1.2

Аналогично для теоремы (4b) в табл. 1.3 мы можем записать:

(x | y) & (~x | y) = (y | x) & (y | ~x)	табл. 1.3 (1b)
= y | (x & ~x)	табл. 1.3 (3a)
= y | 0	табл. 1.2
= y	табл. 1.2

Можно заметить, что последняя теорема доказывается при помощи теоремы Венна. Такие теоремы единства часто называют комбинационными теоремами.

Данная в (4a) табл. 1.3 форма теоремы единства очень полезна для уменьшения количества членов логического уравнения. Если вы заметили произведение членов в сумме произведений выражения, которые различаются лишь одной переменной и ее дополнением, то вы можете вынести общую часть за скобки, и логическая сумма «|» переменной и дополнения будет равна единице.

Абсорбция

В табл. 1.3 представлены две формы абсорбции. В (5a,b) y поглощается, в то время как в (6a,b) поглощается ~x. Формы (5a,b) иногда называют теоремой перекрытий, когда говорится, что x перекрывает y.

Легко заметить, что если добавить x с рис. 1.10, а к закрашенной зоне (x & y) с рис. 1.11, а, то мы получим просто x. Аналогично, если применить И к x и (x | y) с рис. 1.12, а, то мы все равно получим x.

Если сложить закрашенные области с рис. 1.10, а (x) и рис. 1.10, б (~x & y), то мы получим область (x | y) с рис. 1.12, а, чем докажем теорему абсорбции (6a) в табл. 1.3.

Более подробное рассмотрение математических и аппаратных основ цифровой техники приведено в [2].

1.2.3. Базовые элементы микроконтроллеров

Внутреннее устройство электронных интегральных схем, микроконтроллеров и микропроцессоров, выполняющих сложные математические операции, является сравнительно простым. Как правило, всякое наиболее эффективное решение оказывается простым и «красивым». Все схемы состоят всего лишь из нескольких базовых логических схем, называемых логическими элементами.

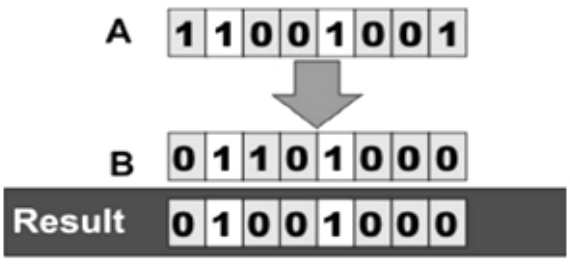
Операции, которые производят эти элементы, базируются на принципах, разработанных Джорджем Булем еще в середине XIX века, до изобретения первых ламп. Изначально предполагалось выразить логические формы через алгебраические функции. Такой подход нашел практическое применение, и, как следствие этого, появились элементы, известные как И, ИЛИ и НЕ. Принципы их работы сейчас называются алгеброй Буля (см. п. 1.2.2).

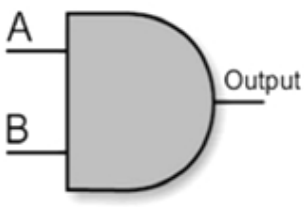
Логические элементы

Часть программных команд по своим функциям подобна логическим элементам. Принципы работы этих элементов рассмотрены ниже.

Элемент И (AND)

Логический элемент И (рис. 1.14) имеет два или более входов и один выход. Логическая единица на выходе будет лишь в том случае, если на оба входа (A и B) поступила единица. Таблица истинности показывает взаимную зависимость между входами и выходом.





A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Рис. 1.14. Логический элемент И

Когда мы используем программную реализацию, операция «логическое И» выполняется при помощи сравнения соответствующих битов двух регистров и выведения результата путем переноса значений совпавших единиц согласно таблице истинности.

Элемент ИЛИ (OR)

Подобно элементу И элемент ИЛИ (рис. 1.15) имеет два или более входов и один выход. Логическая единица (1) будет на выходе, если хотя бы на один из входов (в нашем случае A или B) будет подана логическая единица (1). Если на вход поступают нули (0), то и на выходе будут нули (0). Принципы программной реализации такие же, как и для элемента И. Реализация основана на побитном сравнении, с той лишь разницей, что переносится значение совпавших нулей.

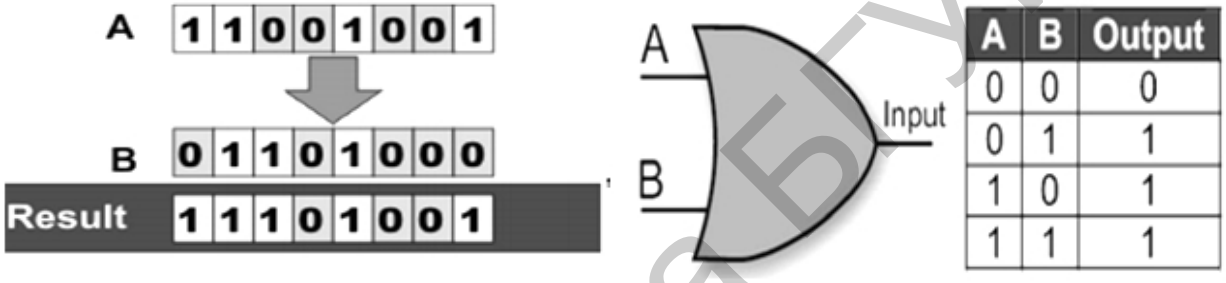


Рис. 1.15. Логический элемент ИЛИ

Элемент НЕ (NOT)

Логика работы элемента НЕ (рис. 1.16) предельно проста. У него всего один вход и выход. Если на вход поступает нуль (0), то на выходе будет единица (1), и наоборот. Это значит, что элемент НЕ инвертирует сигнал, поступающий на вход, поэтому данный элемент называют инвертором.

В программе операция НЕ производится над одним байтом. Результатом будет байт с инвертированными битами. Если байт является числом, то его инверсия будет его двоичным дополнением до единицы. Если сложить восьмибитное число с его двоичным дополнением, получится максимальное восьмибитное число. Иными словами, сумма восьмизначного числа и числа, комплементарного ему, будет всегда равна 255.

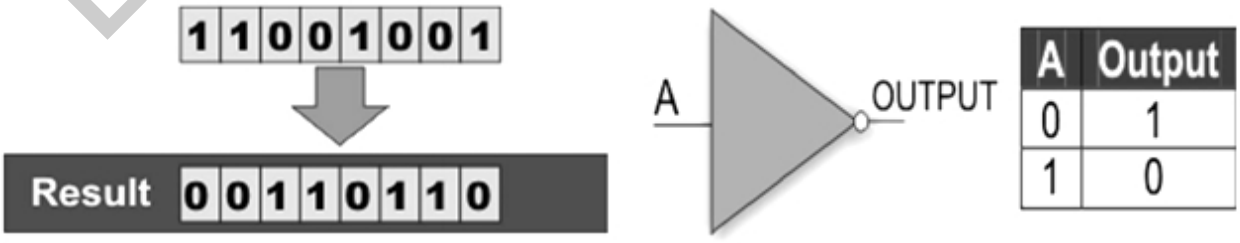


Рис. 1.16. Логический элемент НЕ

Элемент исключающее ИЛИ (XOR)

Элемент исключающее ИЛИ (рис. 1.17) намного сложнее в сравнении с другими элементами. Он представляет собой комбинацию всех этих элементов. Единицу (1) на выходе можно получить лишь в том случае, когда входы находятся в разных логических состояниях.

Программная реализация основывается на сравнении двух байтов. Для этих целей может быть использовано вычитание (если результат нуль (0), байты одинаковы). Но в отличие от вычитания при использовании логической операции XOR мы не можем получить отрицательный результат.

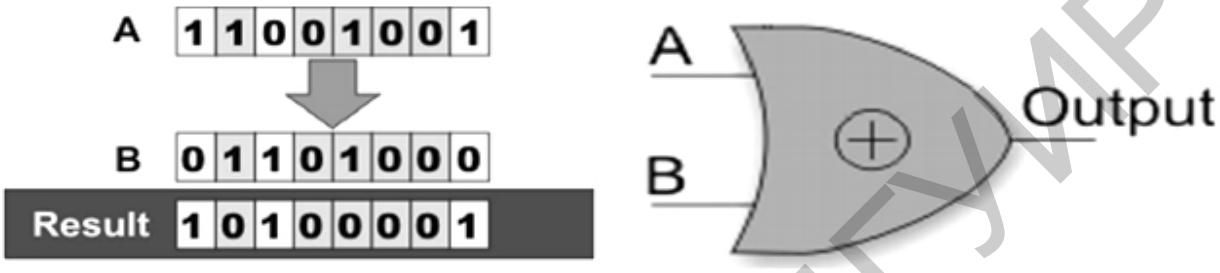


Рис. 1.17. Логический элемент исключающее ИЛИ

Регистр

Регистр (или ячейка памяти) – это электрическая схема, которая может запоминать состояние одного байта (рис. 1.18).

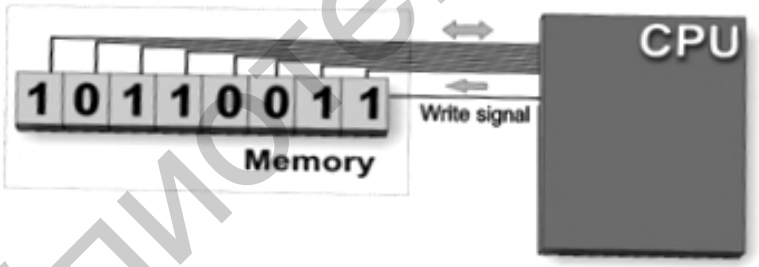


Рис. 1.18. Регистр микроконтроллера

SFR-регистр

В дополнение к обычным регистрам, которые не имеют специальных заранее установленных функций, каждый микроконтроллер оснащен определенным числом SFR-регистров (Special Function Register, рис. 1.19), чьи функции заданы еще во время производства. Их биты связаны (в буквальном смысле) с внутренними компонентами микроконтроллера, такими как таймеры, АЦП, осцилляторы и т. п. Это означает, что они напрямую управляют этими устройствами, т. е. по сути управляют микроконтроллером. Представьте себе восемь переключателей, которые контролируют работу небольшой цепи в микроконтроллере – именно это и делают SFR-регистры.

Другими словами, состояния битов в регистре меняются из программы, регистр запускает небольшие цепи в микроконтроллере, эти цепи соединяются с помощью контактов микроконтроллера с периферийными устройствами, которые решают поставленную задачу под контролем управляющей программы.

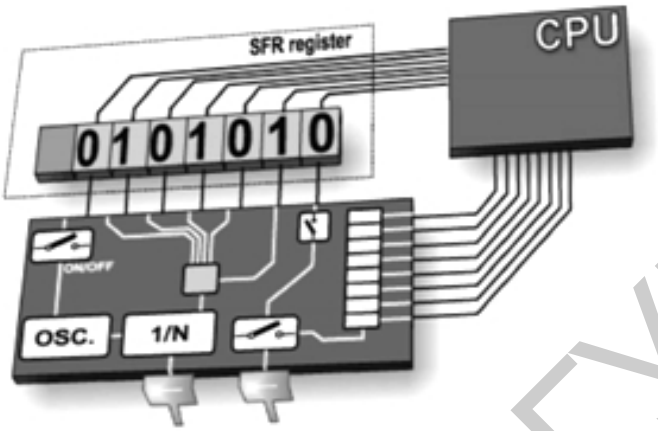


Рис. 1.19. Регистр микроконтроллера специального назначения

Порты ввода-вывода

Для того чтобы сделать микроконтроллер полезным, он должен быть соединен с периферийными устройствами. Каждый микроконтроллер имеет один или более регистров, которые называются портами, соединенными с контактами микроконтроллера (рис. 1.20).

Почему это называется «порт ввода-вывода»? Потому что вы можете изменить функцию этого контакта как угодно. Например, допустим, вы хотите, чтобы ваше устройство включало/выключало три светодиода и одновременно следило за логическим состоянием пяти датчиков или кнопок. Порты должны быть настроены соответствующим образом: три вывода (для соединения со светодиодами) и пять портов ввода (для кнопок). Подобную операцию можно легко проделать с помощью программного обеспечения, при этом функции портов можно менять под свои нужды, когда угодно.

Одно из важнейших технических требований к контактам ввода-вывода – это максимальное значение тока, которое они могут выдержать. Для большинства микроконтроллеров достаточно силы тока, которая могла бы активировать светодиод или любое другое слаботочное устройство (10–20 мА). Чем больше портов ввода-вывода, тем ниже максимальный ток на одном контакте. Другими словами, максимальный ток, указанный в паспортных данных микропроцессора, одинаков для всех портов ввода-вывода.

Другой важной особенностью контакта является возможность подключения нагрузочных элементов. Эти элементы (резисторы) подключаются к плюсу питания и начинают работать, когда к порту подключаются механические переключатели или кнопки. Последние версии микроконтроллеров позволяют настраивать подобные элементы программно.

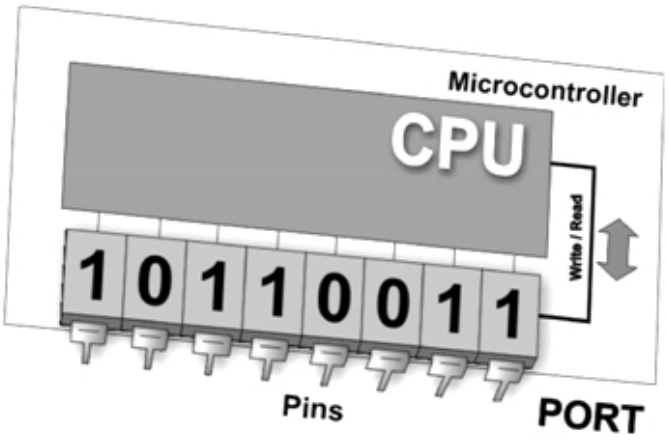


Рис. 1.20. Порт ввода-вывода микроконтроллера

Каждый порт ввода-вывода обычно находится под контролем специального SFR-регистра, что значит каждый бит этого регистра отвечает за состояние опрашиваемого порта микроконтроллера. Например, после записи логической единицы (1) в бит регистра управления соответствующий порт будет автоматически сконфигурирован как вход, а поданное на него напряжение будет распознано как 0 или 1. В противном случае в SFR-регистр будет записан нуль (0) и порт будет настроен как выход. Напряжение (0 В или 5 В) будет соответствовать состоянию определенного бита регистра порта.

Запоминающее устройство (ЗУ)

Память – это часть микроконтроллера, используемая для хранения данных (рис. 1.21).

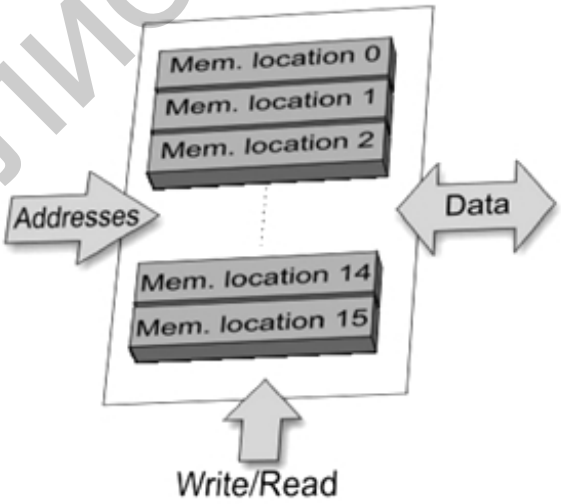


Рис. 1.21. Память микроконтроллера

Самый простой способ объяснить работу ЗУ – это сравнить его со шкафом с множеством выдвижных ящиков. Предположим, что ящики подписаны и мы можем легко узнать их содержимое, просто прочитав соответствующую метку.

Аналогично этому каждый адрес памяти соответствует определенной ячейке памяти. К содержимому каждой ячейки можно получить доступ, зная адрес этой ячейки. Мы можем записывать и считывать данные. В микроконтроллере применяется несколько разных типов памяти:

1. *Read Only Memory (ROM), или постоянное запоминающее устройство (ПЗУ)*. ПЗУ используется для постоянного хранения исполняемой программы. Размер исполняемой программы зависит от объема памяти. Современные микроконтроллеры используют 16-битную адресацию, а это 64 килобайта памяти или 65536 ячеек памяти. Как правило, простая программа редко будет превышать сотню инструкций/команд.

2. *Masked ROM (MROM).* MROM – это один из видов ПЗУ, который заранее запрограммирован изготовителем. Термин «masked – скрытая» возник из-за процесса производства, когда некоторые области платы закрывались до начала процесса фотолитографии. В промышленных масштабах цена подобных продуктов была довольно низкой.

3. *One Time Programmable ROM (OTP ROM), или программируемое ПЗУ (ППЗУ)*. Данный тип памяти позволяет вам записать в нее программу, но, как следует из названия, всего один раз. Если ошибка в программе была обнаружена уже после загрузки, то единственное, что вы можете сделать, это записать верную программу на другой чип.

4. *UV Erasable Programmable ROM (UVEPROM), или УФ-перепрограммируемое ПЗУ (ПППЗУ)*. Процесс производства и характеристики этого типа памяти идентичны ППЗУ. Конструкция микроконтроллера с этим типом памяти имеет характерное «окно», расположенное сверху (рис. 1.22).

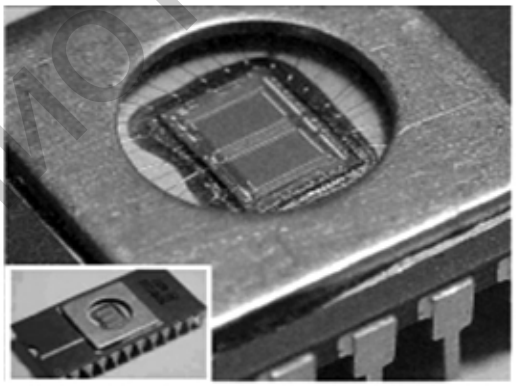


Рис. 1.22. УФ-перепрограммируемая память МК

Это позволяет стирать данные под воздействием мощного ультрафиолетового (УФ) излучения. Спустя несколько минут после этой процедуры можно снова записывать исполняемую программу. Установка этого «окна» – процесс весьма сложный, что негативно сказывается на стоимости данного товара.

5. *Flash-память*. Этот тип памяти был разработан еще в 80-е гг. в лабораториях INTEL и был представлен как преемник UVEPROM. Поскольку

содержимое этой памяти может быть записано, прочитано и стерто «почти» бесконечное число раз, микроконтроллеры с таким типом памяти идеально подходили для обучения, экспериментирования и мелкосерийного производства. Flash-память обрела большую популярность: большинство микроконтроллеров до сих пор выпускаются именно с этим типом памяти.

6. *Random Access Memory (RAM), или запоминающее устройство с произвольным доступом (выборкой), или оперативная память*. Всякий раз, когда прекращается подача напряжения, содержимое RAM очищается. Этот тип памяти используется для хранения данных и промежуточных результатов вычислений, которые используются в дальнейшем микроконтроллером. Например, если программа выполняет сложение каких-либо чисел, то нам необходим регистр, который представлял бы собой то, что мы в повседневной жизни назвали бы «сумма». На этот случай у нас есть один из регистров RAM, который называется «сумма» и используется для хранения результатов сложения.

7. *Electrically Erasable Programmable ROM (EEPROM), или электрически стираемое перепрограммируемое ПЗУ*. Содержимое EEPROM может меняться во время работы (по аналогии с RAM), но данные сохранятся даже в случае потери напряжения (по аналогии с ROM). Значит, EEPROM используется для хранения данных, которые появляются по ходу работы, и которые должны быть сохранены. Например, если вы разрабатываете электронный замок или сигнализацию, желательно дать возможность пользователю создать и сохранить пароль, но он бесполезен, если будет стираться каждый раз, когда отключается питание. Идеальным решением будет использование микроконтроллера с EEPROM-памятью.

Современные микроконтроллеры одновременно содержат несколько (обычно четыре) различных типов памяти, предназначенных для разных целей.

Регистровая память является основой работы процессорного устройства. Количество регистров таково, чтобы обеспечить обслуживание аппаратной реализации разнообразного функционала, например выполнение арифметико-логических операций, хранение текущего состояния, буферизацию в приемопередающих модулях и пр. В некотором смысле весь микропроцессор – это набор регистров памяти и путей передачи данных между ними.

Оперативная память RAM определяет вычислительную мощность микропроцессора, так как ограничивает максимальную «ширину» исполняемой программы. Типичное количество RAM в современном микроконтроллере – единицы килобайтов.

Стек – особая часть регистровой или RAM-памяти, предназначенная для хранения текущего состояния процессора при переходе к выполнению вложенных подзадач, тем самым определяющая «глубину» программы.

Постоянная flash-память хранит исходную программу и данные, тем самым ограничивает разнообразие программного функционала, т. е. «длину» программы. Типичный объем – десятки и сотни килобайтов.

Энергонезависимая EEPROM-память, в объеме, равном RAM или меньше.

Central Processor Unit (CPU), или центральный процессор (ЦП)

Как подсказывает само название, ЦП – это часть микроконтроллера, которая контролирует все процессы в нем. Он состоит из нескольких узлов (элементов, рис. 1.23), самые важные из которых:

1. *Командный дешифратор* – это электронное устройство, которое расшифровывает программные инструкции и, основываясь на этих инструкциях, запускает другие схемы (узлы, элементы). «Набор инструкций», который различается в зависимости от семейства микроконтроллеров, показывает возможности данной схемы.

2. *Арифметико-логическое устройство* (АЛУ) производит все математические и логические операции над данными.

3. *Аккумулятор* – SFR-регистр, работа которого тесно связана с АЛУ. Это своего рода рабочий стол, на котором хранятся все данные, над которыми будут производиться какие-либо операции (сложение, сдвиг/перемещение и т. д.). Также он хранит и результаты уже произведенных операций для их дальнейшей обработки. Один из SFR-регистров, который называется регистр-статуса (PSW), имеет близкое отношение к аккумулятору. В любой момент времени он показывает статус числа, которое хранится в аккумуляторе (оно больше нуля или нет и т. п.). Аккумулятор также называют рабочим регистром и помечают его как W-регистр, или просто W.

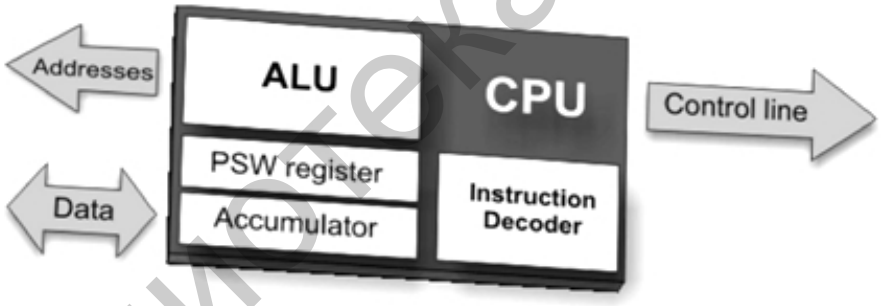


Рис. 1.23. Центральный процессор микроконтроллера

Шина

Шина состоит из 8, 16 или более проводов. Есть два типа шин: шина адреса и шина данных. Шина адреса состоит из такого количества линий, которое необходимо для адресации памяти. Она используется для передачи адресов от ЦП к памяти. Шина данных должна иметь ширину, как и ширина самих данных, в нашем случае шина должна быть 8-битной или шире. Она используется для соединения узлов в микроконтроллере.

Последовательная связь

Параллельное соединение между микроконтроллером и периферией через порты ввода-вывода это идеальное решение для коротких дистанций (до нескольких метров). Однако в других случаях, когда необходимо соединить два

устройства на большой дистанции, использовать параллельное соединение нельзя. Вместо него используется последовательное соединение.

В наши дни микроконтроллеры имеют несколько различных базовых вариантов последовательного соединения. Какой из вариантов будет использован, зависит от множества факторов, в первую очередь от:

- количества устройств, с которыми микроконтроллер будет обмениваться данными;
- скорости обмена данными между этими устройствами;
- расстояния между устройствами;
- необходимости одновременно отправлять и получать данные.

Один из самых важных моментов, касающихся последовательного соединения, – это строжайшее соблюдение определенного протокола (набора инструкций, которых должны придерживаться устройства, чтобы правильно интерпретировать данные, которыми они обмениваются). Но микроконтроллер сам заботится об этом, так что работа пользователя сводится к тому, чтобы записать данные (для отправки) и считать полученные данные.

Прерывания

В ходе выполнения большинства программ используются прерывания. То есть цель работы микроконтроллера – реагирование на изменения окружающих его условий. Другими словами, когда происходит событие, микроконтроллер должен как-то отреагировать. Например, когда вы нажимаете кнопку на пульте дистанционного управления, микроконтроллер должен это зафиксировать и переключить канал, изменить громкость звука и т. д. Если бы микроконтроллер тратил очень много времени (часы или дни) на проверку всего нескольких кнопок, вряд ли он нашел бы какое-нибудь практическое применение.

Именно поэтому микроконтроллеры в процессе развития получили следующую возможность: вместо постоянных проверок каждого порта/контакта или бита микроконтроллер «поручает ожидание вопроса специалисту», который будет реагировать только тогда, когда произойдет что-то действительно стоящее внимания микроконтроллера. Сигнал, который информирует ЦП о таком событии, называется прерыванием.

1.2.4. Основы языка microC

Языки программирования

Микроконтроллер выполняет программы, загруженные в его flash-память. Это исполняемый код, который содержит последовательность нулей и единиц. Он может состоянть из 12-, 14- или 16-битных слов в зависимости от архитектуры микроконтроллера. Каждое слово используется ЦПУ в виде команды, выполняемой во время работы микроконтроллера. Исходя из практических соображений исполняемый код часто представляется в виде последовательности шестнадцатеричных чисел, называемых HEX-код (рис. 1.24). Все инструкции, которые микроконтроллер может распознать, называются

набором команд. Что касается микроконтроллеров PIC, в общей сложности набор команд имеет 35 различных инструкций.

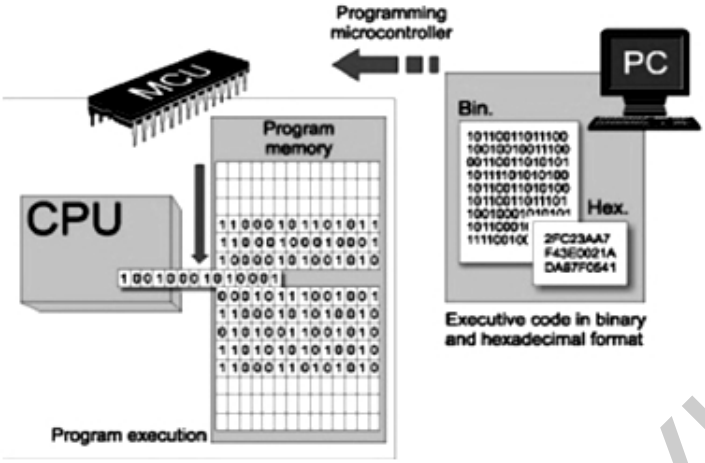


Рис. 1.24. Программирование микроконтроллера в машинных кодах

Ранние программы составлялись в HEX-коде вручную. Так как процесс написания исполняемого кода был слишком утомительным, был создан первый «высший» язык программирования, названный ассемблером. Он сделал процесс программирования более сложным за счет дополнительныех этапов разработки программы (рис. 1.25) и более быстрым за счет удобства записи и чтения. Подробнее программирование микроконтроллеров PIC на ассемблере рассмотрено в [3].

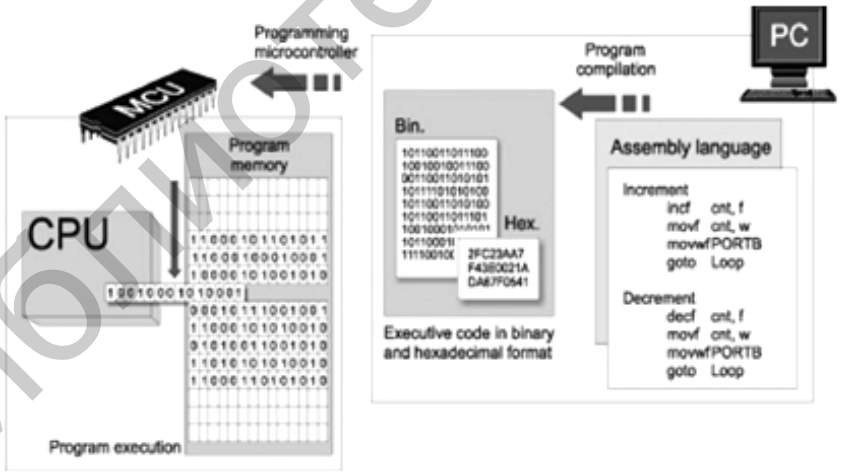


Рис. 1.25. Программирование микроконтроллера на ассемблере

Инструкции в ассемблере представлены в виде значимых сокращений, и процесс их компиляции в исполняемый код перенесен на специальную программу на ПК, названную компилятором. Основное преимущество этого языка программирования – его простота, т. е. каждой команде программы соответствует одна ячейка памяти в микроконтроллере. Это позволяет полностью контролировать то, что происходит внутри чипа, что делает этот

язык широко используемым и сегодня. Тем не менее, программистам всегда нужен язык программирования, близкий к языку, используемому в повседневной жизни. В результате были созданы языки программирования высокого уровня абстракции. Одним из них является С. Главным преимуществом этих языков является простота написания программы (рис. 1.26). Недостатком – то, что невозможно точно знать, как выполняется каждая команда.

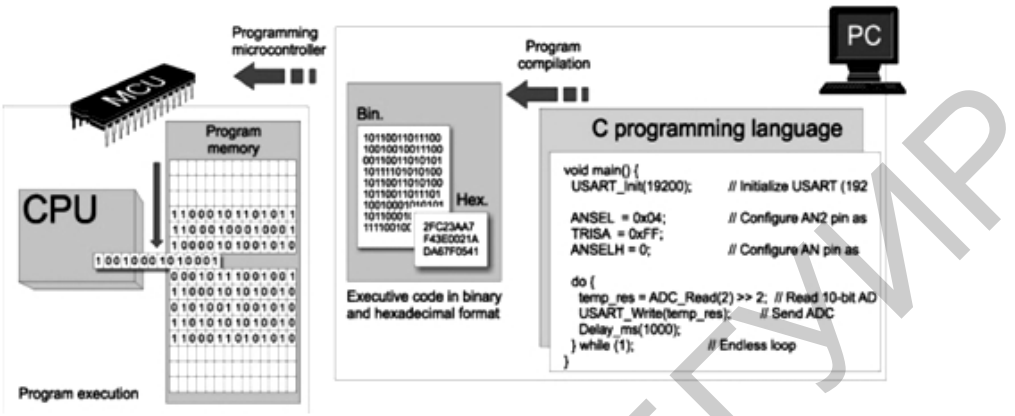


Рис. 1.26. Программирование микроконтроллера на С

Аналогично ассемблеру специализированная программа на ПК, называемая компилятором, отвечает за компиляцию программы на машинный язык (сборка и перевод программы, рис. 1.27 и 1.28). В отличие от ассемблера она создает исполняемый код, который не всегда является кратчайшим.

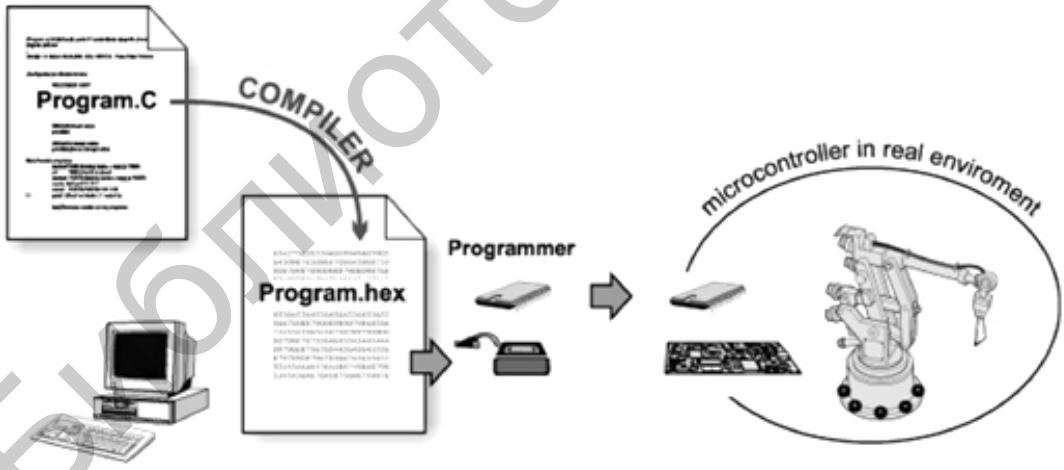


Рис. 1.27. Место компиляции в цикле разработки программ для микроконтроллера

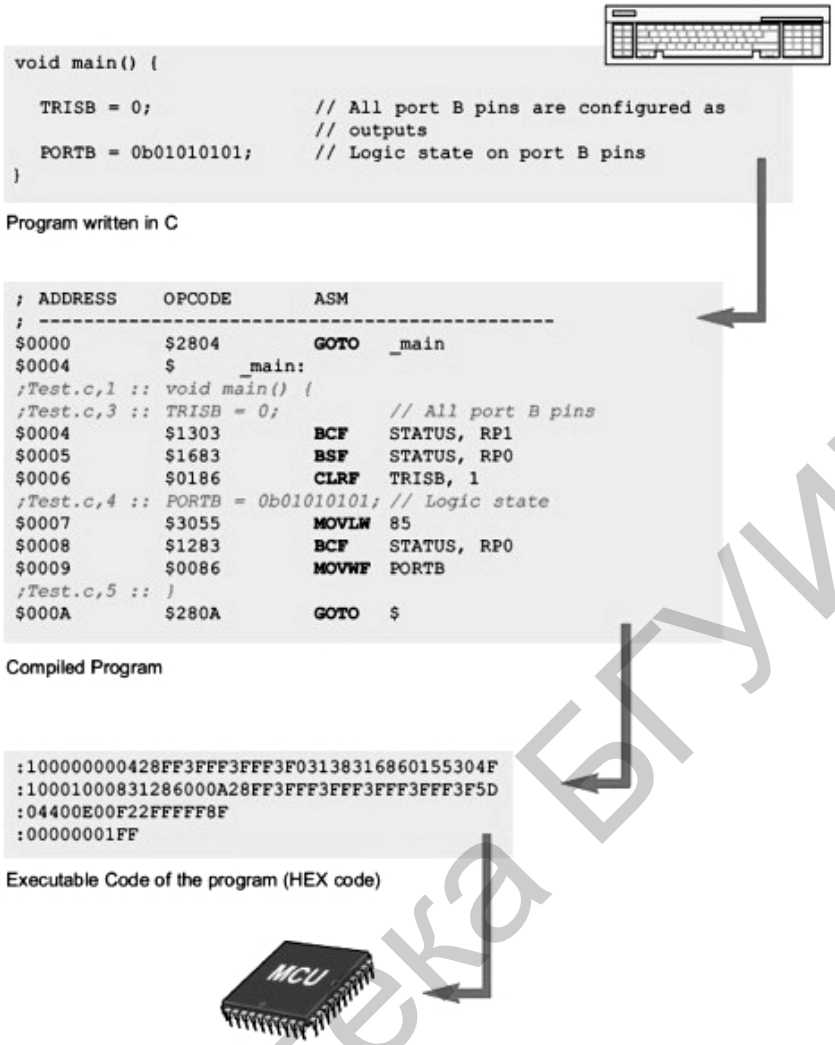


Рис. 1.28. Преобразования программы от высшего к низшему языку программирования

Преимущества языков программирования высокого уровня

Если вы когда-либо писали программу для микроконтроллера на языке ассемблер, то вы, наверное, знаете, что архитектуре RISC не хватает инструкций. Например, нет специальной инструкции для умножения двух чисел. Однако для решения сложных проблем часто используется следующий прием: одна сложная операция разбивается на несколько простых. В частности, умножение может быть легко заменено последовательным многократным сложением ($A \cdot B = A + A + A + ... + A$). Решение этой задачи на языке ассемблера требует довольно много усилий. В то же время языки программирования высокого уровня, такие как C, уже содержат готовые решения этой и других подобных проблем. Пример умножения двух чисел представлен на рис 1.29.

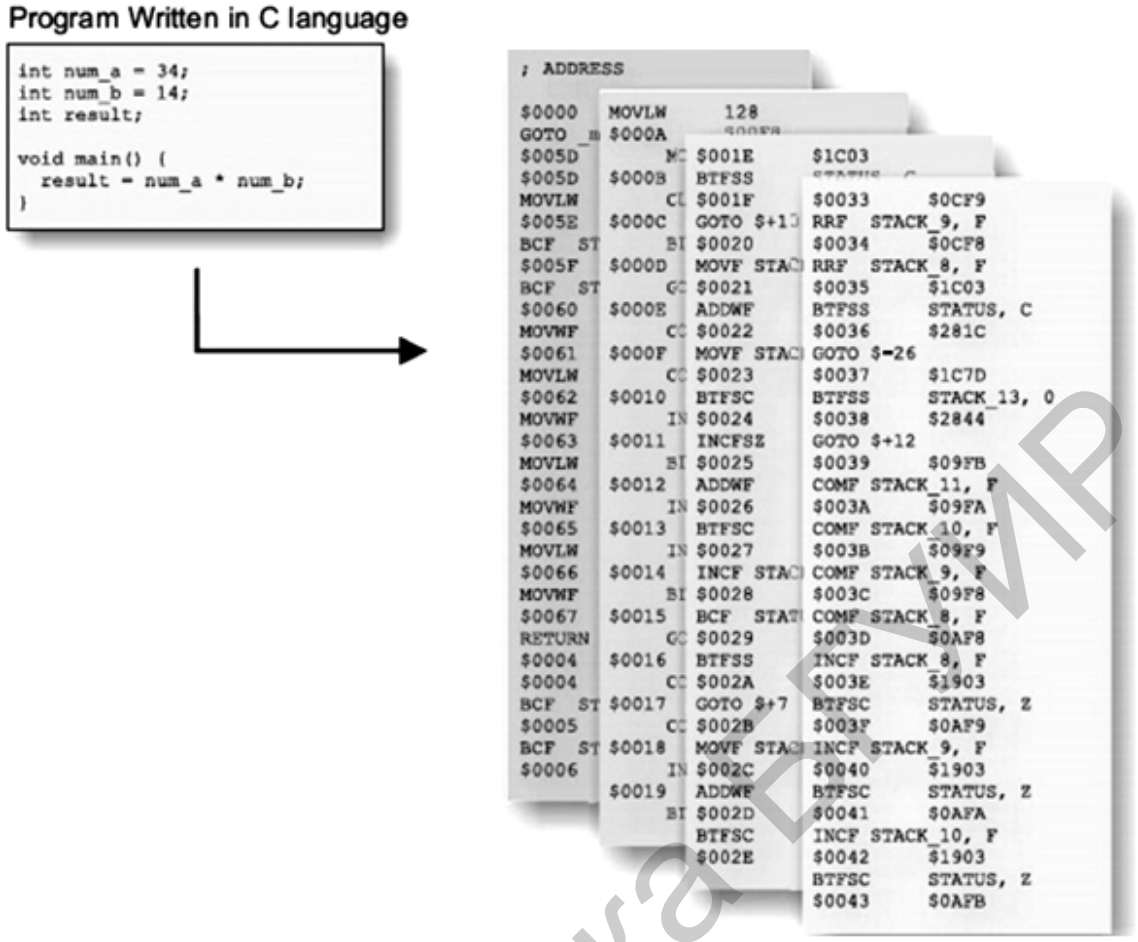


Рис. 1.29. Сравнительный пример умножения двух чисел на С и ассемблере

Препроцессор

Препроцессор – неотъемлемая часть компилятора С. Его функция заключается в обнаружении и исполнении инструкций препроцессора. Это специальные инструкции, которые не принадлежат к языку С, но как часть пакета программного обеспечения поставляются вместе с компилятором. Каждая команда препроцессора начинается с «#». До компиляции программы компилятор С активизирует препроцессор, который идет в рамках программы в поисках этих признаков. Если встретится какой-либо признак, то препроцессор будет просто заменять его на другой текст, который в зависимости от типа команды может быть файлом содержимого или только короткой последовательностью символов. Затем процесс компиляции можно начинать. Инструкции препроцессора могут быть где угодно в программе и относятся только к части программы после их появления до конца программы.

Директива препроцессора #include

Многие программы часто повторяют один и тот же набор команд несколько раз. Для того чтобы ускорить процесс написания программы, эти команды и заявления, как правило, группируются в файлы, которые могут быть легко включены в программу, используя эту директиву. То есть команда *#include*

импортирует текст из другого документа (рис. 1.30) независимо от того, что это (команды, комментарии и т. п.), в программу (как будто копирует в ту позицию, где написана).

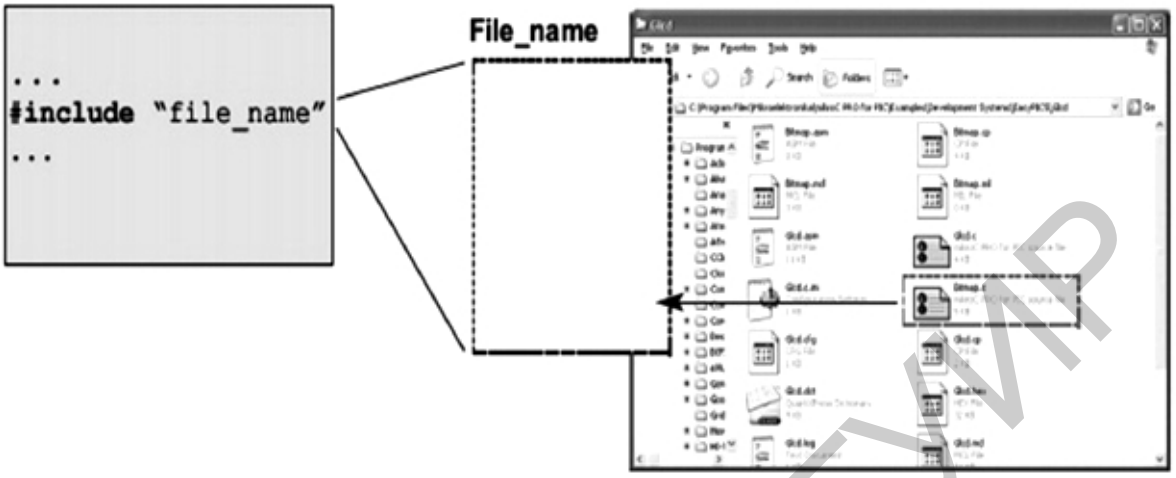


Рис. 1.30. Импорт текста директивой *#include*

Директива препроцессора #define

Команда *#define* предоставляет значение макроса, заменяя идентификаторы в программе на их значения (просто подстановка текста, указанного после пробела; для многострочной записи в конце строки ставится символ «\»):

#define symbol sequence_of_characters

Например:

...

#define PI 3.14

...

Языки программирования высокого уровня не связаны с каким-либо особым типом компьютеров, процессоров или операционных систем. Язык С является на самом деле языком общего назначения. Тем не менее именно этот факт может вызвать некоторые проблемы в процессе работы, так как язык С немного варьируется в зависимости от его применения, что можно сравнить с разными диалектами одного естественного языка.

Основы языка программирования С

Основная идея написания программы на языке С состоит в том, чтобы разбить большую «проблему» на несколько более мелких частей. Предположим, что необходимо написать программу для микроконтроллера, который собирается измерять температуру и отображать результаты на ЖК-дисплее (рис. 1.31).

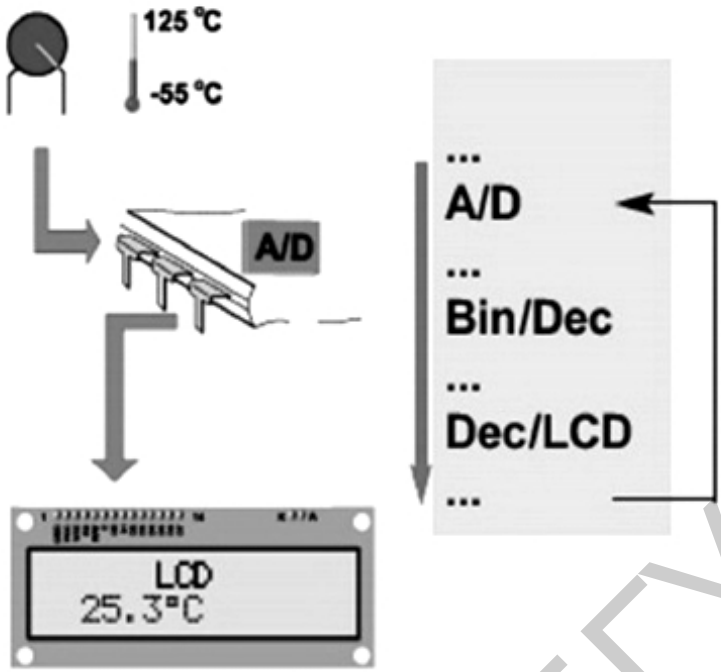


Рис. 1.31. Этапы процесса измерения температуры

Процесс измерения выполняется с помощью датчика, который преобразует температуру в электрическое напряжение. Микроконтроллер использует встроенный АЦП для преобразования этого напряжения (аналоговое значение) в числа (цифровое значение), которое затем отправляется на ЖК-дисплей с помощью нескольких проводников. Соответственно, программа делится на четыре этапа, которые следует пройти в следующем порядке:

1. Активирование и установка встроенного АЦП.
2. Измерение аналогового значения.
3. Расчет температуры.
4. Передача данных в надлежащей форме на ЖК-дисплей.

Как видно, языки программирования высокого уровня, такие как С, позволяют легко решить эту «проблему», написав четыре функции, которые будут выполняться циклически снова и снова.

Здесь описывается очень конкретное применение языка программирования С, используемого для компилятора mikroC PRO for PIC. В этом случае компилятор используется для программирования микроконтроллеров PIC. Это означает, что некоторая часть информации о языке С не включена в описание, потому что не имеет практического применения на изучаемом уровне разработки встраиваемых систем. С другой стороны, описаны некоторые моменты, которые понятны только компилятору microC и могут отсутствовать или иметь другое значение для иных компиляторов языка С.

На схеме рис. 1.32 представлена структура простой программы, поясняющая составляющие части любой программы.

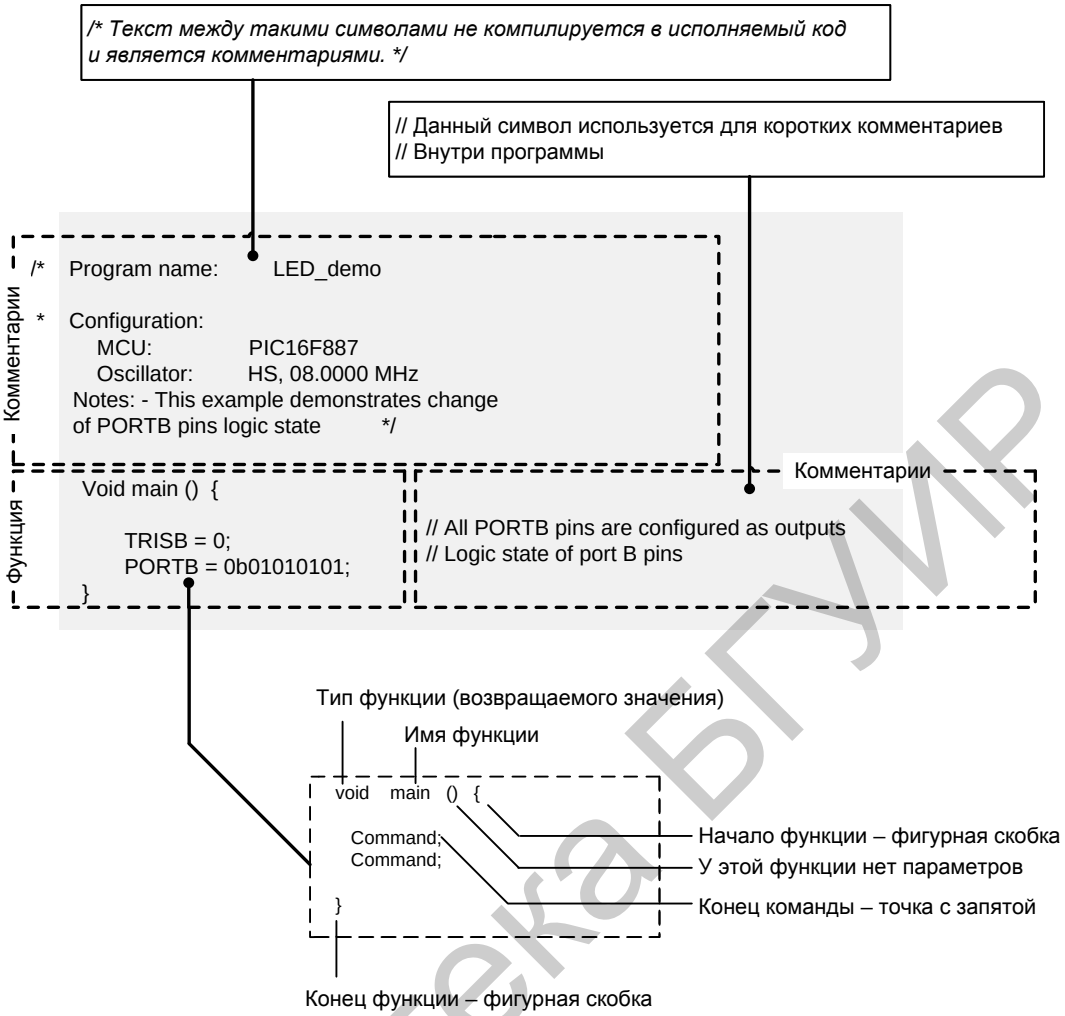


Рис. 1.32. Структура программы на языке С

Комментарии

Комментарии являются частью программы, применяемой для уточнения работы программы или, другими словами, для предоставления более подробной информации о ней. Комментарии игнорируются и не собраны в исполняемый код с помощью компилятора. Иными словами, компилятор распознает специальные символы, обозначающие начало и конец комментария, и полностью игнорирует текст комментария во время компиляции. Существуют два типа таких знаков. Одни обозначают длинные комментарии, проходящие несколько программных строк, в то время как другие обозначают короткие комментарии, занимая одну строку. Несмотря на то что комментарии не могут повлиять на выполнение программы, они так же важны, как и любая другая часть программы, по следующей причине.

Написанную программу всегда можно улучшить, изменить, модернизировать, упростить. Почти каждую программу подвергают доработке. Понять даже самые простые программы без комментариев почти невозможно. Комментарии для начинающих программистов поясняют, что делает опреде-

ленный участок кода, для чего он предназначен. Комментарии для профессионалов не поясняют язык, на котором написана программа, они поясняют ее суть, причину написания участка программного кода или неявные особенности его работы, что нужно учесть при его изменении.

Типы данных в языке С

Есть несколько типов данных, которые могут быть использованы в языке программирования. Приведенная ниже табл. 1.4 показывает диапазон значений, которые эти данные могут иметь при использовании их в основной форме.

Таблица 1.4

Некоторые типы данных языка С

Тип данных	**Описание**	**Размер (количество битов)**	**Пределы значений**
char	Символьный	8	0–255
int	Целый	16	–32768–32767
float	С плавающей точкой	32	$\pm1{,}17549435082\cdot10^{-38}$– $\pm6{,}80564774407\cdot10^{38}$
double	С плавающей точкой двойной точности	32	$\pm1{,}17549435082\cdot10^{-38}$– $\pm6{,}80564774407\cdot10^{38}$

При добавлении префикса для любого типа данных увеличивается диапазон его возможных значений, так же как и количество занимаемой памяти, как это показано в табл. 1.5.

Переменные

Любое число, которое изменяет свое значение в ходе работы программы, называется переменной. Иными словами, если программа добавляет два числа (номер 1 и номер 2), необходимо иметь значение для представления того, что мы в повседневной жизни называем суммой. В этом случае число 1, число 2 и сумма – переменные.

Таблица 1.5

Префиксы типов данных языка С

Тип данных	**Тип данных с префиксом**	**Размер (количество битов)**	**Пределы значений**
char	signed char	8	–128–128
int	unsigned int	16	0–65535
	short int	8	0–255
	signed short int	8	–128–127
	long int	32	0–42
	signed long int	32	–2147483648– 2147483647

Объявление переменных

Имя переменной может включать любой из алфавитных символов AZ («Аризона»), цифры 0–9 и символ подчеркивания «_». Компилятор чувствителен к регистру и различает прописные и строчные буквы. Функции и имена переменных, как правило, содержат строчные символы, в то время как имена констант содержат только символы верхнего регистра.

Имена переменных не должны начинаться с цифр.

Некоторые из названий не могут быть использованы в качестве имен переменных как уже используемых самим компилятором. Такие имена называются ключевые слова. Компилятор microC признает в общей сложности 33 таких слова, представленных в табл. 1.6.

Таблица 1.6

Ключевые слова языка microC

absolute	data	if	return	typedef
asm	default	inline	rx	typeid
at	delete	int	sfr	typename
auto	do	io	short	union
bit	double	long	signed	unsigned
bool	else	mutable	sizeof	using
break	enum	namespace	static	virtual
case	explicit	operator	struct	void
catch	extern	org	switch	volatile
char	false	pascal	template	while
class	float	private	this	–
code	for	protected	throw	–
const	friend	public	true	–
continue	goto	register	try	–

Указатели

Указатель – особый тип переменной, содержащей адрес символьных переменных. Другими словами, указатель «указывает на» другую переменную. Он объявляется следующим образом:

*type_of_variable *pointer_name;*

Для того чтобы сохранить адрес переменной в указатель, необходимо использовать символ «=» и написать символ «&» перед переменной. В следующем примере указатель «multiplex» объявляется и назначается адрес первого из восьми светодиодных дисплеев:

```
unsigned int *multiplex;   // Объявление имени и типа указателя multiplex
multiplex = &display1;     // адресу переменной display1 присваивается
                           // указатель multiplex
```

Чтобы изменить значение указываемой переменной, достаточно написать «*» символ перед ее указателем и присвоить через указатель новое значение:

```
*multiplex = 6;          // Переменной display1 присвоено значение 6
```

Таким же образом, чтобы прочесть значение указываемой переменной, достаточно написать

```
temp = *multiplex;       // Значение переменной display1 скопировано в
                         //temp
```

Изменение отдельных битов

Есть несколько способов, чтобы изменить только один бит переменной. Простейшим является указать имя регистра, позицию бита или имя и желаемое состояние:

```
(PORTD.F3 = 0) ;         // Очистка бита RD3
...
(PORTC.RELAY = 1) ;      // Установка значения выходного бита PORTC
                         // (предварительно бит был назван RELAY)
                         // RELAY должно быть объявлено константой
```

Объявления

Каждая переменная должна быть объявлена до использования в первый раз в программе. Так как переменные хранятся в оперативной памяти, необходимо зарезервировать место для них (один, два или несколько байтов). Вы знаете, какой тип данных вы пишете или ожидаете в результате операции, в то время как компилятор не знает этого. Не забывайте, что программа имеет дело с переменными, которым вы назначили имена «порог», «сумма», «минимум» и т. д. Компилятор распознает их как регистры оперативной памяти. Типы переменных, как правило, назначаются в начале программы:

```
unsigned int gate1;      // объявление имени и типа переменной gate1
```

Кроме имени и типа переменных, как правило, назначаются начальные значения в начале программы. Это необязательный шаг, но так рекомендуется делать, и выглядит это следующим образом:

```
unsigned int gate1;      // объвление типа и имени переменной
signed int start, sum;   // Объявление типа и имени двух других переменных
gate1 = 20;              //Присваивание переменной gate1 начального
                         //значения
```

Процесс присваивания начального значения и объявления типа может быть выполнен за один шаг:

```
unsigned int gate1=20;  // Объявление типа, имени и значения переменной
```

Если нескольким переменным присваивается одинаковое начальное значение, процесс может быть упрощен:

```
unsigned int gate1=gate2=gate3=20;
signed int start=sm=0;
```

Тип переменной не положителен и не отрицателен по умолчанию. Например, char может быть записан вместо signed char (переменная signed byte). В этом случае компилятор считает переменную положительной. Если вы по какой-то причине забыли объявить тип переменной, компилятор автоматически считает ее положительной. Это означает, что такая переменная будет занимать два байта памяти и иметь значения в диапазоне от –32768 до +32767.

Константа

Константа (постоянная) представляет собой число или символ, имеющий фиксированное значение, которое не может быть изменено во время выполнения программы. В отличие от переменных константы хранятся во flash-памяти программ микроконтроллера в целях сохранения ценного пространства оперативной памяти. Компилятор распознает их по имени и префиксу.

Целые константы

Целые константы могут быть десятичными, шестнадцатеричными, восьмеричными или двоичными. Компилятор признает их формат на основе добавления префикса. Префиксы констант перечислены в табл. 1.7. Если число не имеет префикса, то считается десятичным по умолчанию. Тип константы автоматически назначается в зависимости от ее размера.

Таблица 1.7

Префиксы констант в языке microC

Формат	**Префикс**	**Пример**
Decimal	–	Const MAX=100
Hexadecimal	0x or 0X	Const MAX=0xFF
Octal	0	Const MAX=016
Binary	0b or 0B	Const MAX=0b11011101

В следующем примере постоянная MINIMUM автоматически будет рассматриваться как положительное число и сохраняться в двух байтах flash-памяти (16 бит):

```
const MINIMUM = -100;     // Объявление константы MINIMUM
```

Константы с плавающей точкой

Константы с плавающей точкой состоят из целой части, точки, дробной части и необязательной e или E, после которой следует целая часть экспоненты со знаком:

```
const T_MAX = 32.60;      // Объвление константы температуры T_MAX
const T_MAX = 3.260E1; // Объявление той же константы T_MAX
```

В обоих примерах константа под названием T_MAX объявлена и имеет дробное значение 32,60. Это позволяет программе сравнивать измеренную температуру в значимой постоянной вместо чисел, представляющих его. Символьные константы (ASCII символы) – это символы, заключенные в одинарные кавычки. В следующем примере константа объявлена как символ A, в то время как константа II класса объявлена как символ B:

```
const I_CLASS = 'A';      // Объявление константы I_CLASS
const II_CLASS = 'B';     // Объявление константы II_CLASS
```

Объявленные таким образом константы I_CLASS и II_CLASS при выполнении команды отправки на ЖК-дисплей вызовут символы A и B соответственно.

Строковые константы

Константа, состоящая из последовательности символов, называется строкой. Строковые константы заключаются в двойные кавычки:

```
const Message_1 = "Press the START button"; // Сообщение 1 для LCD
const Message_2 = "Press the RIGHT button"; // Сообщение 2 для LCD
const Message_3 = "Press the LEFT button";      // Сообщение 3 для LCD
```

В этом примере отправка константы Message_1 на ЖК-дисплей вызовет отображение сообщения "press the START button".

Перечисляемые константы

Перечисляемые константы представляют собой особый тип целых констант, которые делают программу более понятной и удобной, чтобы присваивать элементам порядковые номера. В следующем примере первому элементу в фигурных скобках автоматически присваивается значение 0, второму – значение 1, третьему – значение 2 и т. д.

```
enum MOTORS {UP, DOWN, LEFT, RIGHT}; // Объявл. конст. MOTORS
```

При каждом появлении слов "LEFT", "RIGHT", "UP" и "DOWN" в программе компилятор будет заменять их на соответствующие цифры (0–3). Конкретно, если выводы порта B 0, 1, 2 и 3 соединены с электродвигателем, который двигает что-то вверх, вниз, влево и вправо, команда для запуска двигателя «RIGHT» выставит бит в третий вывод порта B:

```
PORTB.RIGHT = 1;      // установка бита 3 PORTB
                      // связанного с мотором 'RIGHT'
```

Операторы, операции и выражения

Оператор – символ, обозначающий конкретные арифметические, логические или какие-либо другие операции. Есть более чем 40 доступных операций в языке C, но не более 10–15 из них используются на практике. Каждая операция выполняется на одном или более операндов, которые могут быть переменными или константами. Кроме того, каждая операция имеет приоритет выполнения и ассоциативность.

Арифметические операторы

Арифметические операторы (табл. 1.8) используются в арифметических операциях и всегда возвращают положительные результаты. В отличие от унарных операций, выполняющихся на одном операнде, бинарные операции выполняются на двух операндах. Другими словами, для выполнения бинарной операции необходимы два числа, например: A + B или A / B.

Таблица 1.8

Арифметические операторы языка microC

Оператор	**Операция**
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Взятие остатка от деления

Операторы присваивания

Есть два типа присваивания в языке C: простые операторы присваивания придают значения переменным, используя знак «=» . Например: a = 8.

Составные присваивания являются специфическими для языка C и состоят из двух символов, как показано в табл. 1.9. Некоторые арифметические выражения также могут быть записаны присваиванием из двух символов, это обеспечивает более эффективный машинный код.

Таблица 1.9

Операторы присваивания

Оператор	Пример	
	Выражение	Эквивалент
+=	a+=8	a=a + 8
-=	a-=8	a=a - 8
*=	a*=8	a=a * 8
/=	a/=8	a=a / 8
%=	a%=8	a=a % 8

Операторы инкремента и декремента

Операции инкремента и декремента обозначаются «++» и «- -». Эти символы могут либо предшествовать, либо следовать за переменной (табл. 1.10). В первом случае (++x) переменная будет сначала увеличиваться на 1, а затем использоваться в выражении. В противном случае (x++) переменная будет сначала использована в выражении, а затем увеличена на 1. То же самое относится к работе декремента.

Таблица 1.10

Операторы инкремента и декремента

Оператор	Пример	Описнаие
++	++a	Переменная «a» инкрементирована на 1
	a++	
--	--b	Переменная «b» декрементирована на 1
	b--	

Операторы сравнения

Операторы сравнения (табл. 1.11) используются для сравнения двух переменных, которые могут быть целыми числами (INT) или числами с плавающей точкой (float). Если выражение истинно, возвращается 1. В противном случае возвращается 0. Это используется в таких выражениях, как «если выражение истинно, то ...».

Таблица 1.11

Операторы сравнения

Оператор	Значение	Пример	Условие истинности
>	больше чем	b > a	если **b** больше, чем **a**
>=	больше или равно	a >=5	если **a** больше или равно **5**
<	меньше чем	a < b	если **a** меньше, чем **b**
<=	меньше или равно	a <= b	если **a** меньше или равно **b**
==	равно	a == 6	если **a** равно **6**
!=	не равно	a != b	если **a** не равно **b**

Логические операторы

Есть три типа логических операций в языке С: логическое И (AND), логическое ИЛИ (OR) и отрицание (NOT). Для наглядности логические состояния в приведенной ниже табл. 1.12 представлены в виде логического нуля (0 – False) и логической единицы (1 – истина). Логические операторы возвращают TRUE (логическая 1), если выражение имеет значение отличное от нуля, а ложь (логический 0), если выражение равно нулю. Это очень важно, потому что логические операции, как правило, используется в выражениях, а не на отдельных переменных в программе. Таким образом, логические операции обозначат правдивость всего выражения.

Например, 1 && 0 означает то же самое, что и (true expression) && (false expression). Результат 0, т. е. – False в обоих случаях.

Таблица 1.12

Логические операторы

Оператор	**Логическое И (AND)**		
&&	Операнд 1	Операнд 2	Результат
	0	0	0
	0	1	0
	1	0	0
	1	1	1
Оператор	**Логическое ИЛИ (OR)**		
||	Операнд 1	Операнд 2	Результат
	0	0	0
	0	1	1
	1	0	1
	1	1	1
Оператор	**Логическое НЕ (NOT)**		
!	Операнд 1		Результат
	0		1
	1		0

Битовые операторы

В отличие от логических операций, которые выполняются на переменных, битовые операции выполняются на отдельных битах при помощи операндов. Битовые операторы используются для изменения битов переменной. Они перечислены в табл. 1.13.

Таблица 1.13

Битовые операторы

Операнд	**Значение**	**Пример**	**Результат**	
~	Побитное отрицание	a = ~b	b = 5	a = -5
<<	Сдвиг влево	a = b<<2	b = 11110011	a = 11001100
>>	Сдвиг вправо	a = b>>2	b = 11110011	a = 00011110

Окончание табл. 1.13

Операнд	Значение	Пример	Результат	
&	Побитное И (AND)	c = a&b	a = 11100011 b = 11001100	c = 11000000
|	Побитное ИЛИ (OR)	c = a|b	a = 11100011 b = 11001100	c = 11101111
^	Побитное исключающее ИЛИ (XOR)	c = a^b	a = 11100011 b = 11001100	c = 00101111

Использование операторов

За исключением операторов присваивания два оператора не должны быть записаны рядом друг с другом (выражение «x*%12;» вызовет ошибку).

Операторы группируются при помощи круглых скобок, подобно арифметическим выражениям. Выражения, заключенные в скобки, выполняются в первую очередь. Если необходимо, могут быть использованы несколько скобок.

Каждый оператор имеет свой приоритет и ассоциативность, как показано в табл. 1.14.

Таблица 1.14

Приоритет и ассоциативность операторов

Приоритет	Операторы	Ассоциативность
Высокий	( ) [ ] => .	Слева направо
	! ~ + + - - + (унарный) - (унарный) * (указатель) & (указатель)	Справа налево
	* / %	Слева направо
	+ -	Слева направо
	< >	Слева направо
	< <= > >=	Слева направо
	== !=	Слева направо
	&	Слева направо
	^	Слева направо
	|	Слева направо
	&&	Слева направо
	| |	Справа налево
	? :	Справа налево
Низкий	= + = - = * = / = / = & = ^ = | = <= >=	Слева направо

Типы преобразования данных

Основные типы данных располагаются в иерархическом порядке (рис. 1.33).

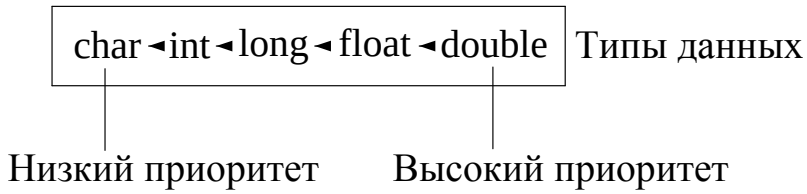


Рис. 1.33. Иерархический порядок типов данных

Если два операнда различного типа используются в арифметической операции, то более низкий приоритет типа операнда автоматически преобразуется в более высокий приоритет типа операнда. В выражениях, свободных от операции присваивания, в результате получают следующее.

Если операнды с наивысшим приоритетом имеют тип double, то типы всех других операндов в выражении автоматически преобразуются также в тип double.

Если наивысший приоритет операндов имеет тип long, то в результате типы всех других операндов в выражении автоматически преобразуются в тип long.

Если операнды имеют тип long или char, то типы всех других операндов в выражении автоматически преобразуются в тип Int. Автопреобразование осуществляется также в операциях присваивания. Результат выражения справа от оператора присваивания всегда преобразуется в тип переменной слева от оператора. Если результат оказывается более высокого типа, он урезается или округляется до соответствия типу переменной. При преобразовании реальных данных в целое число следующие после запятой числа всегда отбрасываются.

```
int x;                  // Переменная x объявлена как целое int
x = 3;                  // Переменной x присвоено значение 3
x += 3.14;              // Число PI (3.14) добавлено к переменной x путем
                        // выполнения операции присваивания.
/* Результат сложения равен 6 вместо ожидаемого 6.14. Для получения
ожидаемого результата без отбрасывания значащих цифр после
запятой нужно выполнить обычное сложение (x+3.14), . */
```

Условные операторы

Условие является общим компонентом программы. При встрече с ним, необходимо выполнить одну из нескольких операций. Условные операнды *if-else* и *switch* используются в условных операциях.

Условный оператор if-else

Условный оператор может появиться в двух формах – как операторы if и if-else («если»–«иначе»). Ниже приведен пример оператора if:

```
if(expression) operation;
```

Если результат выражения, заключенного в скобках, не равен 0 (true), то выполняется указанная операция. Если результат выражения – 0 (false), операция не выполняется и программа сразу приступит к выполнению последующей инструкции.

Как уже упоминалось, другая форма сочетает в себе операторы как *if*, так и *else*:

```
if(expression) operation1 else operation2;
```

Если результат выражения не равен 0 (true), operation1 выполняется, в противном случае выполняется operation2. После выполнения операций одной либо другой ветви программа продолжает выполняться.

Альтернативный синтаксис if-else:

```
if(expression)
operation1
else
operation2
```

Если operation1 и/или operation2 состоят из нескольких отдельных операций, то вся группа операций 1 и/или 2 должна быть заключена в фигурные скобки. Например:

```
if(expression) {
... //
... // operation1
...} //
else
operation2
```

Оператор if-else можно записать с помощью условного оператора «?:», как в примере ниже:

```
(expression1)? expression2 : expression3
```

Если expression1 не равно 0 (true), то результат всего выражения будет равен результату, полученному из expression2. В противном случае, если expression1 – 0 (false), то результат всего выражения будет равен результату, полученному из expression3.

```
maximum = (a > b)? a : b  // Переменной maximum присваивается значение
                          // большей из переменных (a или b)
```

Условный оператор switch

В отличие от if-else, которое делает выбор между двумя вариантами в программе, оператор switch («переключатель») позволяет выбирать между несколькими операциями.

Синтаксис switch:

```
switch (selector)          // Selector является переменной типа char или int
{
case constant1:
operation1                 // Группа операторов, исполняемая в случае
...                        // равенства selector и constant1
break;
case constant2:
operation2                 // Группа операторов, исполняемая в случае, если
...                        // равенства selector и constant2
break;
...
default:
expected_operation         // Группа операторов, исполняемая в случае, если
...                        // ни одна из констант не равна selector
break;
}
```

Работа switch выглядит следующим образом: сначала выполняет сравнение переменной (или вычисляемого выражения) selector с постоянной constant1. Если они совпадают (равны), инструкции в этом блоке выполняются до ключевого слова break или до конца операции 1. Если совпадения не было, selector сравнивается с constant2 и, если совпадение найдено, инструкции в этом блоке выполняются до ключевого слова break или до конца операции 2 и т. д. Если значение переменной selector не соответствует ни одной из констант, ни одна из инструкций не будет выполнена. Однако, если в конце sitch указать ключевое слово «default:», то записанные после него инструкции будут исполнены, если не было совпадений с указанными выше константами. Кроме того, можно сравнить выражение selector с группой констант. Если оно соответствует любой из них, соответствующие операции будут выполнены:

```
switch (number)            // number представляет один из дней недели
                           // необходимо определить, является он рабочим
{                          // или выходным
   case1: case2: case3: case4: case5: LCD_message = 'Weekday'; break;
   case6: case7: LCD_message = 'Weekend'; break;
   default:
   LCD_message_1 = 'Choose one day in a week'; break;
}
```

Программный цикл

Часто бывает необходимо повторить определенную операцию несколько раз в программе. Набор команд, который повторяется, называется программным циклом. Сколько раз он будет выполнен, т. е. как долго программа будет оставаться в работе, зависит от условий, записанных в скобках после ключевого слова.

Цикл *WHILE* выглядит следующим образом:

```
while(expression){
commands
...
}
```

Команды выполняются неоднократно (программа остается в цикле), пока выражение не станет ложным. Если выражение ложно при входе в цикл, то цикл не будет выполнен и программа пропустит инструкции в этом цикле. Особым типом программ является бесконечный цикл. Он образуется, если условие остается неизменным в течение цикла. В этом случае результат в скобках всегда верен (1 = 1), это означает, что программа остается в том же цикле:

```
while(1){
...                    // Выражения внутри фигурных скобок
...                    // будут выполняться бесконечно (бесконечный цикл).
}
```

Цикл *FOR LOOP* выглядит следующим образом:

```
for(initial_expression; condition_expression; change_expression) {
operations
...
}
```

Выполнение такой последовательности программы похоже на the while loop, за исключением того, что в этом случае процесс установки начального значения (инициализации) осуществляется в объявлении initial_expression. Исходное выражение задает начальную переменную, которая в будущем сравнивается с condition_expressions перед входом в цикл. Операции внутри цикла выполняются многократно, и после каждой итерации значение выражения изменяется согласно change_expression. Итерация продолжается до тех пор, пока condition_expression не станет ложным.

```
for(k=1; k<5; k++)      // Увеличить переменную k 5 раз (от 1 до 5)
operation               // и повторить операцию каждый раз
...
```

Операция должна быть выполнена пять раз. После этого будет подтверждено путем проверки, что выражение k < 5 неверно (после 5 итераций k = 5) и программа выходит из цикла.

Цикл *Do-while* выглядит следующим образом:

```
do
operation
while (check_condition);
```

В этом случае операция выполняется как минимум один раз независимо от того, является ли условие condition истинным или ложным, как условие check_expressions выполняется в конце цикла. Если результат не равен 0 (true), то процедура повторяется. В следующем примере программа остается в do-while loop, пока переменная не достигнет 1E06 (миллион итераций).

```
a = 0;                  // установка начального значения
do
a = a+1                 // выполнение операции
while (a <= 1E06);      // проверка условия
```

Написание кода на ассемблере

Иногда процесс написания программы на языке С требует части кода, который будет написан на языке ассемблер. Это полезно в сложных частях программы, которые должны выполняться определенным способом или за точный период времени. Например, когда необходимо иметь очень короткие импульсы (несколько микросекунд), появляющиеся периодически на входах (рис. 1.34).

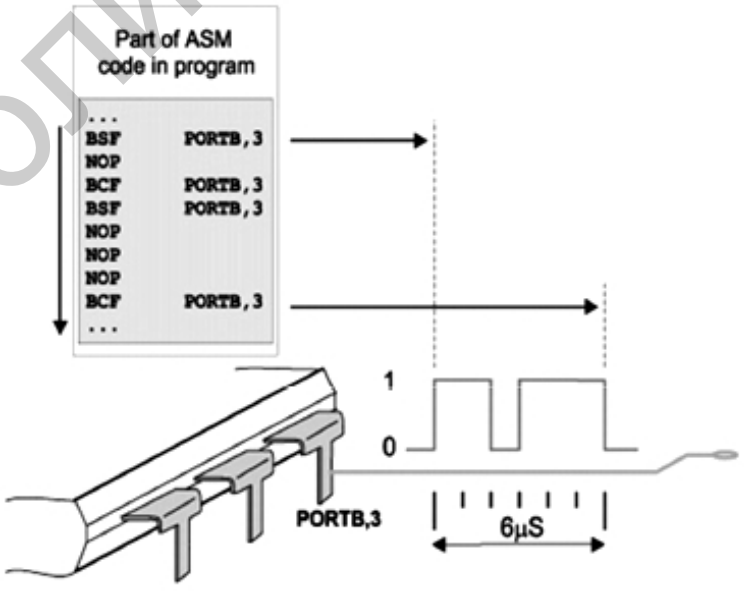


Рис. 1.34. Иерархический порядок типов данных

В таких и подобных случаях простое решение заключается в использовании ассемблера для части управляющей программы длительности импульса. Одна или несколько инструкций ассемблера вставляются в программу, написанную на языке С, используя команду asm:

```
asm
{
Инструкции на ассемблере
...
}
```

В кодах, написанных на ассемблере, можно использовать константы и переменные, определенные ранее в языке С. Конечно, если вся программа написана на языке С, его правила применяются при объявлении этих констант и переменных.

```
unsigned char maximum = 100;   // Declare variables: maximum = 100
asm
{                                   // Start of assembly code
MOVF maximum,W // W = maximum = 100
...
}                                   // End of assembly code
```

Массивы

Группа переменных одного и того же типа называется массивом. Элементы массива называются компонентами, а их тип называется основным видом. Объявление массива, указание его имени, типа и количества элементов будет включать

```
component_type array_name [number_of_components];
```

Массив можно рассматривать как более короткий или более длинный список переменных такого же типа, где каждому из них присваивается порядковый номер (нумерация всегда начинается с нуля). Такой массив часто называют вектор. В табл. 1.15 показан массив с именем shelf, который состоит из 100 элементов.

В этом случае содержимое переменной (элемента массива) представляет ряд продуктов, которые входят в блок. Элементы имеют доступ по индексации, т. е. указывается их порядковый номер (индекс):

```
shelf[4] = 12;          // 12 вещей «положили» на полку shelf №4
temp = shelf [1];       // Значение переменной shelf[1] скопировано
                        // в переменную temp
```

Таблица 1.15

Пример массива элементов

Массив «shelf»	**Элементы массива**	**Содержимое элементов**
7	shelf[0]	7
23	shelf[1]	23
34	shelf[2]	34
0	shelf[3]	0
0	shelf[4]	0
12	shelf[5]	12
9	shelf[6]	9
…	…	…
23	shelf[99]	23

Элементам может быть назначено содержимое в объявлении массива. В следующем примере объявлен массив с именем календарь и каждому элементу присваивается определенное количество дней:

unsigned char calendar [12] = {31,28,31,30,31,30,31,31,30,31,30,31};

Двумерный массив

Помимо одномерных массивов в языке С есть и многомерные массивы. Далее описаны только двумерные массивы, называемые матрицами, которые можно рассматривать в виде таблиц. Двумерный массив объявляется с указанием размера каждой размерности. Рассмотрим пример:

component_type array_name [number_of_rows] [number_of_columns];

number_of_rows and number_of_columns представляют количество строк и столбцов соответственно

int Table [3][4]; *// Таблица с 3 «рядами» и 4 «колонками»*

Этот массив может быть представлен в форме табл. 1.16.

Таблица 1.16

Пример двумерного массива

table[0][0]	table[0][1]	table[0][2]	table[0][3]
table[1][0]	table[1][1]	table[1][2]	table[1][3]
table[2][0]	table[2][1]	table[2][2]	table[2][3]

Как и векторам, элементам матрицы могут быть присвоены значения при объявлении массива. В следующем примере элементам двумерного массива «table» присваиваются значения. Как видно, этот массив имеет два ряда и три колонки:

int Table[2][3] = {{3,42,1}, {7,7,19}};

Функции

Каждая программа, написанная на языке С, состоит из большего или меньшего числа функций. Основная идея заключается в разделении программы на несколько частей с помощью этих функций с целью более легкого решения проблемы. Кроме того, функции позволяют нам использовать навыки и знания других программистов. Например, чтобы послать строку на ЖК-дисплей, гораздо проще использовать уже написанную часть программы, чем начинать все сначала. Функции состоят из команд, определяющих, что должно быть сделано с переменными. Их можно сравнить с подпрограммами.

Как правило, гораздо проще иметь программу, состоящую из большого числа простых функций, чем нескольких сложных функций. Тело функции обычно состоит из нескольких команд на выполнение в порядке их описания. Каждая функция должна быть объявлена таким образом, чтобы быть правильно интерпретированной в процессе компиляции. Декларация содержит следующие элементы:

- название функции;
- тело функции;
- список параметров;
- объявление параметров;
- тип результата функции.

Функция выглядит следующим образом:

```
type_of_result function_name (type argument1, type argument2,...)
{
Command;
Command;
...
}
```

Пример:

```
/* Функция, вычисляющая результат деления делимого на делитель.
Функция возвращает структуру типа div_t. */
div_t div(int number, int denom);
```

Следует отметить, что функциям необязательно иметь параметры, но они должны иметь скобки, которые будут использоваться для их ввода. В противном случае компилятор неправильно истолкует эту функцию. Если функция после выполнения не возвращает результат в основную программу или вызывающую ее функцию, внешняя программа продолжает выполнятся сразу после закрывающей функцию фигурной скобки. Такие функции используются, когда необходимо изменить состояние выходных контактов микроконтроллера, при передаче данных через последовательный интерфейс,

при записи данных на ЖК-дисплее и т. д. Компилятор распознает эти функции по типу их результата, заданного недействительным.

```
void function_name (type argument1, type argument2,...)
{
Commands;
}
```

Пример:

```
void interrupt() {
cnt++ ;                 // Прерывание вызывает увеличение на 1 переменной cnt
PIR1.TMR1IF = 0;// Сброс бита TMR1IF
}
```

Этой функции может быть присвоено произвольное имя. Единственным исключением является главное имя, которое имеет специальное назначение. А именно, программа всегда начинает выполняться с помощью этой функции. Это означает, что каждая программа, написанная на языке С, должна содержать одну функцию с именем «main», которое не должно быть помещено в начало программы. Если необходимо, чтобы вызываемая функция возвращала результат после выполнения, то используется команда возврата «return»:

```
type_of_result function_name (type argument1, type argument2,...)
{
Commands;
...
return expression;
}
```

Если функция содержит команду return без последующего выражения, функция останавливает свое выполнение, когда сталкивается с этой командой, и программа приступает к выполнению первой команды, следующей за фигурной скобкой.

Объявление новой функции

Помимо функций язык С автоматически распознает и совершенно новые функции, которые часто используются в программах. Каждая нестандартная функция должна быть объявлена в начале программы. Объявление функции называется прототипом и выглядит следующим образом:

```
type_of_result function_name (formal parameters)
{
description of formal parameters
definition and declaration
operators
...
}
```

Тип функций, которые не возвращают значение, называется void. Если тип результата не объявляется в программе, это считается тип INT (целое число со знаком). Параметры, написанные в прототипе функции, определяют, что должно быть сделано с реальными параметрами.

Параметры функции прототипа – формальные параметры. Следующий пример объявляет функцию, которая вычисляет объем цилиндра.

Пример:

```
const double PI = 3.14159;          // Объявление константы PI
float volume (float r, float h)     // объявление функции volume типа float
{                                   // с формальными (входными) параметрами r и h
float v;                            // объявление переменной v для результата типа float
v = PI*r*r*h;                       // вычисление значения функции (объем цилиндра)
return v;                           // возврат результирующего значения функции
}
```

Если такой расчет должен быть выполнен позже в программе (на практике это может быть объем бака), то достаточно определить реальные параметры и вызвать функцию. В процессе составления компилятор заменяет формальные параметры на реальные, как показано ниже:

```
float radius=5, height=10, tank;    // объявление реальных переменных
...                                 // параметров радиуса, высоты и объема цилиндра
tank = volume (radius,height);      // вычисление объема цилиндра
...                                 // путем вызова функции объема цилиндра
```

Библиотеки функций

Имена всех функций, используемых в языке С, хранятся в файле, который называется header. Эти функции в зависимости от их назначения помещаются в файлы меньшего размера и называются библиотеками. Перед использованием любого из них в программе необходимо указать соответствующий файл заголовка с помощью команды #include в начале программы. Если компилятор встречает неизвестную функцию во время выполнения программы, он сначала ищет его объявление в установленных библиотеках.

Стандартные библиотеки языка ANSI C

Функции языка С не были стандартизированы вначале и производители программного обеспечения изменили их в соответствии с их потребностями. Но язык С стал очень популярным в настоящее время и возникла необходимость ввода стандарта. Установленный стандарт называется ANSI C и содержит 24 библиотеки с функциями. Этими библиотеками, как правило, снабжен каждый компилятор С:

<assert.h>, <complex.h>, <ctype.h>, <errno.h>, <fenv.h>, <float.h>, <inttypes.h>, <iso646.h>, <limits.h>, <locale.h>, <math.h>, <setjmp.h>, <signal.h>, <stdarg.h>, <stdbool.h>, <stdint.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <tgmath.h>, <time.h>, <wchar.h>, <wctype.h>.

Вышеприведенные краткие теоретические сведения о программировании на языке С являются необходимыми, но для того чтобы вести разработку программ, исполняемых на микроконтроллерах, знания языка С не достаточно. К нему необходимо добавить знание архитектуры и особенностей микроконтроллера, компилятора и всего, что можно получить только из документов-даташитов (от англ. datasheet) и собственной практики. В начале обучения или освоения нового микроконтроллера могут возникать трудности с точными названиями регистров, их адресами, названиями конкретных битов управления, команд и др.

Подробные сведения по языку MicroC приведены в [1, 4].

1.2.5. Основы работы с портами ввода-вывода микроконтроллера

Основные принципы работы с портами ввода-вывода на PIC-микроконтроллерах в среде microC можно проиллюстрировать на основе следующего примера кода:

```
// Настройка портов
// Перевод в цифровой (логический) режим работы
ADCON = 0x00;          // Отключение АЦП
ANSEL = 0;             // Настройка пинов портов как
ANSELH = 0;            // цифровых входов/выходов
C1ON_bit = 0;          // Отключение компараторов
C2ON_bit = 0;          // (устройств сравнения)

// A – вход, B – выход, C – вход/выход
TRISA = 0xFF;
TRISB = 0x00;
TRISC = 0x0F;

// Чтение и запись портов
PORTA = 0x00;          // Отключить подтягивающие резисторы
PORTB = 0x00;          // Обнулить выходной порт B
PORTB = PORTA;         // Перезаписать значение порта A на порт B
```

```
// Присвоить значения младших 4 бит порта C
// (вход) старшим 4 битам порта C (выход)
PORTC = PORTC<<4;

// Прочитать значение первого (младшего) бита
char a = 0;                    // Буфер для чтения
char b = 0;                    // Номер читаемого бита
a = PORTC& 0x01                // Жесткое чтение
a = PORTC& 1<<b                // Гибкое чтение
a = PORTC.F0                   // Только в компиляторе mikroC

// Присвоить значение пятому биту
char b = 4;                    // Номер записываемого бита
PORTC&= 0xEF;                  // Жестко установить 0
PORTC&= ~(1<<b);               // Гибко установить 0
PORTC.F4 = 0;                  // Только в компиляторе mikroC
PORTC |= 0x10;                 // Жестко установить 1
PORTC |= 1<<b;                 // Гибко установить 1
PORTC.F4 = 1;                  // Только в компиляторе mikroC

// Установить неизвестный бит
char a;                        // Записываемый бит (младший)
char b = 4;                    // Номер записываемого бита
PORTC = (a&1) ? PORTC|(1<<b) : PORTC&~(1<<b);  // Гибко
PORTC.F4 = a;                  // Только в компиляторе mikroC
```

Устранение дребезга контактов

Дребезгом контактов называется кратковременное замыкание или размыкание электрической цепи, что часто встречается при использовании простых кнопок или перемычек. Это явление приводит к ложным срабатываниям и множественным нажатиям. Распространен программный способ борьбы с дребезгом – многократная проверка состояния интересующего контакта. Например, для определения отдельной нажатой кнопки, выдающей уровень логической единицы, можно использовать следующую функцию:

```
char key_press(void) {                  // Функция устранения дребезга
   char i = j = 0x00;                   // Переменные-счетчики
   for(i=3; i>0; i--) {                 // Цикл на три повтора
   if (PORTD & (1<<(7))) j++;           // Проверка контакта RD7
   delay_ms(1); }                       // Пауза в одну миллисекунду
   if (j==3) return 0x01;               // нажата – 1
else     return 0x00; }                 // иначе – 0
```

1.2.6. Ипользование программного инструментария

Используемый для работы набор программ включает в себя:

– mikroC PRO for PIC (написание и компиляция кода, отладка);
– mikroProg Suite For PIC (прошивка микроконтроллера);
– Saleae LLC (анализ логических сигналов и протоколов).

Для запуска можно использовать соответствующие ярлыки (рис. 1.35) или пункты в меню «Пуск».

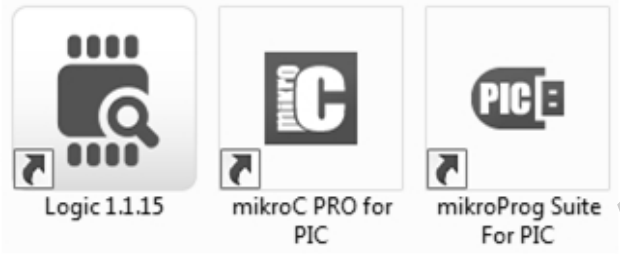


Рис. 1.35. Ярлыки используемых программ

Создание проекта

Подробное описание работы со средой «mikroC PRO for PIC» дано в [5, 6]. Работа над новой программой начинается с создания нового проекта. Мастер создания проектов вызывается из меню «File» или комбинацией клавиш («хоткей») Shift+Ctrl+N (рис. 1.36). Процесс создания нового проекта при помощи мастера включает четыре шага.

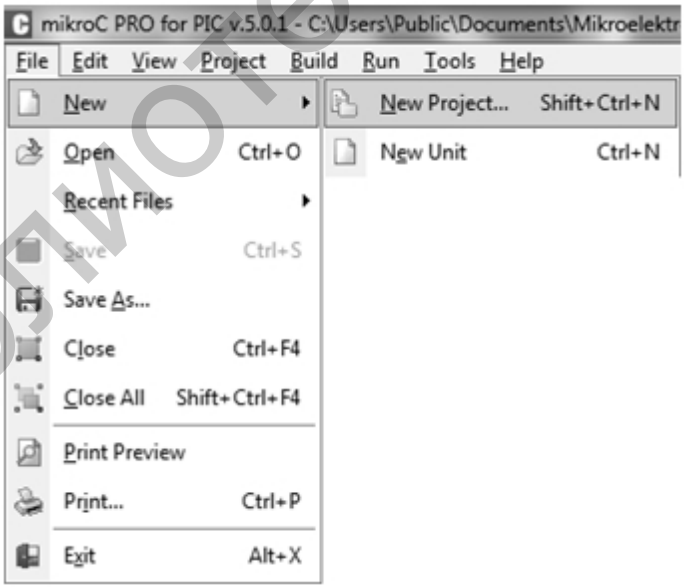


Рис. 1.36. Создания нового проекта из меню «File»

Мастер создания проектов имеет стандартную для такого типа программ организацию из нескольких последовательно сменяющихся «страниц» с предлагаемыми для выбора настройками.

На первом шаге (рис. 1.37) задается имя проекта (Project Name), рабочая директория (Project folder), вид микроконтроллера (Device Name, выбираем PIC16F887) и частота внешнего тактирования (Device Clock, 8 МГц).

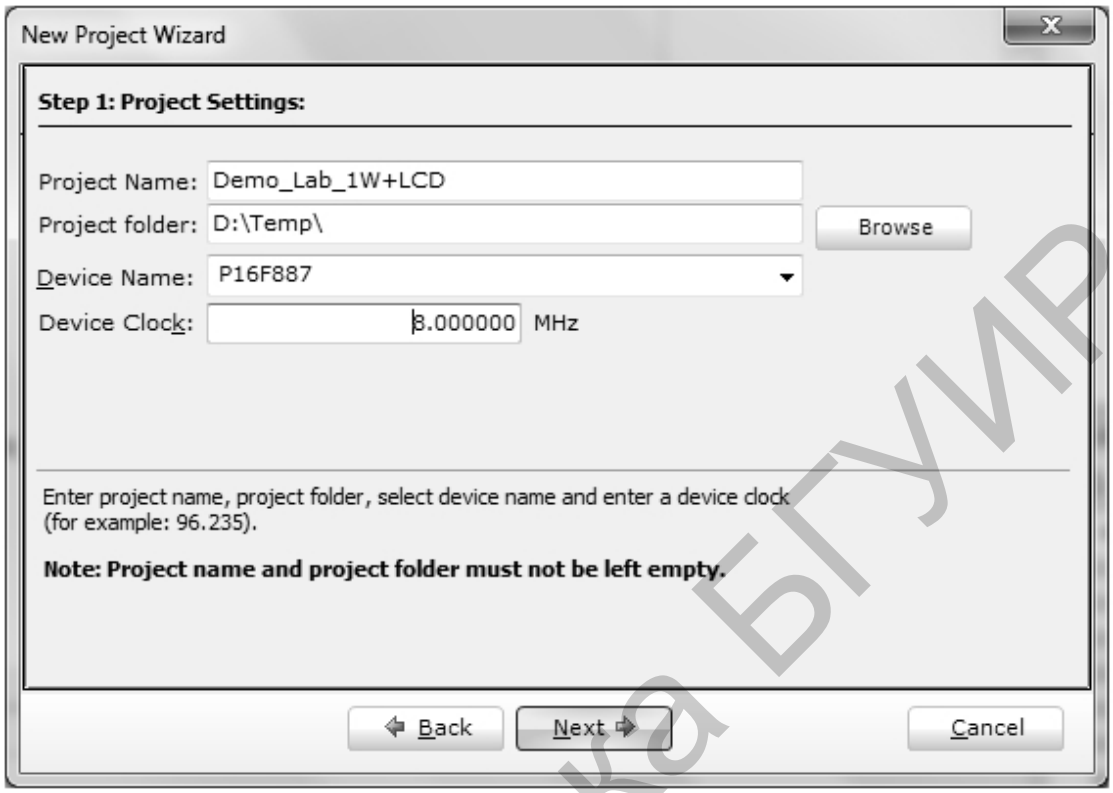


Рис. 1.37. Настройки проекта «Project Settings»

На втором шаге можно подключить к проекту дополнительные файлы, но для дальнейшей работы с подключенными файлами потребуется ознакомление с инструментом управления файлами проекта, что излишне перегружает работу над простыми проектами. Файлы к проекту не подключаем.

На третьем шаге (рис. 1.38) к проекту можно подключить встроенные библиотеки среды «mikroC PRO for PIC». По умолчанию библиотеки не подключаем (опция «Include None»), при необходимости их можно подключить позднее. Основным недостатком встроенного набора библиотек является их «закрытость» (другой термин инкапусулированность – когда происходит сокрытие реализации при открытом интерфейсе; в качестве примера можно рассматривать функцию с параметрами, у которой внутренняя реализация неизвестна, но известен принцип ее работы и результат выполнения).

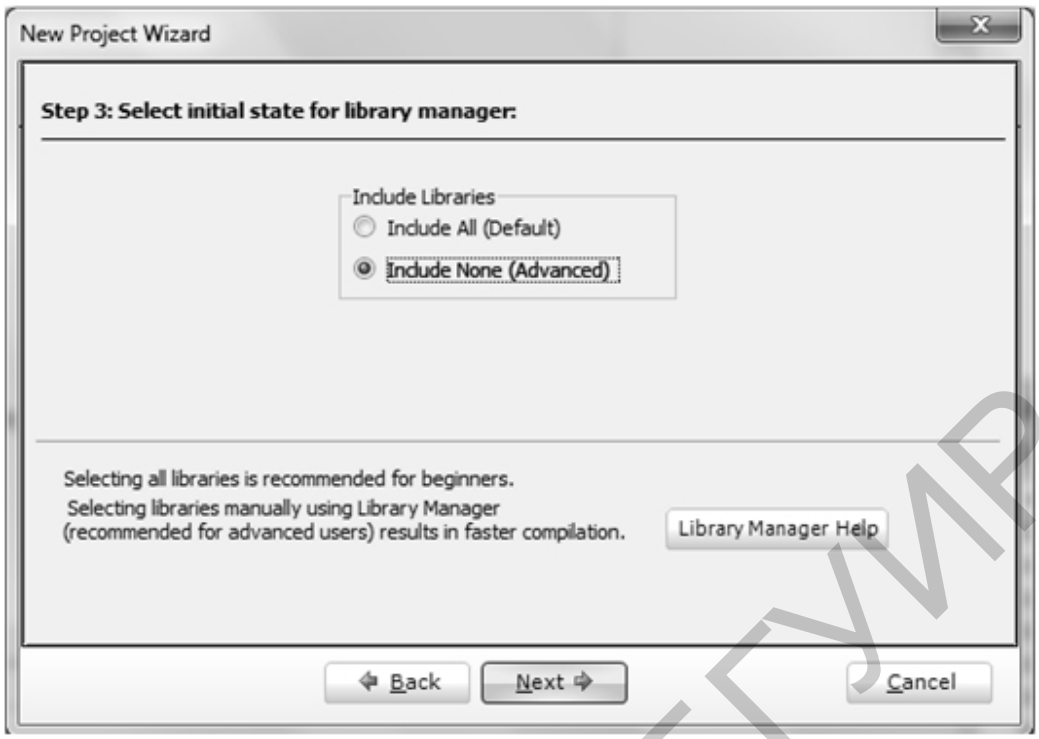


Рис. 1.38. Окно подключения библиотек

На последнем четвертом шаге (рис. 1.39) можно после создания проекта вызвать форму установки конфигурационных битов микроконтроллера, но для простых проектов это не понадобится. Все необходимые настройки можно будет выбрать позднее в приложении для прошивки mikroProg Suite For PIC.

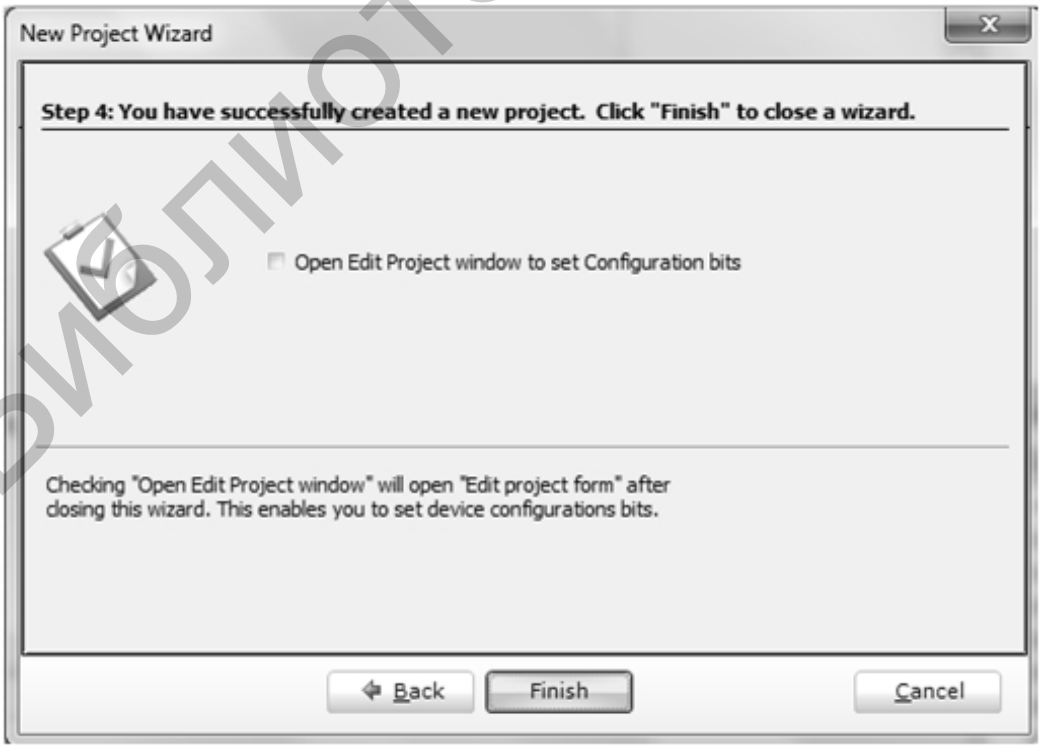


Рис. 1.39. Окно успешного завершения создания проекта

Выбранный при создании проекта вид микроконтроллера и частоту тактирования можно просмотреть и изменить, воспользовавшись специальной закладкой «Project Settings» (рис. 1.40) в левом верхнем углу рабочего окна среды mikroC PRO for PIC. Эти параметры важны на стадии компиляции, код для другого вида микроконтроллера не будет работать.

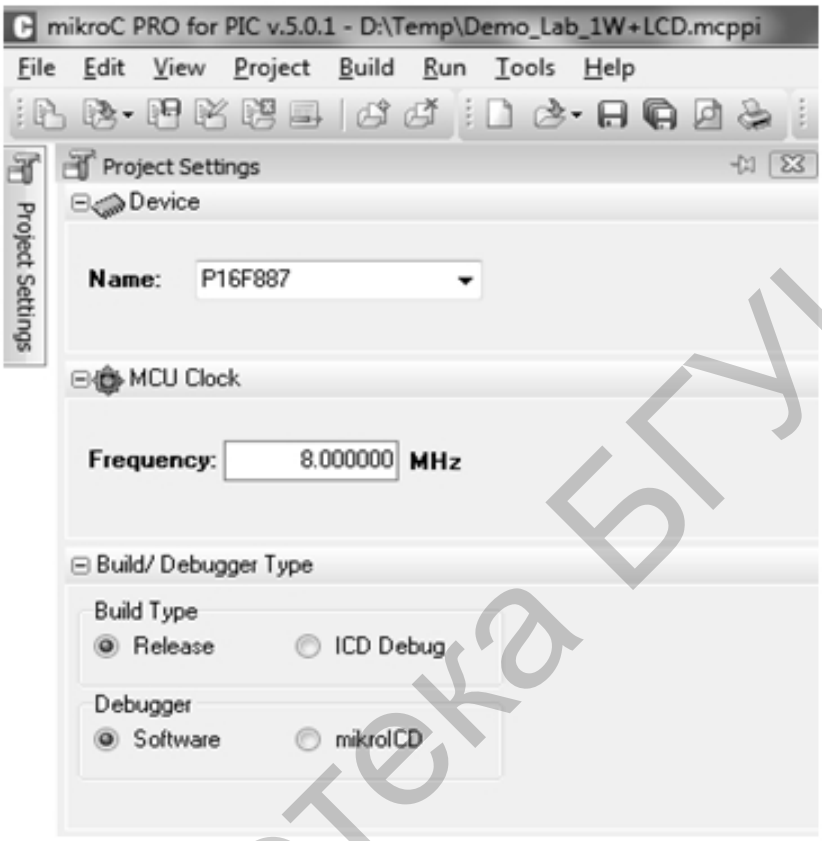
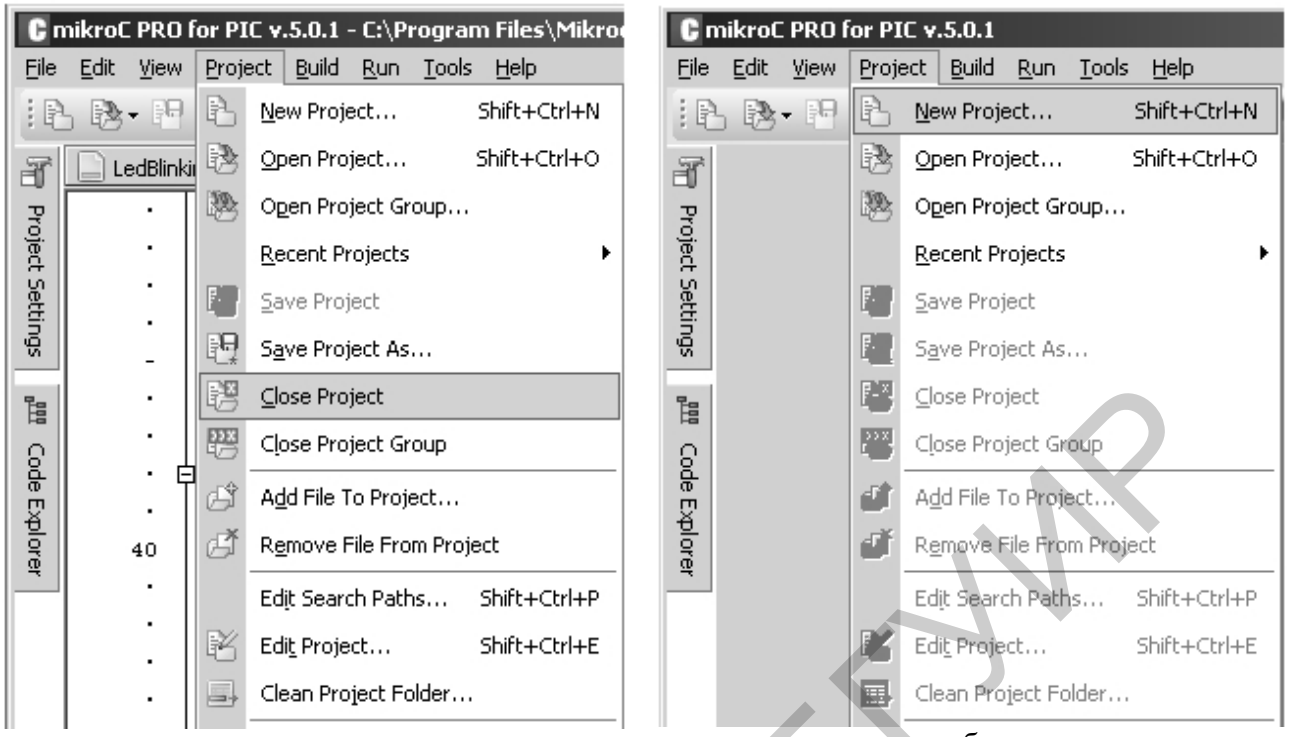


Рис. 1.40. Конфигурация выбранного микроконтроллера PIC P16F887

Иногда при открытии среды может быть автоматически открыт предыдущий проект. В таком случае его следует закрыть (рис. 1.41, а), а затем создать новый (рис. 1.41, б). Если забыть закрыть старый проект и вместо этого создать новый файл для кода своей программы, то без дополнительного управления иерархией подключенных к проекту файлов не удастся откомпилировать и прошить свою программу, так как будет компилироваться и прошиваться главная программа старого проекта.

Если возникла необходимость изменить настройки конфигурационных битов микроконтроллера после создания проекта, следует воспользоваться опцией «Edit Project» (рис. 1.42), что аналогично установке галочки «Open Edit Project window» в процессе создания проекта (см. рис. 1.39). Например, таким образом можно избавиться от необходимости каждый раз задавать параметры тактирования от внешнего осциллятора (опция «HS») в программе mikroProg Suite for PIC.



а б

Рис. 1.41. Закрытие старого и создание нового проекта

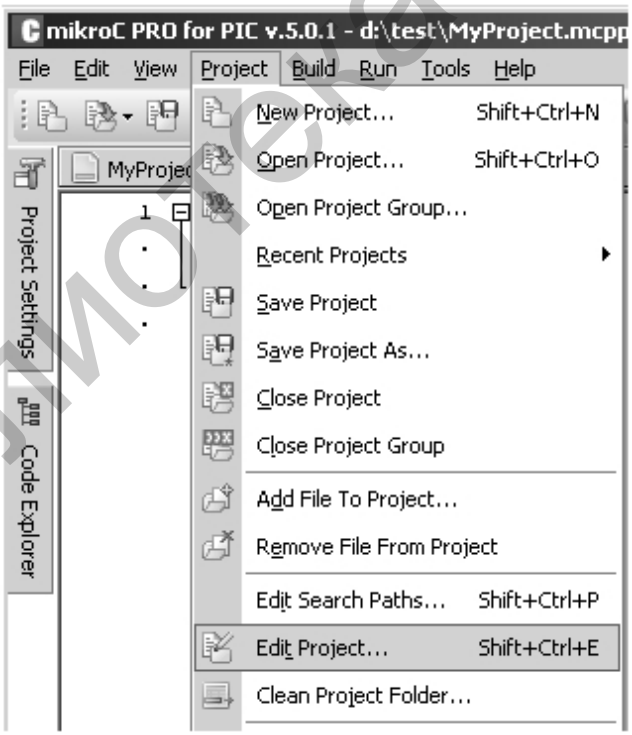


Рис. 1.42. Вызов окна редактирования конфигурационных битов микроконтроллера

В открывшемся окне (рис. 1.43) следует проверить правильность выбора модели микроконтроллера («P16F887»), источника и частоты тактирования («HS» и «8.000000» соответственно).

Рис. 1.43. Настройка конфигурационных битов микроконтроллера PIC P16F887

Добавление библиотек

После создания проекта иногда требуется вручную скопировать файлы необходимых библиотек в директорию с проектом. Подключить библиотеки из директории с проектом можно при помощи директивы препроцессора *#include* с указанием имени файла библиотеки в двойных кавычках (рис. 1.44). Подключать файлы с настройками библиотеки следует до самой библиотеки (например, «GLCD_Screen_Settings» перед «GLCD_Screen»).

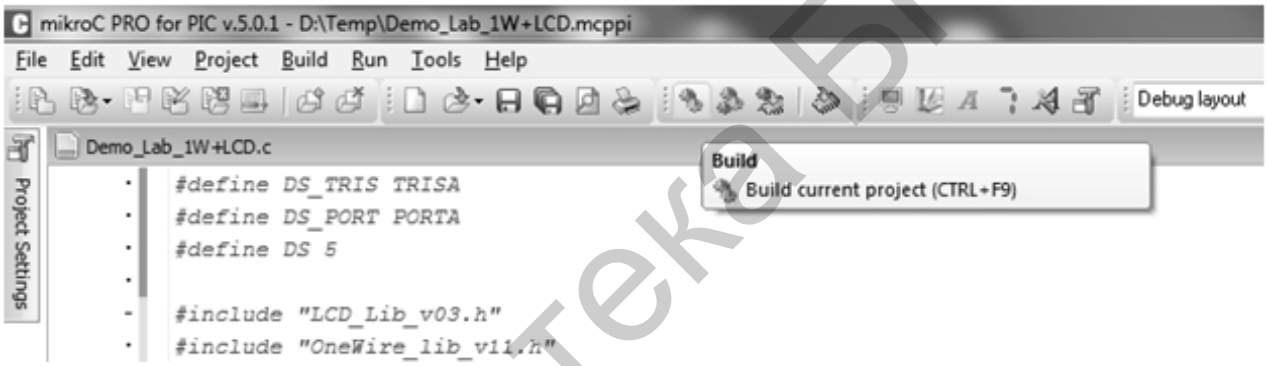


Рис. 1.44. Пример подключения библиотек через директиву include

После написания кода программы для проверки и компиляции проекта можно воспользоваться кнопкой ***Build*** на панели инструментов или Ctrl+F9 (рис. 1.44). Успешная компиляция подтверждается соответствующими сообщениями внизу рабочего окна (рис. 1.45).

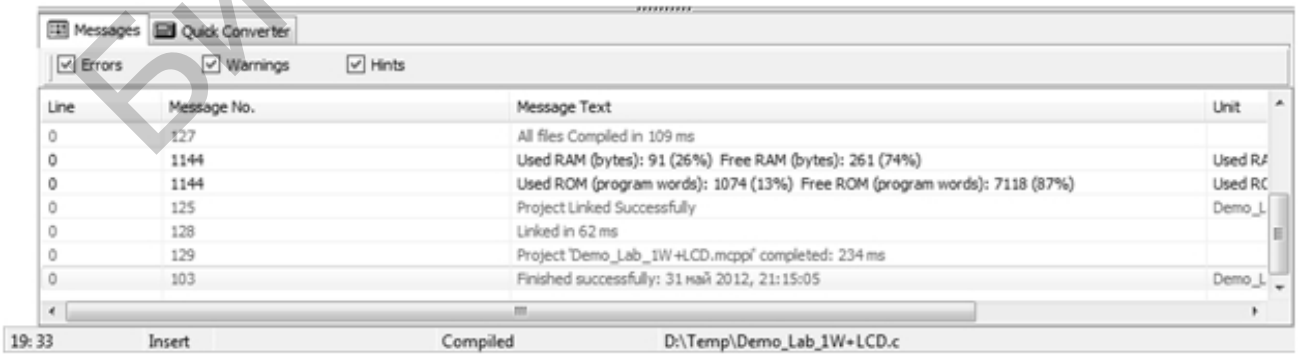


Рис. 1.45. Пример успешной компиляции проекта

Прошивка микроконтроллера

Прошивка микроконтроллера при помощи приложения ***«mikroProg Suite For PIC»*** осуществляется в пять шагов и начинается с включения питания отладочной платы. Подключение платы иллюстрируется специальным значком внизу рабочего окна программы (рис. 1.46). При этом возможен сброс некоторых настроек приложения.

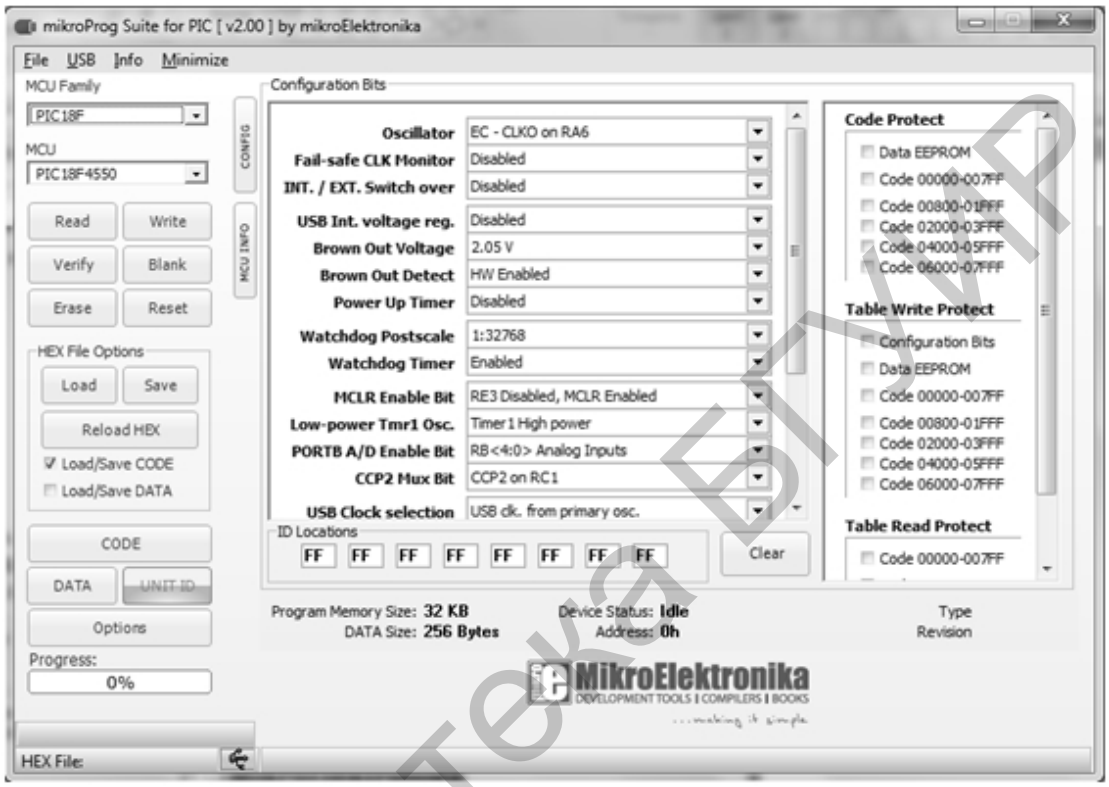


Рис. 1.46. Пример окна «mikroProg SuiteFor PIC» с подключенным макетом

На втором шаге следует выбрать семейство (рис. 1.47) и модель (рис. 1.48) прошиваемого микроконтроллера при помощи «выпадающих строк» на панели в левой части рабочего окна программы.

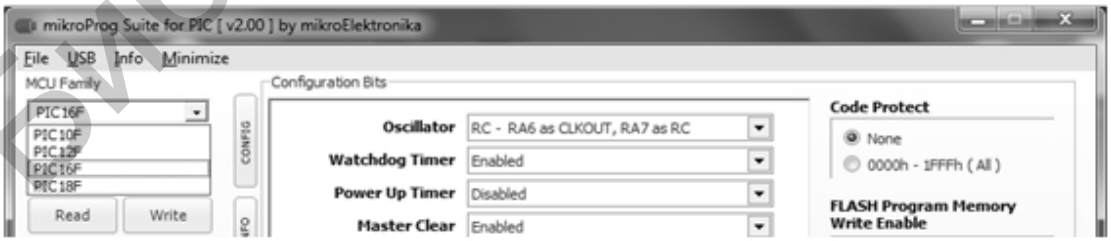


Рис. 1.47. Пример выбора семейства микроконтроллеров

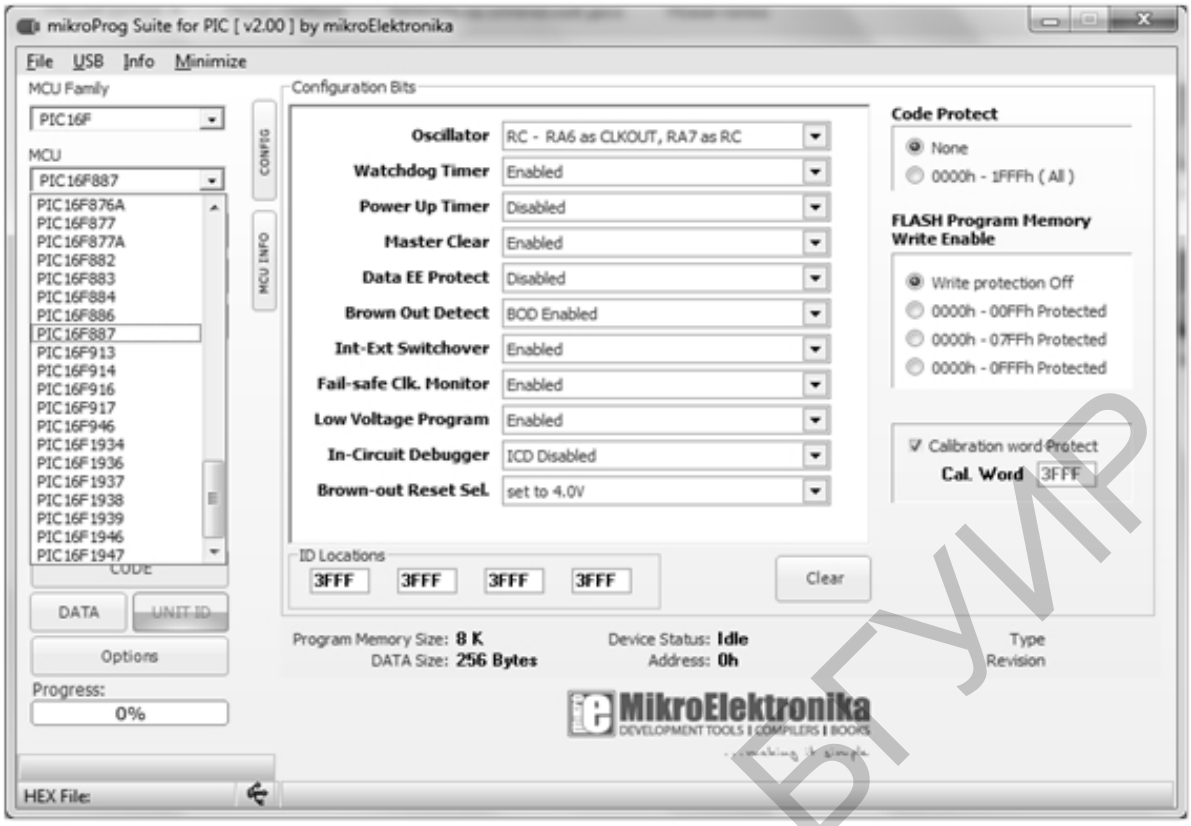


Рис. 1.48. Пример выбора микроконтроллера

Третьим шагом является выбор файла прошивки. Для этого можно воспользоваться кнопкой «Load» (рис. 1.49) на панели в левой части рабочего окна программы. Файл прошивки появляется после компиляции в рабочей директории проекта и имеет расширение *.hex (рис. 1.50).

Рис. 1.49. Пример выбора новой прошивки

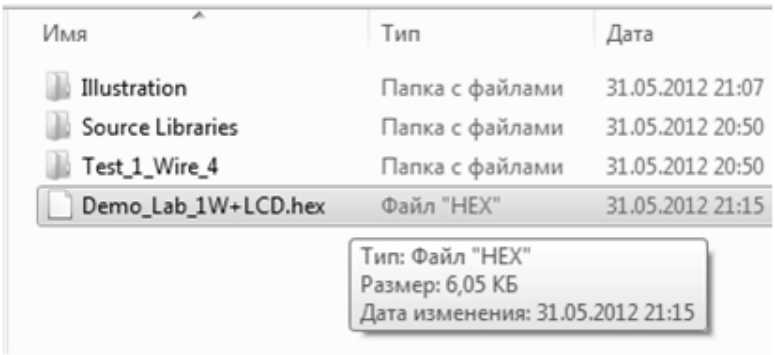


Рис. 1.50. Пример файла прошивки

Успешный выбор файла прошивки подтверждается отображением пути к файлу внизу рабочего окна программы (рис. 1.51).

Четвертый шаг заключается в выборе конфигурационных битов микроконтроллера в средней части рабочего окна программы. В простых проектах обычно необходимо и достаточно выбрать внешний тип тактирования «HS»в первой выпадающей строке «Oscillator».

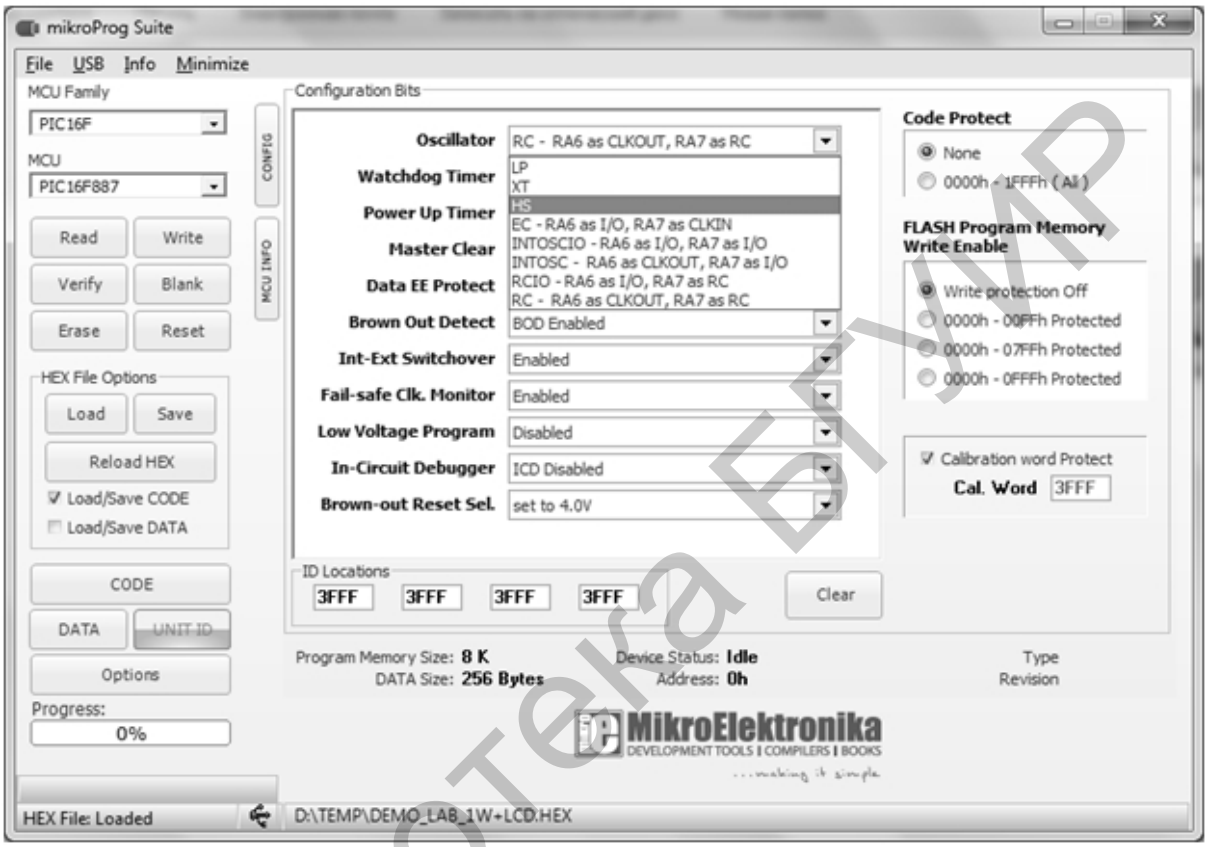


Рис. 1.51. Пример выбора внешнего осциллятора HS

Последним, пятым шагом является команда к началу процесса прошивки, которая дается при помощи кнопки «Write» на панели в правой части рабочего окна программы. После подачи команды следует дождаться окончания процесса прошивки, ход которого отображается при помощи специальной строки и статусных подписей на панели в левой части рабочего окна программы.

Использование анализатора Logic-U

Использование приложения SaleaeLLC для анализа логических сигналов включает в себя три шага. Первый шаг заключается в выборе дискретных отсчетов для анализа (минимум 1 млн точек, рис. 1.52) и частоты дискретизации (например 1 МГц, рис. 1.53) при помощи выпадающих строк левой верхней области рабочего окна программы.

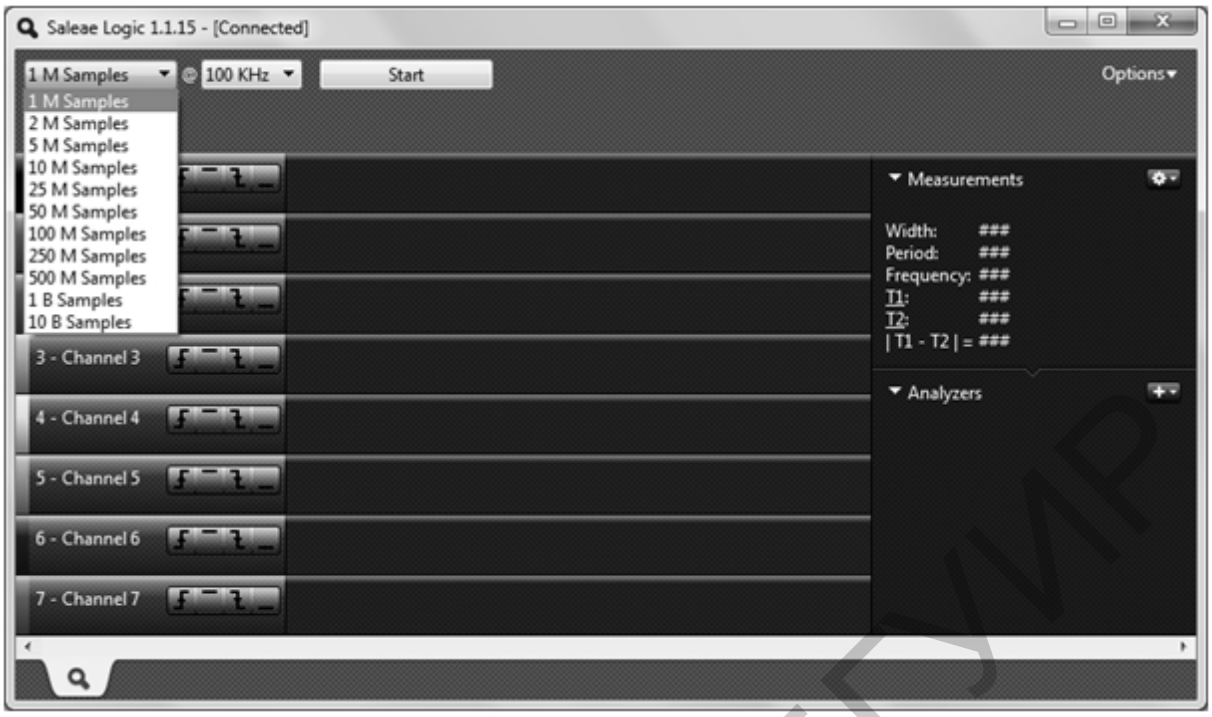


Рис. 1.52. Выбор количества накапливаемых данных

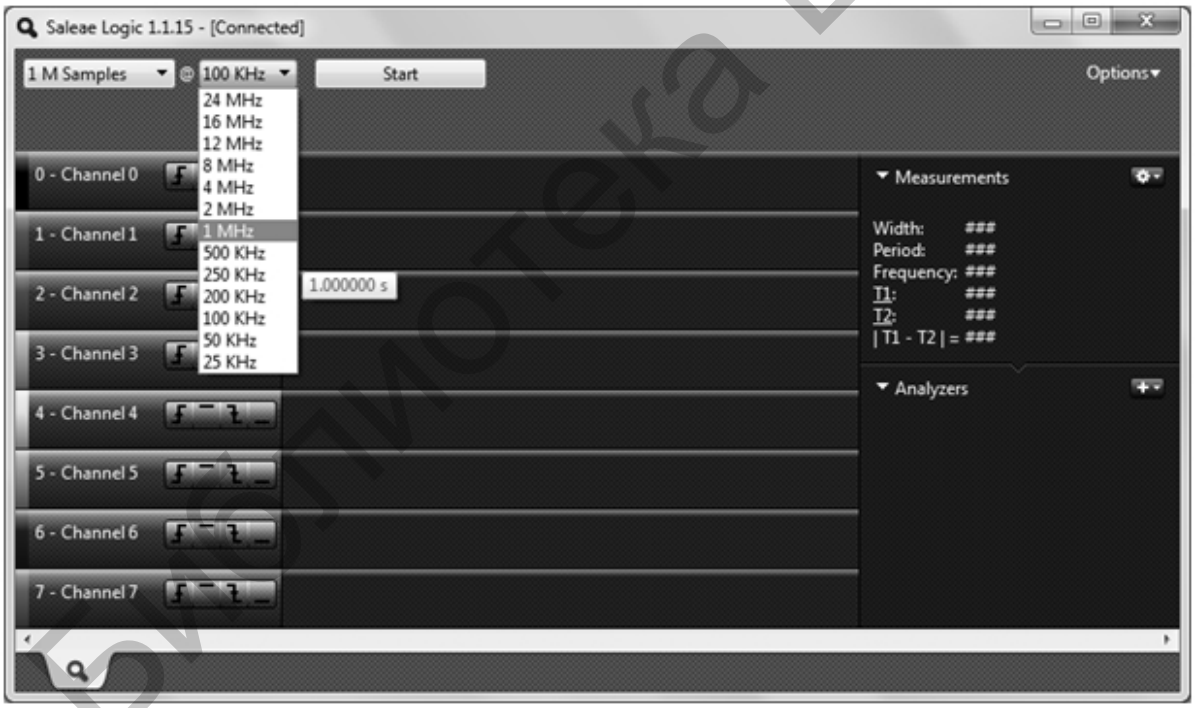


Рис. 1.53. Выбор частоты дискретизации

Второй шаг заключается в сборе статистических данных. Сбор данных можно начать кнопкой «Start», находящейся в верхней части рабочего окна программы, или Ctrl+R. Процесс сбора статистики иллюстрируется в специальном окне, которое открывается и закрывается автоматически. Продолжительность процесса сбора данных определяется заданными на первом шаге параметрами и показывается в специальной всплывающей подсказке.

Третий шаг заключается в анализе полученных данных. По окончании сбора статистики в соответствующих каждому каналу цифрового анализатора областях в центре рабочего окна программы появляются графики снятых данных и общая ось времени над ними. Масштабировать ось времени можно при помощи колесика мыши, при этом максимальный масштаб оси соответствует продолжительности процесса сбора статистики. Уменьшение масштаба от максимального и наведение указателя мыши на интервал между фронтами логического сигнала позволяют соответственно рассмотреть форму сигнала и измерить продолжительность непрерывных логических уровней сигнала (рис. 1.54). Продолжительность логических уровней показывается в правой части рабочего окна программы.

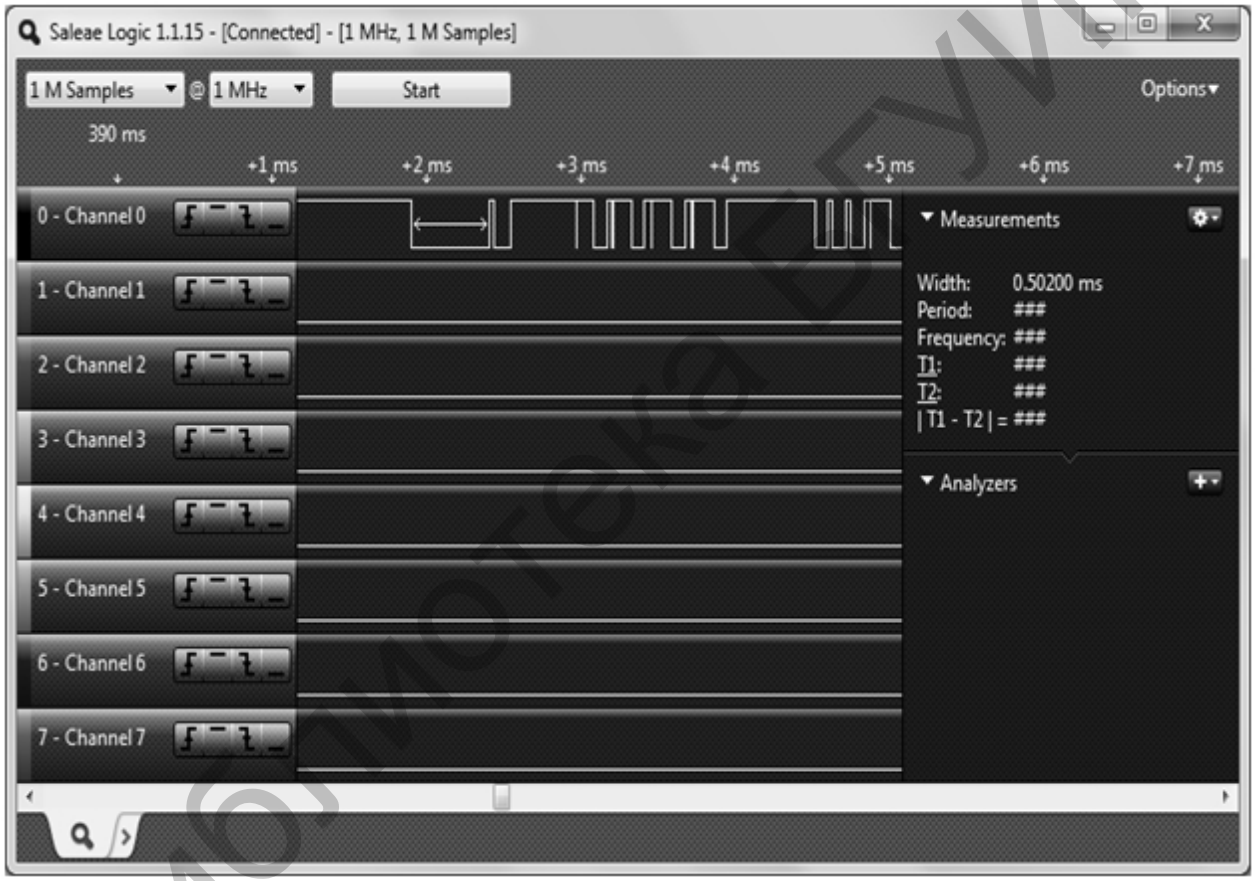


Рис. 1.54. Отображение формы и измерение длительности сигнала

1.2.7. Ознакомление с лабораторным макетом

Лабораторный макет, представленный на рисунке во введении, построен на базе платформы EasyPIC6. Эта платформа спроектирована компанией Mikroelektronika и обладает широким набором периферии для разработки и отладки приложений на базе восьмибитных микроконтроллеров PIC компании Microchip. Платформа универсальна, так как позволяет устанавливать, прошивать и проводить внутрисхемную отладку различных моделей микроконтроллеров PIC в корпусах DIP8, DIP14, DIP18, DIP20, DIP28 и DIP40.

Схематическое изображение платформы EasyPIC6 представлено на рис. 1.55, расшифровка обозначений функциональных блоков дана ниже. Более подробные сведения приведены в [7].

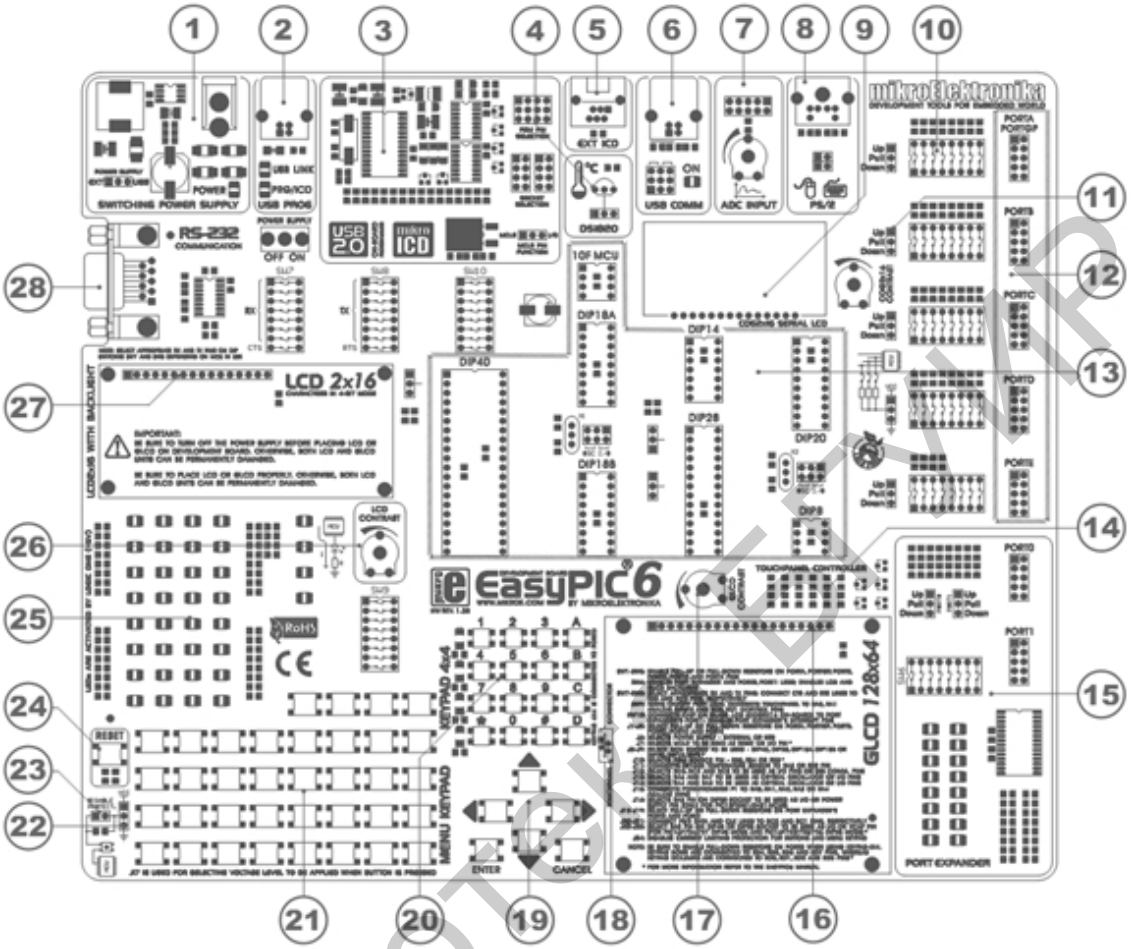


Рис. 1.55. Расположение периферии на платформе EasyPIC6

Расшифровка обозначений функциональных блоков:

1. Стабилизатор питающего напряжения.
2. USB-разъем встроенного программатора.
3. Встроенный программатор USB 2.0 с поддержкой внутрисхемного отладчика mikroICD.
4. Посадочное место цифрового термометра DS1820.
5. Разъем подключения внешнего отладчика MICROCHIP (ICD2 или ICD3).
6. Разъем USB для передачи пользовательских данных.
7. Тестовые входы АЦП.
8. Разъем PS/2.
9. Встроенный символьный ЖК-индикатор 2×16.
10. Переключатели для подключения подтягивающих резисторов.
11. Перемычка для подтягивания портов к земле/питанию.
12. Контакты портов ввода-вывода микроконтроллера.

13. Посадочные места для микроконтроллеров PIC.
14. Контроллер сенсорной панели.
15. Порты расширения.
16. Разъем подключения графического ЖК-индикатора 128×64.
17. Потенциометр графического ЖК-индикатора 128×64.
18. Разъем подключения сенсорной панели.
19. Кнопки для навигации меню.
20. Клавиатура 4×4.
21. Кнопки для эмуляции цифрового ввода.
22. Перемычка для выбора логического уровня при нажатии.
23. Перемычка для подключения защитного резистора.
24. Кнопка сброса (reset).
25. 36 светодиодов для индикации логических состояний.
26. Потенциометр текстового ЖК-индикатора.
27. Разъем подключения текстового ЖК-индикатора.
28. Разъем для интерфейса RS-232.

1.2.8. Методические рекомендации по настройке EasyPIC6

Перед тем как приступить к работе с макетом, следует путем установки правильной конфигурации переключателей и перемычек на плате сформировать такое подключение узлов, которое соответсвует принципиальной электрической схеме для первой работы (рис. 1.56). Более подробные сведения приведены в [7, 8].

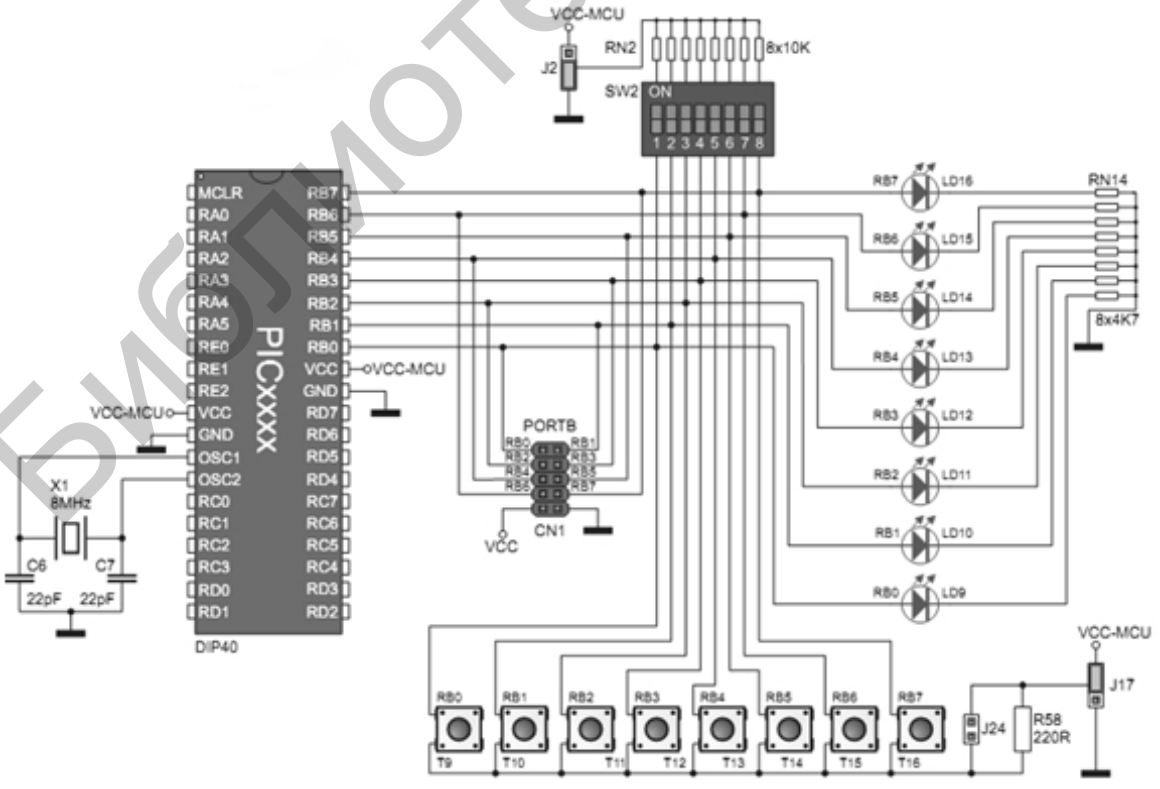


Рис. 1.56. Принципиальная электрическая схема подключения кнопок и светодиодов

***Проверять подключение рекомендуется по следующей методике*:**

1. Внимательно прочитайте и разберитесь в данной инструкции до начала выполнения каких-либо операций с макетом.

2. Проверьте, что находящиеся в составе макета платформа EasyPIC6 и анализатор Logic-U физически подключены к рабочим USB-портам компьютера и для них установлены драйверы в операционной системе. Драйверы в ОС Windows XP устанавливаются для каждой пары «разъем порта – устройство» отдельно.

3. Удостоверьтесь, что макет обесточен. Визуально удостоверьтесь в отсутствии повреждений и коротких замыканий внутри макета и на внешних разъемах. Все работы с коммутацией электронного оборудования настоятельно рекомендуется выполнять при выключенном питании. При присоединении/отсоединении модулей ЖКИ, элементов со значительной электрической емкостью, элементов с силовым питанием отключение питания ОБЯЗАТЕЛЬНО!

4. Отключите всю неиспользуемую периферию от портов расширения A, B, C, D, E, 0, 1 платформы EasyPIC6. Модули текстового и графического ЖК-дисплеев, а также термодатчик DS18B20, расположенные в специальных разъемах, можно оставить подключенными.

5. Установите микроконтроллер PIC16F887 в корпусе DIP40 в разъем DIP40. На корпусе микроконтроллера подписана его модель. Удостоверьтесь, что в остальных разъемах для микроконтроллеров ничего не установлено.

6. Установите кварцевый осциллятор с частотой 8 МГц в гнездо OSC1 и удостоверьтесь в его подключении перемычками J13 к контактам RA6 и RA7 порта микроконтроллера (положение «OSC» замкнуто).

7. В общем случае переведите все внешние подтягивающие резисторы портов A, B, C, D, E в положение отключено при помощи установки блоков переключателей SW1-SW5. При этом рекомендуется оставить резисторы «подтягивающими» к питанию +5В (перемычки J1-J5 «Pull Up»). Некоторые необходимые для конкретной работы резисторы можно оставить подключенными.

8. Для подключения светодиодов необходимо перевести переключатели SW9[1-4] (около светодиодов) в положение «On», в соответствии с подписями портов.

9. Для подключения кнопок необходимо установить перемычку J17 в верхнее положение, тем самым подсоединив кнопки к напряжению питания +5В, и применить защитный резистор R58, удостоверившись, что перемычка J24 не замкнута.

10. Для подключения ЖК-дисплеев достаточно их установить в специальные разъемы и подключить подсветку при помощи переключателя SW6[8] – «On» («BACKLIGHT» в правом нижнем квадранте платы). Контрастность монохромного изображения регулируется потенциометрами P3 и P4 для графического и текстового дисплеев соответственно.

11. Для подключения COM-порта через микросхему согласования уровней напряжения MAX202 и работы с ним через аппаратный UART микро-

контроллер требуется перевести переключатели: SW7[8] – «On», SW7[7-1] – «Off», SW8[8] – «On», SW8[7-1] – «Off», что подключит линии передачи данных Rx и Tx к контактам RC7 и RC6 соответственно. Также необходимо установить переключатель SW1[6] – pullUp (подтягивающий резистор №6 порта расширения A) для обеспечения повышенной нагрузочной способности линии RA5 и работы термодатчика в паразитном режиме питания.

12. Для подключения цифрового термометра DS18B20 требуется установить его в разъем TS1. Далее убедиться в том, что перемычка J11 (около термометра) находится в положении RA5, тем самым подключает его к одноименному контакту микроконтроллера. По нарисованному контуру датчика на плате убедиться в правильном расположении контактов, иначе датчик перегреется и задымит, после чего сгорит либо датчик, либо платформа EasyPIC6.

13. При необходимости подключите контакты логического анализатора Logic-U к точкам съема данных. Восемь цветных проводов соответствуют восьми каналам данных анализатора. Белый и серый провода соответствуют задаваемым логическим уровням единицы и нуля (питания и земли). Их подключение желательно, но необязательно, так как и EasyPIC6, и Logic-U работают на напряжении питания +5 В относительно земли 0 В, получаемом от USB-портов одного и того же компьютера. Кроме того, подключение этих двух контактов не позволит полностью обесточить макет одним лишь переключателем питания, EasyPIC6 получит паразитное питание от Logic-U.

14. Если на этапе прошивки возникает проблема управления питанием микроконтроллера, которая делает навозможной операцию прошивки, следует дополнительно убедиться в том, что периферийные модули (например, CAN-SPI, Ethernet2, Easy Wi-Fi) непосредственно отключены от платформы (портов A, B, C, D, E, 0, 1) или их цепи питания разомкнуты специальными переключателями на внешнем корпусе макетов, а также в том, что на плате нет внешних повреждений и коротких замыканий.

***Выполнять лабораторную работу рекомендуется по следующей методике*:**

1. Внимательно изучите теоретический материал и поставленную задачу. Разберитесь в методике работы с макетом и методике выполнения лабораторной работы, продумайте план действий до начала лабораторной работы. Изучите документацию на используемые аппаратные средства. Разработайте алгоритм программы, решающей поставленную задачу. Сделайте заготовку отчета по лабораторной работе.

2. Подготовьте макет к работе (см. пп. 1–15 методики проверки подключения макета).

3. Запустите рабочую среду MicroC PRO for PIC. Создайте новый проект, при необходимости предварительно закрыв предыдущий. При создании проекта учтите информацию о испольуемой модели микроконтроллера, источнике и частоте тактирования, необходимых для работы библиотеках (см. п. 1.2.8).

4. На основе заранее разработанного алгоритма и документации на электронные компоненты напишите код программы, рещающей поставленную задачу. Проведите компиляцию кода программы в прошивку для микроконтроллера (кнопка Build). Если компиляция не завершается успешно с первого раза и возникают ошибки – это нормально для начинающего программиста. Начните исправление ошибок с самой первой, о которой сообщает среда разработки. Часто первая ошибка по цепочке вызывает другие. Повторяйте до успешной компиляции.

5. При необходимости осуществите симуляцию разработанной программы в среде виртуального моделирования, например Proteus. В такой среде можно собрать принципиальную схему электронного устройства из типовых компонентов с известным поведением и проверить поведение системы под управлением разработанной программы.

6. Прошейте микроконтроллер при помощи программы microProg Suite for PIC (кнопки «Program» или «Build & Program», также можно запустить программу отдельно через ярлык и самостоятельно указать прошивку и параметры конфигурационных битов микроконтроллера). Если программа прошивки выдает сообщение о несовместимости прошивки и устройства, проверьте правильность выбора микроконтроллера. Если программа выдает ошибку о невозможности управления питанием микроконтроллера, действуйте согласно п. 15 методики проверки подключения макета.

7. Если прошивка прошла успешно, но микроконтроллер не работает, проверьте правильность выбора источника тактирования. Если прошивка прошла успешно, но контроллер работает не так, как ожидается, вероятнее всего код программы, из которой скомпилирована прошивка, описывает не то, что под ним понимает автор программы. Различия междум скопилированной и записанной прошивкой практически исключаются из-за автоматических проверок. В роцессе разработки программы подавляющее число ошибок является программными. Случаи шибок при проектировании архитектуры микроконтроллера единичны, но все же случаются и наблюдаются на всей линейке чипов с одинаковой архитектурой. В процессе длительной эксплаутации оборудования очень редко (а вот в условиях космоса и повышенной радиоактивности – заметно чаще) могут возникать и постепенно накапливаться аппаратные поломки, обычно вызывающие программные сбои. В таких условиях проще не искать ошибки, а делать систему с резервированием и мажоритарным принятием решений. Обычно в процессе нарушения условий хранения или эксплуатации, в том числе от электростатических разрядов и резких скачков напряжения, микроконтроллер сгорает целиком или сгорает нескольких его портов (контактов). Такие ошибки выявляются элементарными тестами ввода-вывода.

8. Если прошивка прошла успешно, при необходимости можно воспользоваться внутрисхемным отладчиком ICD (Inner Circuit Debugger). Это специальный инстрмент на основе отдельной микросхемы на плате EasyPIC6 с программным обеспечением, встроенным в среду MicroC PRO for PIC. Данный инструмент предоставляет возможность остановки исполняемой микроконтролле-

ром в реальном времени программы и пошагового анализа выполняемых операций и происходящих изменений с отображением их прямо в среде разработки с указанием на соответствующие участки исходного кода исполняемой программы.

9. Для визуального исследования и подтверждения работы программы в микроконтроллере рекомендуется использовать анализатор логических сигналов и протоколов Logic-U. Это устройство со специальным ПО, которое предоставляет возможность записывать до восьми каналов логических сигналов и автоматически распознавать и расшифровывать в сделанных записях распространенные протоколы передачи данных. Съем данных можно осуществить в любой физически доступной части электрической схемы, например прямо на входах/выходах микроконтроллера.

10. После завершения выполнения задания, проведения необходимых исследований и измерений и сохранения данных для отчета следует закрыть все программы и выключить орборудование.

11. При возникновении ошибок среды MicroC PRO for PIC следует ее перезапустить и при необходимости перезагрузить компьютер, так как одним из следствий возникших ошибок является непредсказуемость работы среды и недостоверность отображаемой информации (например, выдается сообщение об успешной перекомпиляции программы, но на самом деле файл прошивки не обновился).

12. При использовании в работе COM-порта интерфейса RS-232 и модулей UART для осуществления ввода-вывода данных со стороны компьютера рекомендуется использовать программы типа «Терминал», например отдельное приложение Terminal 1.9b (см. п. 3.2.2). В среду MicroC PRO for PIC также встроена программа-терминал, но иногда она работает нестабильно.

1.2.9. Архитектрура и регистры микроконтроллера PIC16F887.

Структурная схема микроконтроллера PIC16F887 показана на рис. 1.57.

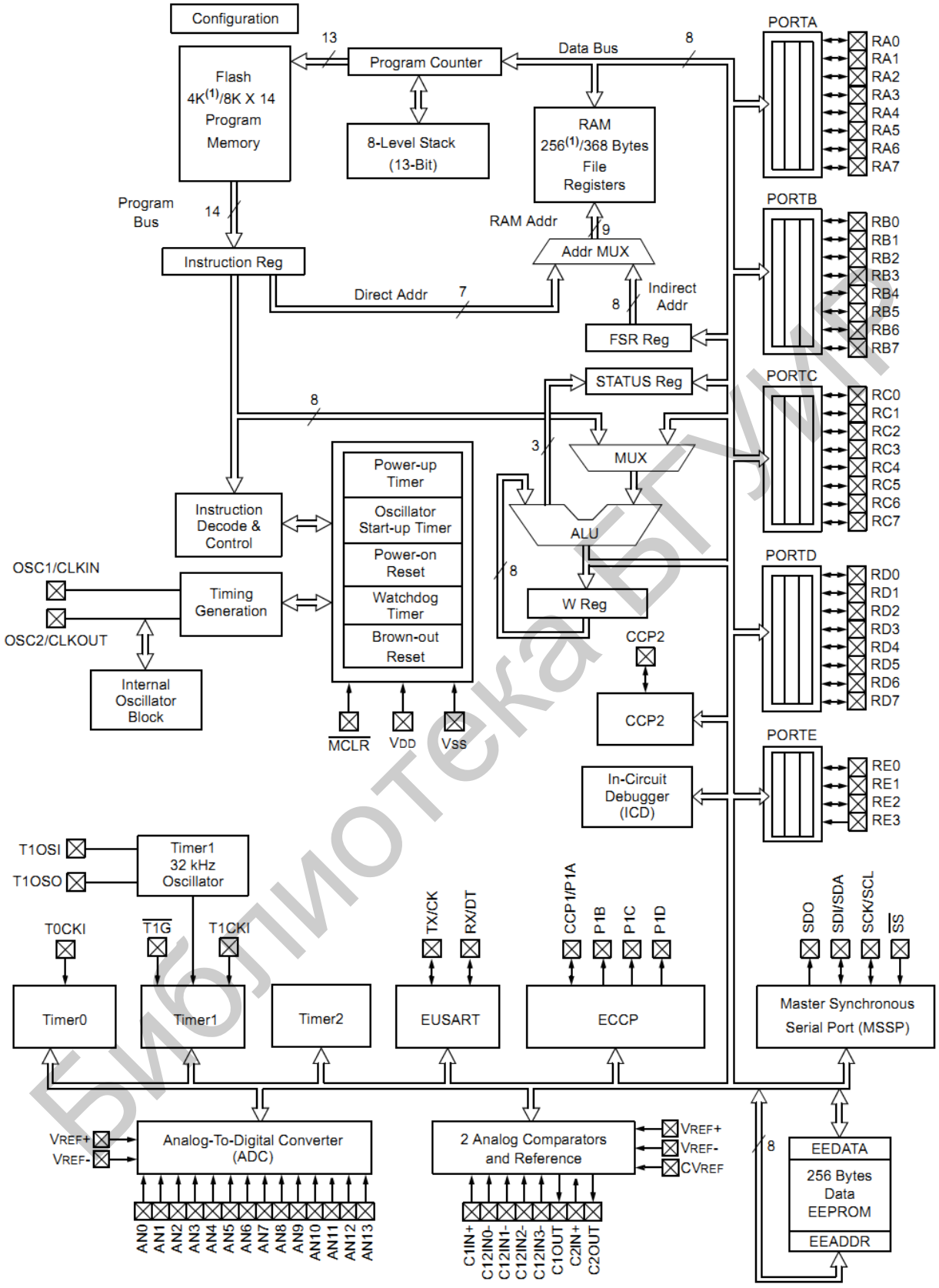


Рис. 1.57. Структурная схема микроконтроллера PIC16F887

Восьмиразрядный микроконтроллер PIC16F887 разрабтан компанией Microchip. Как и любой современный (2015 г.) микроконтроллер, он содержит набор функциональных блоков, минимально необходимый для работы полнофункционального микрокомпьютера, размещенный на одном кристалле внутри цельного корпуса.

К числу этих блоков относятся центральный процессор из операционных и статусных регистров, регистра и декодера инструкций, арифметико-логического устройства; постоянная и оперативная память; таймеры и цепи сброса; внутренний осциллятор и цепи тактирования; счетчик адреса программ и аппаратный стэк; регистры портов ввода-вывода; периферийные блоки в составе: энергонезависимой пользовательской памяти, модуля синхронного последовательного интерфейса, модуля универсального синхронно-асинхронного приемопередатчика, трех таймеров, четырнадцатиканального аналого-цифрового преобразователя, двух аналоговых компараторов, модуля широтно-импульсной модуляции. Отдельные шины адресов и инструкций связывают центральный процессор с памятью. Общая шина данных связывает процессор с памятью и периферйными модулями. Более подробные сведения приведены в [3, 9, 10].

Для выполнения данной лабораторной работы следует знать следующие подробности о регистрах.

Микроконтроллер является восьмиразрядным, следовательно, он обрабатывает данные группами по 8 бит, поэтому все адреса, переменные и данные также хранятся и обрабатываются группами по 8 бит. Исключение составляют некоторые специальные регистры и таймеры.

Порты ввода-вывода подключены к регистрам PORTX, где X – название порта A, B, C, D, E. Порт E содержит четыре бита, остальные порты – восемь. К регистру данных порта ввода-вывода можно обращаться, как к переменной. При этом в случае обращения к переменной в микроконтроллере к шине данных подключается модуль оперативной памяти, а вместо имени переменной подставляется адрес соответсвующей ячейки в оперативной памяти, по котрому следует осуществить запись или чтение. В случае обращеня к регистру порта или периферийного модуля вместо имени порта осуществляется подключение к общей шине данных соответствующего регистра.

Порты ввода-вывода могут быть сконфигурированы с помощтю регистров TRISX либо как входы («TRISX = 0xFF;»), либо как выходы («TRISX = = 0x00;»). Значения в регистрах TRIS являются управляющими для защелок, однонаправленно подключающих к ножкам порта микроконтроллера регистр PORT либо для чтения, либо для записи. Обновления данных в регистре и на выходе происходит с частотой тактирования микроконтроллера.

1.2.10. Пример решения типового задания

Формулировка задания в примере: вывести на порт С микроконтроллера результат операции побитное исключающее ИЛИ между предыдущим значением порта С и значением на входе порта В, заданным при помощи кнопок.

Код программы:

```
// Объявление и инициализация переменных
char buffer = 0;

// Объявление функции главной программы
void main()
{
// Отключение аналоговых входов для PIC16F887
   ADCON0 = 0x00;
   ANSEL  = 0;
   ANSELH = 0;
   C1ON_bit = 0;
   C2ON_bit = 0;

// Отключение аналоговых входов для PIC18F4550
   ADCON0 = 0;
   ADCON1 = 0x0F;
   CMCON = 7;

// установки направления порта В «вход» и порта С «выход»
   TRISB = 0xFF;
   TRISC = 0x00;
   PORTB = 0;
   PORTC = 0;

// бесконечный цикл для исполнения программы без остановки
   while(1)
   {
     // проверка активности на порту Б и чтение его
     if(PORTB != 0x00)
       buffer = buffer ^ PORTB;

     // вывод нового значения на порт Ц каждый цикл
     PORTC = buffer;
   }
}
```

1.3. Практическая часть

1.3.1. Контрольные вопросы

1. Что такое микроконтроллер?
2. Каковы математические и логические основы цифровой техники?
3. Как заполняется таблица соответствий десятичных, двоичных и шестнадцатеричных цифр?
4. Как доказать основные теоремы булевой алгебры?
5. Что является основными синтаксическими элементами языка MicroC?
6. Как работают директивы, функции, библиотеки?
7. Какие бывают переменные и операторы, как их использовать?
8. Какие основные аппаратные и программные инструменты используются для проектирования на микроконтроллерах и как ими пользоваться?
9. Какие аппаратные блоки есть в микроконтроллере PIC16F887?
10. Что входит в состав лабораторного макета и платформы EasyPIC6?
11. Какова методика работы с макетом и выполнения лабораторных работ?
12. Как выглядит принципиальная электрическая схема подключения кварца, кнопок и светодиодов к микроконтроллеру?

1.3.2. Содержание отчета

Содержание отчета аналогично по структуре для всех четырех лабораторных работ и включает следующее:

1. Титульный лист по форме из стандарта БГУИР.
2. Цель работы. Индивидуальное задание.
3. Принципиальная электрическая схема к лабораторной работе.
4. Разработанный алгоритм. Схема алгоритма и код программы.
5. Результаты выполнения работы. Описание аппаратных испытаний. Временные диаграммы передачи данных по изучаемым интерфейсам, снятые с помощью U-Logic. Результаты моделирования в Proteus.
6. Выводы по результатам лабораторной работы.
7. Приложения (например, при необходимости включить справочный материал или исходные коды библиотек).

1.3.3. Варианты заданий

1. Реализовать обработку нажатия двух любых кнопок, подключенных на PORTA, при нажатии первой – увеличивается значение числа A на +1, а при нажатии второй – на –1, в обоих случаях происходит пересчет и вывод результата.

2. Вычислить A = (45+78)·(180–25) OR 10011010b, вычесть из него число C = 18. Поместить по адресу 010h памяти RAM данных младшую десятичную цифру результата, а по адресу 012h – старшую. 000h – соответствует началу RAM-памяти.

3. Найти среднее арифметическое чисел A, B, C. Использовать следующие начальные значения A = 012h, B = 033h, B = 0Ah. Результат умножить на 22, используя операции сдвига. Младшую тетраду полученного числа разместить в старшей тетраде порта PORTB, а старшую – в младшей.

4. Реализовать обработку нажатия двух любых кнопок, одна из которых подключена на PORTA, другая – на PORTB. При нажатии первой – увеличивается значение числа PORTC на +1, а при нажатии второй – на –1, при этом происходит пересчет результата.

5. Реализовать программный стек для хранения чисел. Кнопка +1 и –1, затем кнопка вывода значений из стека и занесения в стек. (1-й порт – вводимое значение, 2-й – результат вывода из стека). Глубина стека – четыре восьмибитных элемента.

6. Поместить при помощи указателей результат вычисления N-й десятичной цифры числа π после запятой (N-номер бригады) по указанному адресу (адрес – указанная дата). Область памяти – 10 первых ячеек RAM, адрес вычисляется циклически по модулю. Результат извлечь из памяти и вывести на PORTB.

7. Найти разность чисел 4836 и 232. Младший байт результата поделить на 2. Поместить по адресу 025h внутренней памяти данных младший байт, а по адресу 030h – старший байт результата.

8. Найти адрес ячейки памяти данных путем перемножения двух чисел 0Ch и 0Eh. В эту ячейку записать результат логической операции «исключающее ИЛИ» между текущим содержимым регистра R0 и числа 09h.

9. Найти частное чисел 236 и 59. Результат умножить на 23, используя операции сдвига. По вычисленному таким образом адресу ячейки внутренней памяти данных разместить результат двойного декремента полученного числа.

10. В младшую тетраду порта PORTB вывести число десятков от числа 044h. Старшую тетраду необходимо оставить без изменений.

11. Найти сумму чисел 9701 и 32. Младший байт результата умножить на 4. Поместить старший байт в порт PORTC. Сбросить младшую тетраду байта порта.

12. Вычислить значение выражения (81+64)·(112–25) OR 10011010b, сохраняя промежуточные результаты в стеке.

13. Найти разность чисел 4801 и 209. Число десятичных единиц старшего байта результата поместить в старшую тераду порта PORTD. Младшую тетраду оставить без изменений.

14. В ячейки внутренней памяти данных 128h, 129h, 12ah занести число сотен, десятков, единиц числа 080h.

15. Вычислить младший байт адреса ячейки внутренней памяти данных 7XXh как произведение 0A1h и 7, поместить по этому адресу значение выражения NOT (0101001b OR 74).

16. При появлении положительного фронта на порте ввода-вывода PE3 найти произведение чисел 24 и 41, при отрицательном фронте на входе вычесть из результата 100.

17. При появлении положительного фронта на порте ввода-вывода PE3 найти произведение чисел 27 и 38, при отрицательном фронте на входе вычесть из результата 56.

18. При появлении положительного фронта на входе RA3 увеличивать глубину использования аппаратного стека на 1; при появлении отрицательного фронта на входе PD0 уменьшать на 1.

19. Реализовать программу типа «Змейка» на поле светодиодов портов A, B, C, D. Управление с кнопок порта E.

20. Реализовать генератор сигнала псевдослучайной двоичной последовательности с выводом значений на PORTC. Для ввода значений полиномов генерации и начального значения использовать PORTB для управления PORTA.

Лабораторная работа №2

ВЫВОД ДАННЫХ НА МОНОХРОМНЫЙ МАТРИЧНЫЙ ЖИДКОКРИСТАЛЛИЧЕСКИЙ ИНДИКАТОР

2.1. Цели работы

1. Изучение контроллеров монохромных ЖК-дисплеев, их интерфейсов, структуры памяти и системы команд на примере типовых KS0066 и KS0107.
2. Получение навыков работы с монохромными ЖК-дисплеями WH1602B2 и WDG0151 при помощи языка программирования mikroC и готовых библиотек.
3. Исследование эффективности алгоритмизации линейных вычислений с использованием языка MicroC и PIC микроконтроллера.

2.2. Теоретические сведения

В реальном мире большое значение имеет понятие об обратной связи – влияние каждого события на все последующие. При работе с техникой человеку требуется видеть результат своих действий, это особенно важно для электронной техники, работа которой сокрыта от органов чувств.

Наилучшим способом представления больших объемов информации традиционно считается графический. Для электронной техники это обуславливает удобство и популярность экранов и дисплеев. На практике важным критерием является экономический, поэтому широкое распространение получили самые простые и дешевые монохромные дисплеи, которые предоставляют возможность отображения текстовой и графической информации. Еще в 90-е годы прошлого века они были монополистами на рынке.

В настоящее время наиболее универсальными являются дисплеи, которые строят изображение из отдельных точек – пикселей. Управление большими массивами точек является задачей достаточно трудоемкой, и для ее решения используются специальные устройства (микросхемы) – контроллеры.

Контроллеры ЖК-дисплеев используют собственный внешний интерфейс, память и систему команд для управления массивами пикселей при помощи электричества. Так как задачи, решаемые такими контроллерами, являются тривиальными, то в сложившейся мировой практике сами виды контроллеров и их системы управления для каждого вида контролируемых устройств являются аналогами.

Например, для управления монохромными текстовыми ЖК-дисплеями одними из первых были контроллеры HD44780 и аналогичные KS0066. Они обеспечивают знакогенерацию (построение изображения цифр, букв и других символов из пикселей) и обеспечивают управление массивами до двух строк по 40 символов. Для управления монохромными графическими ЖК-дисплеями ис-

пользуются контроллеры KS0107, которые не содержат знакогенератора, но позволяют работать с отдельными точками в массиве 64×64 пикселя. Теперь существует гораздо больше контроллеров, но для дисплеев указанного типа все они имеют интерфейс и систему команд, аналогичные первым KS0066 и KS0107. Подробное описание контроллеров дано в [11, 12].

2.2.1. Описание контроллера KS0066. Интерфейс, система команд, временные диаграммы

Контроллер KS0066 фирмы Samsung наряду с контроллером HD44780 от фирмы Hitachi из-за своей повсеместной распространенности является фактически промышленным стандартом для контролеров черно-белых жидкокристаллических знакосинтезирующих дисплеев. Данный контроллер имеет параллельный 4- и 8-битный интерфейс, неизменную таблицу символов, а также таблицу для хранения пользовательских символов. Получил широкое распространение в различной технике (принтерах, роутерах, модемах) из-за своей дешевизны и простоты использования. На базе этого контроллера выпускалось огромное количество дисплеев, начиная с 8×1 (восемь символов в одной строке) и заканчивая 40×4 (содержащих два независимых управляющих чипа).

Основной интерфейс контроллера состоит из 14 линий (табл. 2.1).

Таблица 2.1

Интерфейс контроллера KS0066

Имя линии контроллера	**Назначение**
GND	Земля, общий провод
VCC	Напряжение питания (+5V)
VEE	Настройка контрастности
RS	Передача команды или данных
R/W	Чтение/запись
E	Стробирующий вход
D7-D0	Шина данных: - для 8-битного режима D7-D0; - для 4-битного режима D7-D4

Импульс напряжения на линии **E** указывает контроллеру на необходимость считать (записать) данные с (на) шину данных. Уровень R/W указывает, происходит запись в память дисплея или чтение из памяти дисплея. Если на этой линии установлена логическая «1», то происходит чтение из памяти дисплея, если логический «0», то запись в память. Уровень RS указывает, передается команда или данные. Если на этой линии установлена логическая «1», то передаются данные, если логический «0» – передается

команда. Шиной данных для 8-битного режима работы интерфейса является вся шина D7-D0, где старшим битом является линия D7, а младшим – линия D0. Для 4-битного режима работы интерфейса шиной данных является шина из линий D7-D4, где старшим битом является линия D7, а младшим – линия D4.

Структура памяти контроллера разбита на три части: DDRAM (Display Data RAM), CGROM (Character Generator ROM) и CGRAM (Character Generator RAM). **DDRAM** – перезаписываемая память дисплея, содержимое которой выводится на экран. Размер этой памяти составляет 640 бит (80 символов × 8 бит). **CGROM** – неизменная таблица символов. Символы, записанные в DDRAM, отображаются на экране согласно таблице символов в CGROM. Размер этой памяти составляет 10,080 бит (204 символа × 5 × 8 точек + 32 символа × 5 × 11 точек). **CGRAM** – перезаписываемая таблица символов с линейной адресацией предназначена для хранения пользовательских символов. Размер этой памяти составляет 512 бит (8 символов × 5 × 8 точек). Для выбора позиции в памяти существует виртуальный курсор (номер текущей ячейки памяти, AC), которым можно управлять посредством команд. По умолчанию при записи символа в ячейку курсор сдвигается вперед на одну позицию.

Система базовых команд контролера и время их выполнения представлены в табл. 2.2.

Таблица 2.2

Система команд контроллера KS0066

Команда	**Код команды**										**Пояснение**	**Время**
	RS	**R/W**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**		
Очистка дисплея	0	0	0	0	0	0	0	0	0	1	Очистка экрана, AC = 0, адресация AC на DDRAM	1,53 мс
Возвращение на начало строки	0	0	0	0	0	0	0	0	1	x	AC = 0, адресация AC на DDRAM, сброс сдвигов, начало строки адресуется в начало DDRAM	1,53 мс
Установка режима работы	0	0	0	0	0	0	0	1	ID	S	Выбираем направление сдвига курсора или экрана	39 мкс
Включение/ выключение дисплея	0	0	0	0	0	0	1	D	C	B	Выбираем режим отображения	39 мкс

Окончание табл. 2.2

Команда	**RS**	**R/W**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**Пояснение**	**Время**
Сдвиг курсора или экрана	0	0	0	0	0	1	SC	RL	x	x	Команда сдвига курсора/экрана	39 мкс
Установка настроек	0	0	0	0	1	DL	N	F	x	x	Определение параметров развертки и ширины шины данных	39 Мкс
Установка DDRAM-адреса	0	0	1	A6	A5	A4	A3	A2	A1	A0	Присвоение счетчику AC адреса в DDRAM	39 Мкс
Запись в RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Запись в RAM (DDRAM/CGRAM)	43 Мкс

Примечание: x – не имеет значения.

Пояснения к флагам системы команд приведены в табл. 2.3.

Таблица 2.3

Флаги системы команд контроллера KS0066

Обозначение флага	**Описание**
ID	Режим смещения счетчика адреса AC, 0 – уменьшение, 1 – увеличение
S	Режим сдвига экрана, 0 – сдвиг экрана не происходит, 1 – после записи в DDRAM экран сдвигается в направлении, определенном флагом ID
D	Наличие изображения, 0 – выключено, 1 – включено
C	Курсор в виде «_», 0 – выключен, 1 – влючен
B	Курсор в виде мерцающего символа «□», 0 – выключен, 1 – влючен
SC	Сдвиг содержимого экрана или курсора, 0 – сдвиг курсора, 1 – сдвиг экрана
RL	Направление сдвига содержимого экрана или курсора, 0 –влево, 1 – вправо
DL	Определение ширины шины данных, 0 – 4 линии, 1 – 8 линий
N	Режим развертки изображения на ЖКИ, 0 – одна строка, 1 – две строки
F	Размер матрицы символов, 0–5 × 8 точек, 1–5 × 11 точек
A6:A0	Адрес в DDRAM
D7:D0	Данные для записи в DDRAM/CGRAM

Инициализация контроллера на 4-битный режим приведена на рис. 2.1.

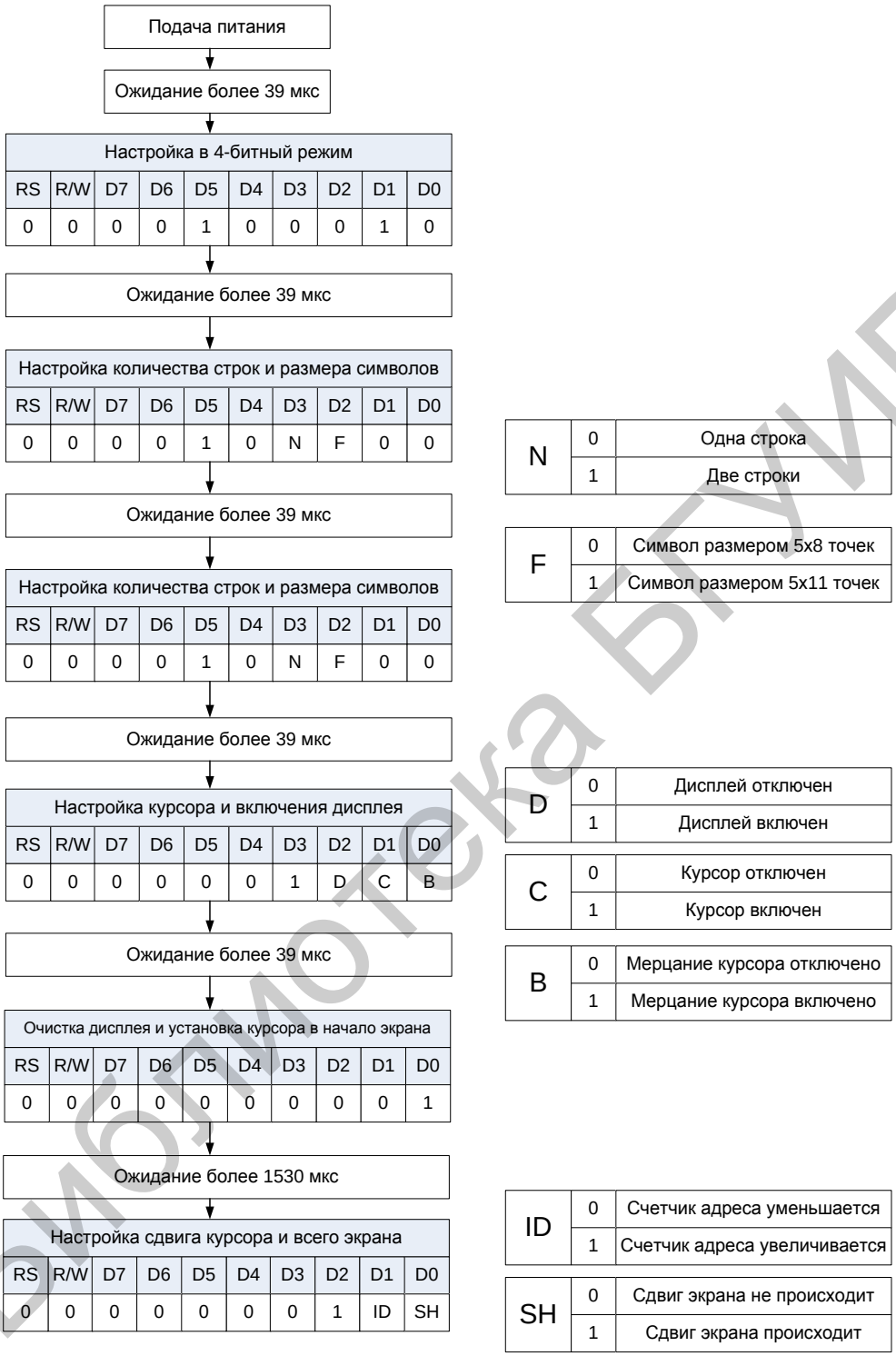


Рис. 2.1. Инициализация контроллера KS0066 в 4-битный режим работы

Таким образом, для инициализации контроллера в 4-битном режиме со следующими параметрами: две строки символов по 5 × 8 точек, дисплей включен, курсор и мигание курсора отключены, счетчик адреса увеличивается, сдвиг экрана не происходит – потребуется передать следующую последовательность команд: «0x22, 0x28, 0x28, 0x08, 0x01, 0x06».

Ниже приведены примеры временных диаграмм для записи команд и для записи данных в дисплей (рис. 2.2).

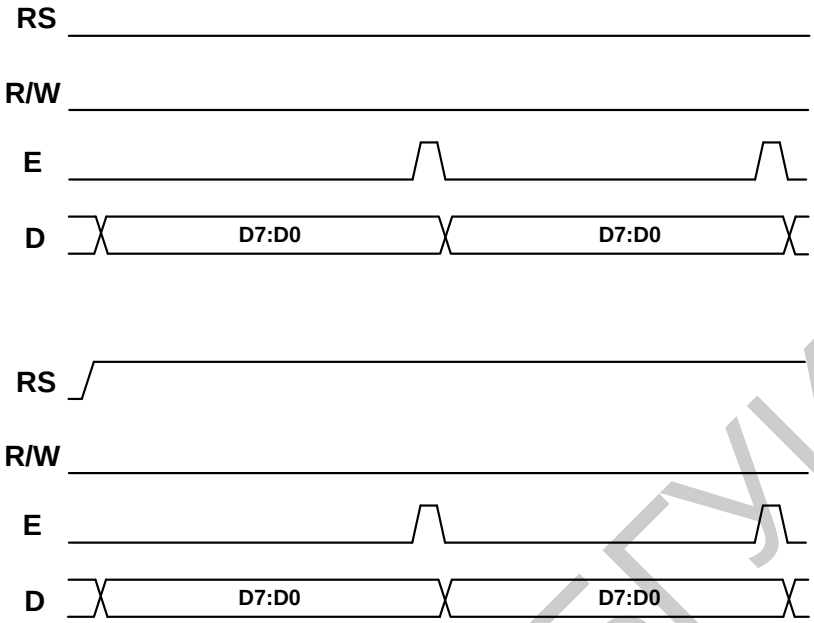


Рис. 2.2. Временные диаграммы записи команд и данных

Из временных диаграмм видно, что для записи команд требуется установить логические «0» на линиях RS и RW. Далее требуется установить код команды на шину данных, а затем стробировать линию E. Для передачи данных вначале требуется установить логическую '1' на линии RS. При работе в 4-битном режиме вначале передается старший полубайт, затем младший с учетом всех необходимых задержек.

2.2.2. Описание контролера KS0107. Интерфейс, система команд, временные диаграммы

Модуль графического ЖК-дисплея WDG0151 содержит матрицу 128 × 64 пикселя (условно x на y) и два контроллера KS0107 или аналогичных, каждый из которых управляет половиной экрана размерами 64 × 64 пикселя. ЖК-дисплей использует 20 контактов, список которых приведен в табл. 2.4.

Таблица 2.4

Список контактов модуля графического ЖК-дисплея WDG0151

Номер	Название	Уровень	Функция	Подключение
1	CS1	Низкий	Выбор первого контроллера (1–64)	RB0
2	CS2	Низкий	Выбор второго контроллера (1–128)	RB1
3	GND	Земля	Общий	GND
4	Vcc	Питание	Напряжение питания	+5V
5	Vo	Переменный	Подстройка контрастности	

Окончание табл. 2.4

Номер	Название	Уровень	Функция	Подключение
6	RS	Низкий/ высокий	Н: Инструкции В: Данные	RB2
7	R/W	Низкий/ высокий	Н: Запись данных В: Чтение данных	RB3
8	E	Высокий	Разрешение чтения/записи	RB4
9–16	D0–D7	Низкий/ высокий	Шина данных	RD0 – RD7
17	RST	Низкий	Сброс модуля ЖКИ	RB5
18	Vee	–	Выход отрицательного напряжения	
19	LED+	–	Анод подсветки дисплея	+5V
20	LED-	–	Катод подсветки дисплея	GND

Управление дисплеем происходит по алгоритму, изображенному на рис. 2.3, управление каждым из контроллеров происходит согласно временным диаграммам, представленным на рис. 2.4, и списку команд, представленных в табл. 2.5. Каждый контроллер оперирует командами, определяющими режим работы, адрес памяти данных (энергозависимой) и данные, которые записываются/читаются по этому адресу. Пределы адресации – 64 столбца пикселей по оси x, 8 строк по оси y. По каждому адресу находится 1 байт данных, соответствующий группе (линии по оси y) из восьми пикселей, таким образом, общее количество адресуемых по оси y строк одиночных пикселей также 64.

Например, для рисования одной точки выполняется следующая последовательность действий: при обращении к экрану происходит выбор контроллера по оси x ($x \leq 63$ или $x \geq 64$); установка адреса: координаты по x (0...64) и строки по y (0...7); чтение байта данных по этому адресу в буфер управляющего устройства (микроконтроллера); изменение одного устанавливаемого бита в буфере; снова выбор контроллера; установка адреса; запись байта из буфера в RAM дисплея. Линии рисуются в цикле отдельными точками, при этом счетчик цикла отсчитывает изменение координаты вдоль оси, на которую проекция линии длиннее. Отображение текста производится при помощи записываемых в энергонезависимую память управляющего устройства (микроконтроллера) 5-байтных шаблонов букв для знакогенерации, при этом размеры символов составляют 5 × 7 точек, их вывод осуществляется сразу побайтно, и может начинаться с любого пикселя по оси x, но привязан к одной из восьми строк по оси y.

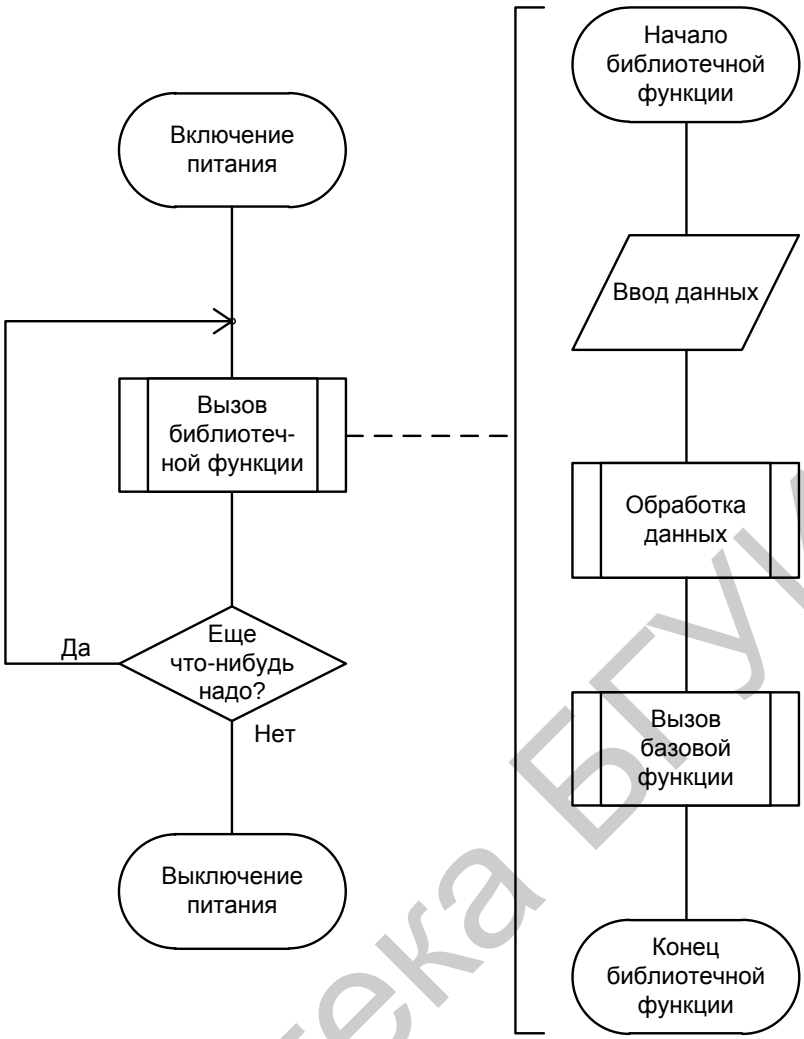
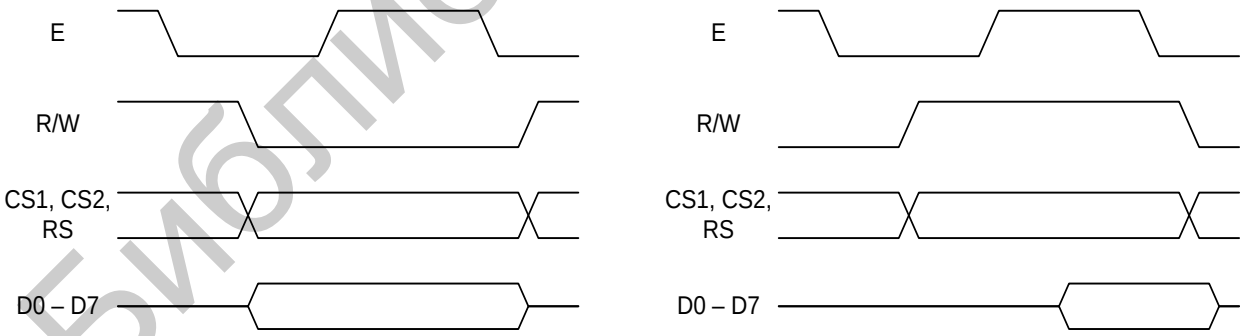


Рис. 2.3. Общий вид алгоритма управления графическим ЖК-дисплеем WDG0151



а б

Рис. 2.4. Общий вид временных диаграмм сигналов чтения и записи контроллера KS0107 или аналогичного:
а – запись данных; б – чтение данных

Таблица 2.5

Список команд контроллера KS0066

Инструкция	**RS**	**R/W**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**	**Функция**
Дисплей включен/ выключен	0	0	0	0	1	1	1	1	1	1/0	Управляет включением и выключением дисплея. Внутренний статус и RAM сохраняются: 0 – выкл., 1 – вкл.
Установить адрес (X)	0	0	1	0	Адрес по X (0 – 63)						Устанавливает адрес по X в соответствующем счетчике
Установить строку (Y)	0	0	1	0	1	1	1	Строка по Y (0 – 7)			Устанавливает адрес по Y в соответствующем регистре
Установить начальный адрес	0	0	1	1	Стартовая линия для отображения (0 – 63)						Указывает, где на экране располагается верхняя граница изображения (по оси Y)
Прочитать статус	0	1	Busy	0	On/Off	Reset	0	0	0	0	Читает состояние дисплея. Busy: 0 – готов, 1 – занят. On/Off: 0 – дисплей включен, 1 – дисплей выключен. Reset: 0 – нормально, 1 – сброс
Записать отображаемые данные	1	0	Записываемые данные								Записывает данные (DB0:7) в память RAM текущего изображения. После записи адрес X увеличивается на 1 автоматически
Прочитать отображаемые данные	1	1	Читаемые данные								Читает данные о текущем изображении из RAM (DB0:7)

Примечание: 1 – высокий логический уровень напряжения; 0 – низкий.

2.2.3. Библиотека для работы с LCD

Для работы с контролером KS0066 ЧБ ЖКИ была разработана библиотека, состоящая из функций управления, инициализации и вывода информации на дисплей. Библиотека помещена в стандартный файл библиотек языка программирования С и имеет расширение «.h».

Список доступных функций библиотеки LCD_lib.h

LCD_Init()

Описание. Данная функция производит инициацию дисплея и выставляет следующие настройки: 4-битный режим работы, двухстрочечная развертка изображения, размеры символов – 5×8 точек, отсутствие курсора и включение ЖКИ. Данная функция должна вызываться *первой* перед использованием ЖКИ.

Список аргументов: отсутствует.
Пример вызова функции:

```
LCD_Init();                 // Инициализация дисплея
```

LCD_Send_Cmd()

Описание. Данная функция является функцией исполнения командных кодов. В качестве кодов выступает байт данных, соответствующий коду команды, записанный в виде «старший_полубайт младший_полубайт».

Список аргументов: аргумент типа unsigned char.
Пример вызова функции:

```
LCD_Send_Cmd(0b00000001);          // Команда очистки дисплея
```

LCD_Goto()

Описание. Данная функция изменяет адрес курсора на экране. Установка происходит по двум параметрам – номер строки (1 или 2) и номер позиции в строке экрана (от 1 до 16).

Список аргументов: два аргумента типа unsigned char.
Пример вызова функции:

```
LCD_Goto(2,4);              // Курсор перемещается на на 4-ю позицию
                            // 2-й строчки
```

LCD_Send_Chr()

Описание. Данная функция выводит на ЖКИ один символ. Символ может быть задан как в виде 'символ', так и кодом таблицы ASCII.

Список аргументов: аргумент типа unsigned char.
Пример вызова функции:

```
LCD_Send_Chr('A');     // Вывод символа A
LCD_Send_Chr(0x42);    // Вывод символа B
```

LCD_Send_Text()

Описание. Данная функция выводит на ЖКИ строку слева направо. Строка однозначно задается в «двойных кавычках»

Список аргументов: аргумент типа указатель на char.
Пример вызова функции:

```
LCD_Send_Text("Hello");   // Выводит текст в скобках на ЖКИ
```

***Внимание*!** Во избежание порчи имущества не устанавливайте и не снимайте панель ЖКИ-дисплея, когда есть напряжение питания.

2.2.4. Методические рекомендации по настройке EasyPIC6

Для выполнения лабораторной работы следует путем установки правильной конфигурации переключателей и перемычек на макете сформировать такое подключение узлов, которое соответствует принципиальной электрической схеме (рис. 2.5, 2.6). Общее описание методики настройки дано в п. 1.2.8. Более подробные сведения приведены в [7, 8].

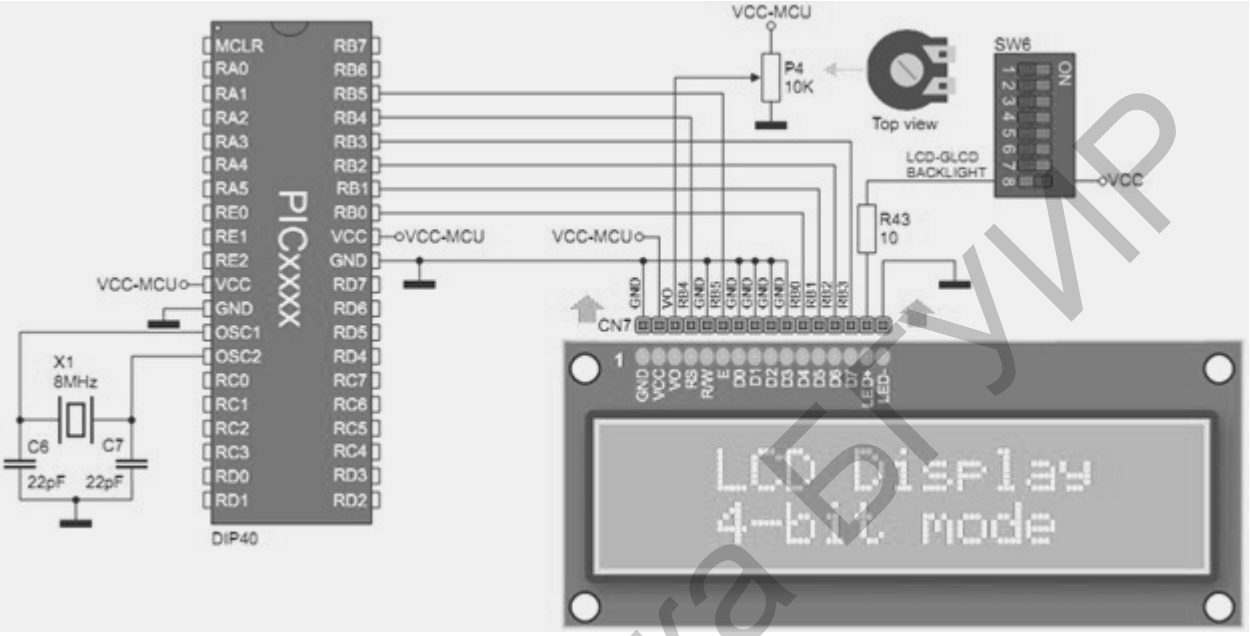


Рис. 2.5. Принципиальная электрическая схема подключения текстового ЖК-дисплея

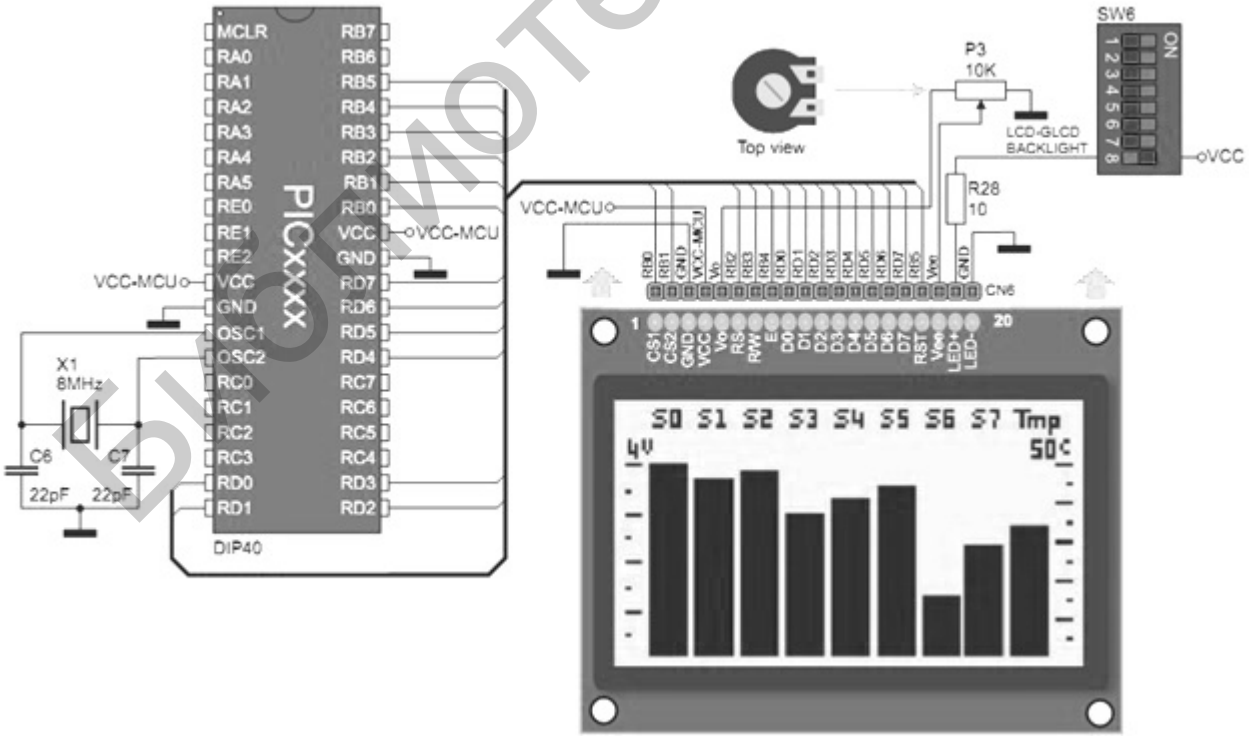


Рис. 2.6. Принципиальная электрическая схема подключения графического ЖК-дисплея

2.2.5. Пример решения типового задания

Формулировка задания в примере: вывести на текстовый дисплей при помощи открытой библиотеки «LCD_Lib» значение восьмибитного счетчика количества нажатий на кнопки, подключенные к порту С микроконтроллера.

Код программы:

```
// подключение библиотек из директории с проектом
#include "LCD_Lib_v03.h"

char clickCounter = 0;
char textSample[17] = {"Number of Clicks"};

void main()
{
   ADCON0 = 0x00;
   ANSEL  = 0;
   ANSELH = 0;
   C1ON_bit = 0;
   C2ON_bit = 0;

   TRISC = 0xFF;
   PORTC = 0;

 // Инициализация дисплея, установка курсора, вывод текста
   LCD_Init();
   Lcd_Goto(1,1);
   Lcd_Send_Text (textSample);

   while(1)
   {

// Вывод количества нажатий при активности порта С
      if(PORTC)
      {
      clickCounter++;
      Lcd_Goto(2,7);
      LCD_Send_Chr(clickCounter/100+48);
      LCD_Send_Chr(clickCounter%100/10+48);
      LCD_Send_Chr(clickCounter%10+48);
      }
   }
}
```

2.3. Практическая часть

2.3.1. Контрольные вопросы

1. Каково назначение и возможности контроллера KS0066?
2. Какие типы памяти присутствуют в KS0066?
3. Как выглядит интерфейс и принципиальная схема подключения KS0066?
4. Какова система команд и временные диаграммы работы KS0066?
5. Запишите процесс инициализации KS0066 в однострочном 4-битном режиме с курсором без мерцания (размер символов 5 × 8 точек).
6. Какие функции входят в библиотеку текстового дисплея и как их использовать?
7. Каково назначение и возможности контроллера KS0107?
8. Какие типы памяти присутствуют в KS0107?
9. Как выглядит интерфейс и принципиальная схема подключения KS0107?
10. Какова система команд и временные диаграммы работы KS0107?
11. Запишите процесс инициализации KS0107.
12. Какие функции входят в библиотеку графического дисплея и как их использовать?

2.3.2. Варианты заданий

1. Вывести на текстовый дисплей статическую информацию (например, фамилии и номер группы студентов).
2. Вывести на текстовый дисплей динамическую информацию (бегущая строка с позитивной мотивирующей фразой).
3. Изобразить на графическом дисплее стереометрическую фигуру (сферу, правильную пирамиду, куб).
4. Рассчитать и вывести на текстовый дисплей 30 знаков числа π.
5. Рассчитать и вывести на текстовый дисплей 10 знаков синуса, рассчитанного для параметра, равного текущей дате (номер дня в месяце).
6. Продемонстрировать построение взаимокорреляционной функции двух сигналов (видео- и радиоимпульс) в реальном времени (циклически) на графическом дисплее.
7. Построить на графическом дисплее функцию косинуса с курсором, управляемым кнопками, и выбирающим точку для отображения параметров функции (угла и значения).
8. Разработать меню на текстовом дисплее с навигацией с клавиатуры (до трех уровней вложенности и до четырех элементов на каждом уровне).
9. Разработать графопостроитель для кубической функции с меню для задания числовых параметров на графическом дисплее. Навигация по меню и задание параметров функции осуществляется с клавиатуры.

10. Рассчитать свертку двух восьмибитных сигналов, вводимых с клавиатуры, отобразить результат на текстовом дисплее.

11. Вычислить и постороить наглядный график дискретного преобразования Фурье для сигнала, записанного во flash-память микроконтроллера. Сигнал – несущая 10 кГц, амплитудно модулированная четырехуровневым сигналом, рассчитанным из разложения номера текущего дня нашей эры на пары битов, скорость изменения модулирующего сигнала 500 бод.

12. Реализовать на текстовом дисплее числовые часы реального времени с управлением от кнопок.

13. Реализовать на графическом дисплее графические радиальные часы реального времени с управлением от кнопок.

14. Реализовать программу типа «Змейка» на графическом дисплее с упавлением от кнопок.

15. Реализовать программу типа «Тетрис» на графическом дисплее с управлением от кнопок.

16. Реализовать программу типа «Танчики» на графическом дисплее с управлением от кнопок.

17. Реализовать программу типа «Пинг-понг» на графическом дисплее с управлением от кнопок.

18. Реализовать программу типа «Рогалик» на текстовом дисплее с применением программируемых символов дисплея и управлением от кнопок.

19. Реализовать программу типа «Трехмерный FPS» на графическом дисплее с управлением от кнопок.

20. Реализовать программу, выводящую свой собственный код на языке С на текстовый дисплей.

Лабораторная работа №3

ПЕРЕДАЧА ДАННЫХ ПО ИНТЕРФЕЙСУ RS-232

3.1. Цели работы

1. Изучение внутреннего устройства, принципа работы и настройки универсального синхронно-асинхронного последовательного приемопередатчика (USART).
2. Изучение интерфейса RS-232.
3. Приобретение навыков применения и анализа распространенного интерфейса USART для обмена данными с компьютером через COM-порт.
4. Приобретение практических навыков программирования микроконтроллеров семейства PIC и периферийных устройств.

3.2. Теоретические сведения

Universal Synchronous and Asynchronousserial Receiverand Transmitter (USART) – универсальный синхронно-асинхронный последовательный приемопередатчик. USART – внутренний блок микроконтроллера, предназначенный для побайтного обмена данными с другими устройствами.

3.2.1. Интерфейс RS-232 и COM-порт

RS-232 (Recommended Standard 232) – в телекоммуникациях стандарт последовательной синхронной и асинхронной передачи двоичных данных между терминалом и коммуникационным устройством.

Чаще всего используется в промышленном и узкоспециальном оборудовании, встраиваемых устройствах. Иногда присутствует на современных персональных компьютерах (рис. 3.1).

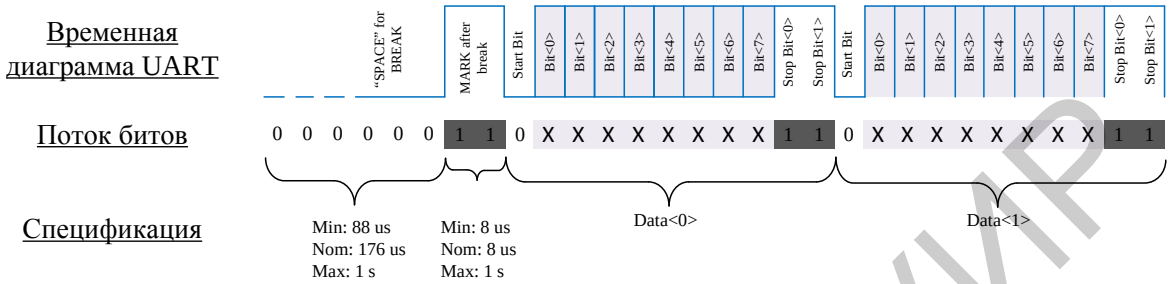
Рис. 3.1. Разъем DB-9 Female для интерфейса RS-232

В персональных компьютерах этот интерфейс используется COM-портом (Communication port).

Передача данных по интерфейсу RS-232 может осуществляться в двух режимах:

– асинхронная передача;
– синхронная передача.

По структуре это обычный асинхронный последовательный протокол, т. е. передающая сторона по очереди выдает в линию 0 и 1, а принимающая отслеживает их и запоминает (рис. 3.2).



Настройки UART :
Скорость = 250,000 бод; Длина слова = 8 бит; Число стоповых битов = 2; Четность = Нет.

Рис. 3.2. Последовательная асинхронная передача данных по интерфейсу RS-232

Данные передаются пакетами по одному байту (8 бит). Вначале передается стартовый бит полярности, противоположной состоянию незанятой линии (idle), после чего передается непосредственно кадр полезной информации от 5 до 8 бит.

Увидев стартовый бит, приемник выжидает интервал T1 и считывает первый бит, потом через интервалы T2 считывает остальные информационные биты.

Последний бит – стоповый бит (stop bits), соответсующий состоянию незанятой линии и говорящий о том, что передача завершена. Возможно 1; 1,5; 2 стоповых бита.

В конце байта перед стоповым битом может передаваться бит четности (parity bit) для контроля качества передачи. Он позволяет выявить ошибку в нечетное число битов (используется, так как наиболее вероятна ошибка в 1 бит).

При передаче в асинхронном режиме для связи используются 3 контакта: RxD (Receive), TxD (Transmit) и GND (Ground). При передаче в синхронном режиме для связи используются 9 контактов: DCD, RxD, TxD, DTR, GND, DSR, RTS, CTS и RI.

Последовательный порт персонального компьютера. *Последовательный порт (serial port), серийный порт, или COM-порт* (произносится «ком-порт», от англ. Communication port) – двунаправленный последовательный интерфейс, предназначенный для обмена битовой информацией. Последовательным данный порт называется, потому что информация через него передается по одному биту, бит за битом (в отличие от параллельного порта). Хотя некоторые другие интерфейсы компьютера, такие как Ethernet, Fire Wire и USB, также используют последовательный способ обмена, название «последовательный

порт» закрепилось за портом, имеющим стандарт RS-232C, и предназначенным изначально для обмена информацией с модемом.

Наиболее часто для последовательного порта персональных компьютеров используется стандарт RS-232C. Ранее последовательный порт использовался для подключения терминала, позже – для модема или мыши. Сейчас он используется для соединения с источниками бесперебойного питания, для связи с аппаратными средствами разработки встраиваемых вычислительных систем, со спутниковыми ресиверами, с приборами систем безопасности объектов и др. Распайка выводов порта приведена в табл. 3.1.

Таблица 3.1

Распайка выводов COM-порта

Номер	Обозначение	Направление	Сигнал
1	DCD	Вход	DataCarrierDetect
2	RxD	Вход	ReceiveData
3	TxD	Выход	Transmit Data
4	DTR	Выход	Data Terminal Ready
5	GND	–	Ground
6	DSR	Вход	Data Set Ready
7	RTS	Выход	Request To Send
8	CTS	Вход	Clear To Send
9	RI	Вход	Ring Indicator

Стандартные скорости обмена данными через COM-порт выбираются из следующего списка (в бодах): 600, 1200, 2400, 4800, 9600, 14 400, 19 200, 28 800, 38 400, 56 000, 57 600, 76 800, 115 200, 128 000, 256 000.

В COM-порте ПК используются следующие уровни сигналов: +5…+15 В соответствует логическому нулю, а –5…–15 В соответствует логической единице (рис. 3.3).

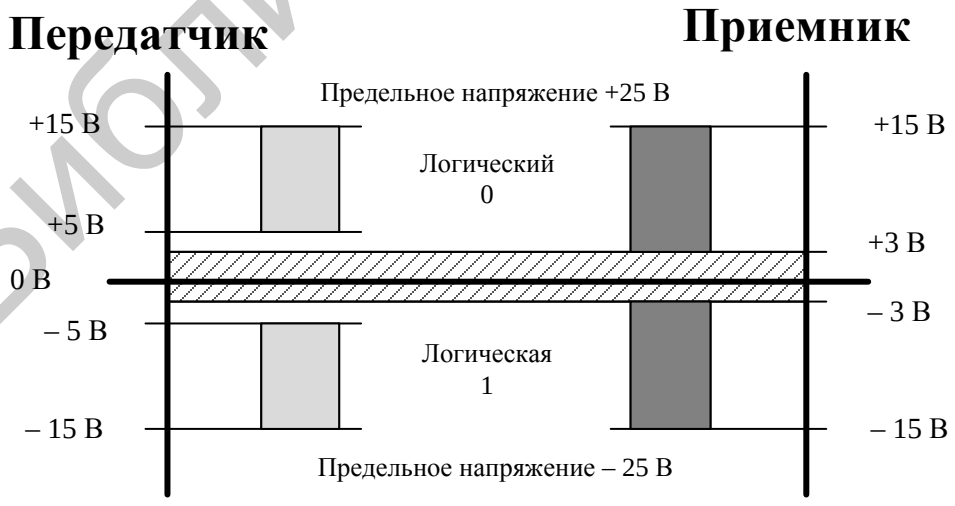


Рис. 3.3. Уровни управляющих сигналов RS-232

Использование интерфейса rs-232 в микроконтроллерах. Микроконтроллер PIC16F886 содержит в своем составе блок USART. Данный модуль используется для связи МК с внешними устройствами, содержащими в своем составе порты, поддерживающие последовательную передачу данных по интерфейсу RS-232. Это могут быть ПК, другой МК или другие внешние устройства.

3.2.2. Программы для работы с COM-портом

Передача и прием данных со стороны компьютера осуществляются при помощи программ типа «Терминал». Такое название сложилось исторически. В среду MicroC PRO for PIC встроена программа-терминал, но рекомендуется использовать внешнюю программу «Terminal 1.9b» (рис. 3.4) из-за большей стабильности работы последней.

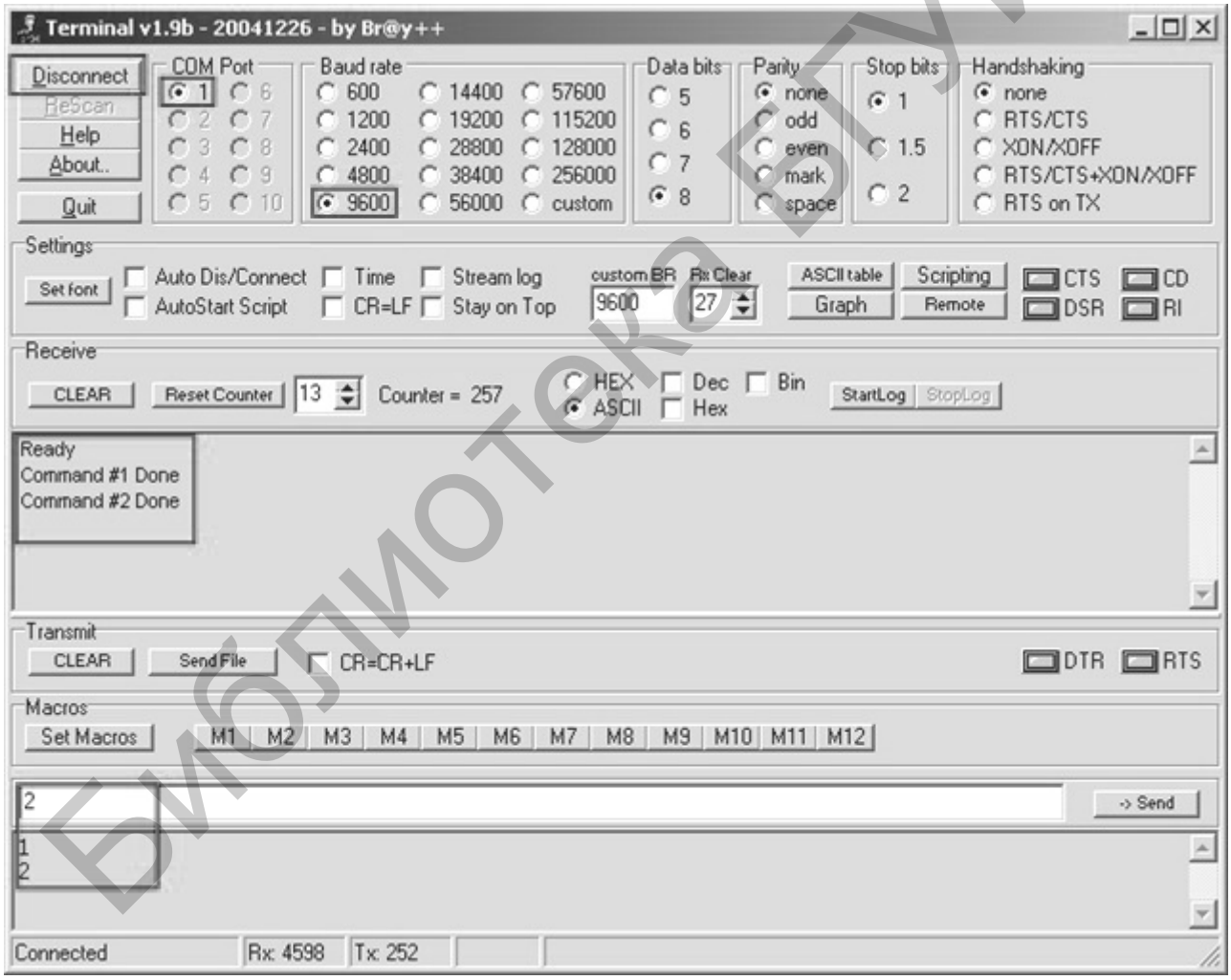


Рис. 3.4. Главное окно программы Terminal 1.9b

Перед установкой подключения следует выбрать номер COM порта в операционной системе и скорость обмена данными. Далее нажать кнопку «Connect» для подключения, вводить данные для передачи в нижней части окна программы, читать полученные данные в средней части окна программы. Для

отправки сообщений в шестнадцатеричном коде следует использовать приставку «$». Для отключения нажать кнопку «Disconnect». Следует помнить, что в операционной системе к одному COM-порту может одновременно иметь доступ лишь одна программа, поэтому перед установкой нового подключения убедитесь в завершении предыдущего.

3.2.3. Аппаратная реализация UART в микроконтроллере PIC16F887

В микроконтроллере PIC16F887 USART реализован в виде модуля EUSART (Enhanced Universal Synchronous Asynchronous Receiver Transmitter) – усовершенствованного универсального синхронного и асинхронного передатчика и приемника. Этот модуль содержит тактовые генераторы, сдвиговые регистры, буферы, необходимые для выполнения последовательной передачи (ввода или вывода) данных независимо от исполнения основной программы.

Устройство EUSART также известно как SCI (Serial Communications Interface) – последовательный коммуникационный интерфейс, который может быть настроен на полнодуплексный асинхронный или полудуплексный синхронный режимы работы. Полнодуплексный режим используется для взаимодействия с терминалами и персональными компьютерами. Полудуплексный синхронный режим используется для взаимодействия с такими периферийными устройствами, как ЦАП и АЦП, память EEPROM, или с другими микроконтроллерами. Эти устройства не имеют встроенного генератора тактирования передачи и требуют внешнего тактирования от ведущего устройства.

Модуль EUSART предоставляет следующие возможности: полнодуплексная асинхронная приемопередача; входной буфер на два символа; выходной буфер на один символ; программируемая длина символа в 8 или 9 бит; детектирование адреса в 9-битном режиме (полезно для сетей RS-485); обнаружение переполнения входного буфера; обнаружение нарушения кадра принимаемого символа; полудуплексный синхронный ведущий (англ. master); полудуплексный синхронный ведомый (англ. slave); программно настраиваемая полярность тактового сигнала в синхронном режиме; режим сна. Также модуль включает следующие возможности, делающие его оптимизированным для создания сетей LIN (Local Intervonnect Network – локальная сеть для коротких расстояний, промышленный стандарт в автомобилестроении): атоматическое обнаружение и подстройка тактовой частоты; пробуждение по команде Break; передача 13-битной команды Break.

Структурные схемы EUSART передатчика и приемника представлены на рис. 3.5, 3.6.

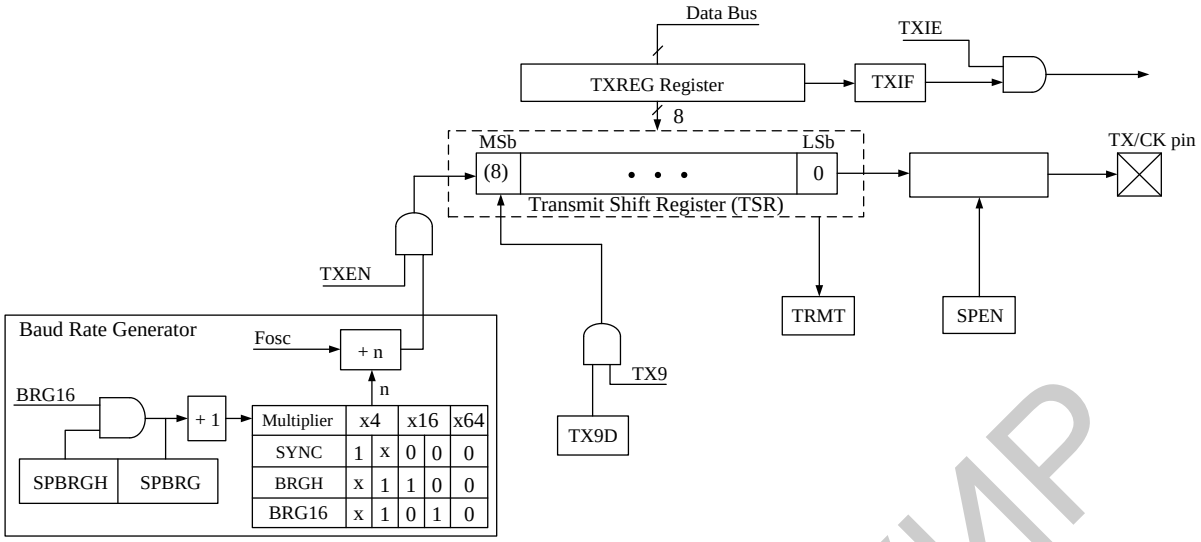


Рис. 3.5. Структурная схема передатчика EUSART

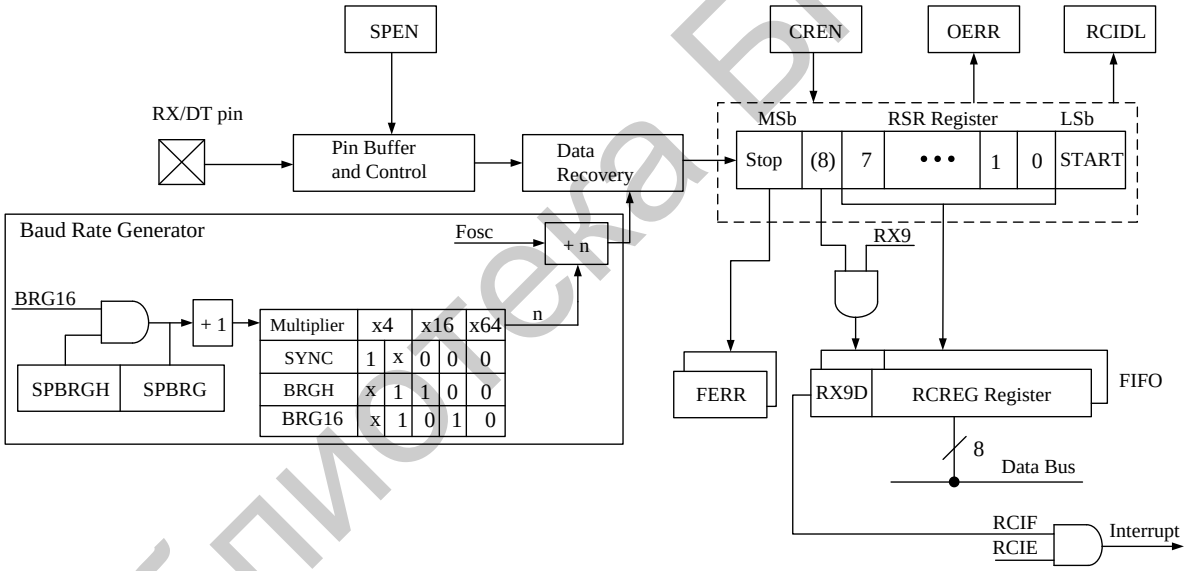


Рис. 3.6. Структурная схема приемника EUSART

Использование модуля EUSART конролируется тремя регистрами микроконтроллера: Transmit Status and Control (TXSTA) – отвечает за состояние передачи; Receive Status and Control (RCSTA) – отвечает за состояние приема; Baud Rate Control (BAUDCTL) – отвечает за частоту тактирования передачи данных.

В лабораторных работах используется только асинхронный режим передачи данных.

Модуль UART передает и принимает данные по стандарту NRZ (non-return-to-zero – не возвращаясь к нулю), что означает наличие двух логических состояний напряжения на линии данных: единицы (высокого уровня) и нуля

(низкого уровня). При этом состояние незанятой линии (между посылками, когда нет передачи данных, но линия подключена) соответствует единице.

Одна посылка данных состоит из стартового бита (нуля), 8 или 9 бит данных (наиболее распространен 8-битный формат) и одного или нескольких стоповых битов (единиц). Передача одного любого бита занимает время, равное 1/(частоту тактирования, бод). Встроенный 8-битный или 16-битный генератор тактирования позволяет получить стандартные тактовые частоты от общего тактирования микроконтроллера.

Передача данных осуществялется начиная с младшего бита. Передача и прием работают независимо друг от друга, но на основе общего формата посылок данных и частоты тактирования. Аппаратная реализация проверок четности отсутствует, но ее можно реализовать программно на основе 9-го бита данных.

Асинхронный передатчик EUSART. Блок передатчика, представленный на рис. 3.5, реализован на основе сдвигового регистра передачи TSR, который получает данные автоматически от буфера передачи, являющегося регистром TXREG. Регистр TSR не представлен в памяти данных и, соответсвенно, не доступен пользователю.

Работа передатчика в асинхронном режиме включается установкой трех битов TXEN = 1, SYNC = 0, SPEN = 1, все остальные управляющие биты остаются в состоянии по умолчанию. Бит TXEN регистра TXSTA разрешает передачу данных, бит SYNC регистра TXSTA включает асинхронный режим, бит SPEN регистра RCSTA автоматически настраивает ножку МК TX/CK I/O как выход. Если последняя используется как аналоговый вход-выход, аналоговый функционал надо отключить очисткой бита ANSEL. Когда бит SPEN установлен, ножка МК RX/DT I/O автоматически настраивается на вход, независимо от состояния соответствующего бита TRIS и отключенного передатчика EUSART. Данные с ножки RX/DT можно читать через обычный регистр PORT, но защелка регистра PORT не допустит вывода данных наружу.

Полный список регистров, связанных с асинхронной передачей, представлен в табл. 3.2

Таблица 3.2

Регистры, связанные с асинхронной передачей

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resents
BAUDCTL	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	01-0 0-00	01-0 0-00
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000x
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	-000 0000	-000 0000
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	-000 0000	-000 0000
RCREG	EUSART Receive Data Register								0000 0000	0000 0000
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x

Окончание табл. 3.2

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resents
SPBRG	BRG7	BRG6	BRG5	BRG4	BRG3	BRG2	BRG1	BRG0	0000 0000	0000 0000
SPBRGH	BRG15	BRG14	BRG13	BRG12	BRG11	BRG10	BRG9	BRG8	0000 0000	0000 0000
TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	1111 1111	1111 1111
TXREG	EUSART Transmit Data Register								0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	0000 0010	0000 0010

Примечание: «x» – неизвестно, «–» – не реализовано, читается как «0». Затененные серым ячейки не используются для асинхронной передачи.

Передача данных инициируется записью символа (минимального блока данных) в регистр TXREG. Если записанный символ – первый, или предыдущий был полностью выдвинут из сдвигового регистра TSR, данные из регистра TXREG немедленно передаются в регистр TSR. Если регистр TSR все еще продолжает выдачу битов предыдущего символа, новый символ задерживается в регистре TXREG до окончания передачи стопового бита предыдущего символа. В таком случае прямо за стоповым битом со стартового бита начнется передача следующего символа.

Флаг прерывания TXIF в регистре PIR1 устанавливается всегда, если включен передатчик и пуст буферный регистр TXREG. Бит TXIF очищается на второй процессорный такт после записи данных в TXREG, т. е. сразу после записи TXREG чтение флага даст неверный результат. Бит TXIF доступен только для чтения и не управляется программно.

Разрешение прерывания по флагу TXIF осуществляется установкой бита TXIE разрешения прерывания в регистре PIE1. Флаг TXIF устанавливается вне зависимости от того, разрешены или запрещены прерывания. Бит флага прерывания TXIF устанавливается автоматически при установке бита разрешения прерывания TXEN. Для реализации передачи через прерывания установите бит TXIE, только когда есть данные для передачи, и очистите его, когда последний символ записан в буфер TXREG.

Бит TRMT регистра TXSTA показвает состояние регистра TSR и доступен только для чтения. Бит TRMT устанавливается, когда регистр TSR пуст и очищается, когда в регистр передается новый символ из TXREG, и остается очищенным до выдвижения последнего бита символа. К этому биту не привязано прерываний, поэтому пользователю необходимо самостоятельно его проверять.

Временные диаграммы передачи данных в асинхронном режиме представлены на рис. 3.7 и 3.8

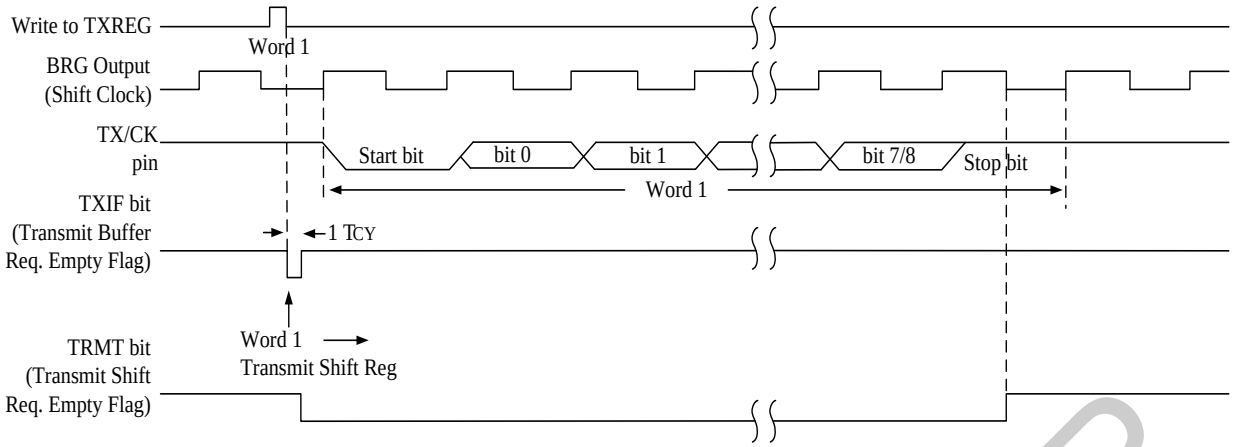
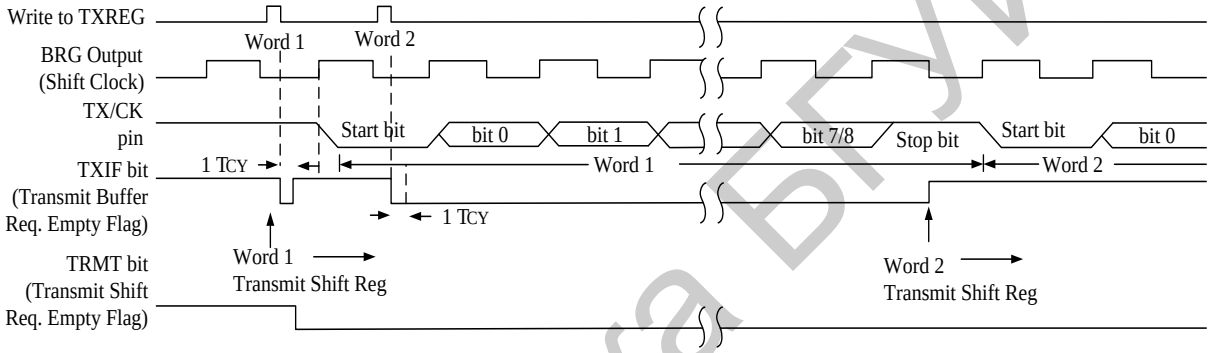


Рис. 3.7. Временная диаграмма асинхронной передачи по EUSART



Note: This timingdiagram shows two consecutive transmissions.

Рис. 3.8. Временная диаграмма асинхронной передачи по EUSART байт «один за одним»

Из рис. 3.2, 3.7, 3.8 видно, что датаграммы передачи по USART и RS-232 аналогичны, в то же время COM-порт предоставляет больше линий для служебных сигналов, без которых в большинстве случаев можно обойтись. Очевидно, что прямое подключение микроконтроллера к любому устройству по интерфейсу RS-232 невозможно из-за проблемы разных уровней напряжения. Со стороны микроконтроллера логическая единица составляет +5 В (+3,3 В), логический нуль 0 В, а со стороны COM-порта логическая единица может составлять до – 25 В, логический нуль до +25 В.

Для решения проблемы согласования уровней напряжения логических сигналов USART и RS-232 используются специальные микросхемы типа MAX232 и аналогов. В лабораторных макетах для этой цели служит микросхема MAX202, принципиальная схема включения которой приведена далее в п. 3.2.5, методика подключения описана ранее в п. 1.2.10.

Асинхронный приемник EUSART. Асинхронный режим обычно используется в системах, работающих по стандарту RS-232. Структурная схема приемника представлена на рис. 3.6. Данные поступают через ножку RX/DT микроконтроллера на блок восстановления данных, который по сути является высокоскоростным двигателем, работающим на частоте, в 16 раз превышающей частоту тактирования данных, после чего попадают в сдвиговый регистр

приема RSR, работающий на частоте тактирования данных. Когда все 8 или 9 бит символа поступят в RSR, они автоматически передадутся в двухсимвольную FIFO-память, которая позволяет принять два полных символа и начало третьего до того, как приемник EUSART будет необходимо программно обработать. Регистры RSR и FIFO недоступны программно, доступ к данным осуществляется через регистр RCREG.

Приемник EUSART настраивается на асинхронный прием при помощи трех следующих битов: CREN = 1, SYNC = 0, SPEN = 1, все остальные управляющие биты остаются в состоянии по умолчанию. Установка бита CREN в регистре RCSTA подключает приемник, очистка бита SYNC в регистре TXSTA включает асинхронный режим, установка бита SPEN в регистре RCSTA включает EUSART и автоматически настраивает ножку RX/DT как вход. Если RX/DT используется совместно с функциями аналогового ввода-вывода, их следует отключить очисткой соответствующего бита ANSEL. Когда бит SPEN установлен, ножка TX/CK автоматически настраивается как выход, вне зависимости от соответствующего бита TRIS и подключения передатчика EUSART. Защелка PORT отключена от выхода, поэтому ножку TX/CK невозможно использовать для общего вывода данных.

Полный список регистров, связанных с асинхронной передачей, представлен в табл. 3.3.

Таблица 3.3

Регистры, связанные с асинхронным приемом

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resents
BAUDCTL	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	01-0 0-00	01-0 0-00
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000x
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	-000 0000	-000 0000
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	-000 0000	-000 0000
RCSTA	SPEN								RX9	SREN
SPBRG	BRG7	BRG6	BRG5	BRG4	BRG3	BRG2	BRG1	BRG0	0000 0000	0000 0000
SPBRGH	BRG15	BRG14	BRG13	BRG12	BRG11	BRG10	BRG9	BRG8	0000 0000	0000 0000
TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	1111 1111	1111 1111
TXREG	EUSART Transmit Data Register								0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	0000 0010	0000 0010

Примечание: «x» – неизвестно, «–» – не реализовано, читается как «0». Затененные серым ячейки не используются для асинхронной передачи.

Цепь приема данных начинает прием символа по срезу первого бита, также известного как стартовый бит, всегда равный нулю (здесь для нулевых бит срез – начало, фронт – конец во времени). Цепь приема данных отсчитывает половину времени бита от среза стартового бита и проверяет, что он все еще равен нулю. Если бит уже не равен нулю, то прием символа отменяется, не вызывая ошибки, и продолжается поиск среза стартового бита.

Если проверка стартового бита успешна, цепь приема отсчитывает половину времени до середины следующего бита и передает его на мажоритарный детектор, с которого получает итоговое значение «0» или «1», поступающее в RSR. Эта операция повторяется пока все биты данных не будут помещены в RSR. Далее отсчитывается время последнего стопового бита и проверяется, что его уровень на линии приема соответствует единице, иначе генерируется ошибка формирования кадра, флаг которой очистится после приема следующего правильного кадра.

Сразу после получения всех битов данных и стопового бита принятый символ из RSR передается в FIFO и устананавливается флаг прерывания RCIF в регистре PIR1. Верхний символ, находящийся в FIFO, извлекается чтением регистра RCREG. Если FIFO переполнится, прием новых сиволов прекратится до тех пор, пока не будет очищено условие переполнения.

Бит флага прерывания RCIF в регистре PIR1 устанавливаетя всегда, когда включен приемник и есть непрочитанные символы в приемном FIFO. Флаг RCIF предназначен только для чтения и не может быть установлен или очищен программно. Прерывания от флага RCIF разрешаются при установке следующих битов: бит разрешения прерывания по приему RCIE в регистре PIE1, бит разрешения прерывания от периферии PEIE в регистре INTCON, бит глобального разрешения прерываний GIE в регистре INTCON. Флаг RCIF устанавливается, когда есть непрочитанные данные в FIFO, вне зависимости от установленных битов разрешения прерываний.

Каждый символ в приемном FIFO-буфере имеет собственный бит ошибки формирования кадра, означающей, что в должное время не был обнаружен стоповый бит. Статус бита ошибки формирования кадра для верхнего непрочитанного символа из приемного FIFO-буфера находится в бите FERR регистра RCSTA. Таким образом, бит FERR следует читать перед чтением RCREG и нет необходимости его очищать. Бит может быть очищен путем очищения бита SPEN в регистре RCSTA, что вызовет сброс модуля EUSART. Очистка бита CREN в регистре RCSTA не влияет на бит FERR. Ошибка формирования кадра не вызывает прерываний. Если все символы в приемном FIFO содержат ошибки формирования кадров, повторные чтения регистра RCREG не вызовут очистку бита FERR.

Приемный FIFO-буфер может вместить два символа. Ошибка переполнения возникает, когда третий символ полностью принят, до того как FIFO прочитан. В таком случае устанавливается бит OERR в регистре RCSTA. Символы, уже находящиеся в FIFO, могут быть прочитаны, но новых символов не будет принято до тех пор, пока бит ошибки не будет очищен. Бит ошибки должен быть очищен либо путем очистки бита CREN в регистре RCSTA, либо сбросом модуля EUSART путем очистки бита SPEN в регистре RCSTA.

Алгоритм настройки режима асинхронного приема (настройка асинхронной передачи осуществляется аналогично):

1. Инициализируйте регистры SPBRGH, SPBRG и биты BRGH и BRG16 для выбора желаемой частоты тактирования данных.

2. Включите последовательный порт путем установки бита SPEN. Бит SYNC должен быть очищен для асинхронного режима.

3. Если требуются прерывания, установите бит RCIE регистра PIE1 и биты GIE и PEIE регистра INTCON.

4. Если требуется 9-битный прием, установите бит RX9.

5. Включите приемник путем установки бита CREN.

6. Флаг прерывания RCIF будет установлен, когда принятый символ будет передан из RSR в приемный буфер. Если установкой бита RCIE было разрешено прерывание, оно будет сгенерировано.

7. Прочитайте регистр RCSTA для получения флагов ошибок. Для 9-битного режима прочитайте девятый бит.

8. Получите 8 младших бит данных из приемного буфера путем чтения регистра RCREG.

9. Если произошло переполнение, очистите флаг OERR путем очистки бита CREN, включающего приемник.

Временная диаграмма асинхронного приема представлена на рис. 3.9.

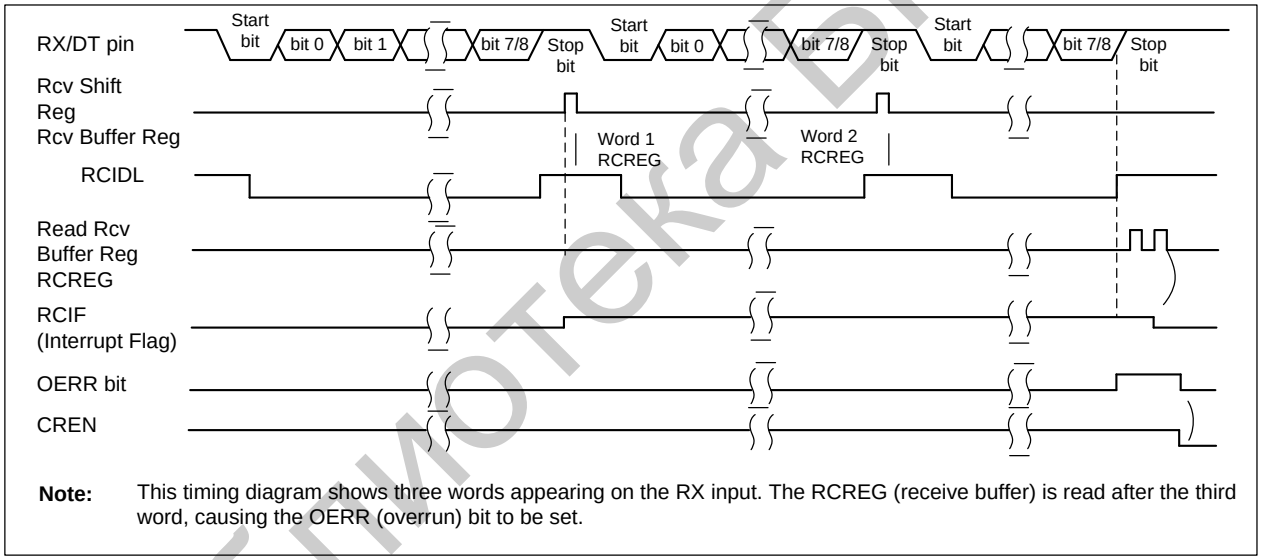


Рис. 3.9. Временная диаграмма асинхронного приема по EUSART

Настройка генератора тактирования данных. *Генератор тактирования данных (BRG)* является 8- или 16-битным таймером, поддерживающим как синхронный, так и асинхронный режимы работы модуля EUSART. По умолчанию генератор настроен на 8-битный режим. Установка бита BRG16 в регистре BAUDCTL переводит генератор в 16-битный режим.

Регистры SPBRGH, SPBRG определяют период свободно бегущего таймера BRG. В асинхронном режиме множитель таймера определяется двумя битами: битом BRGH регистра TXSTA и битом BRG16 регистра BAUDCTL. В синхронном режиме бит BRGH игнорируется. Полный список регистров, связанных с BRG, приведен в табл. 3.4.

Таблица 3.4

Регистры, связанные с установкой частоты тактирования данных

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resents
BAUDCTL	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	01-0 0-00	01-0 0-00
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
SPBRG	BRG7	BRG6	BRG5	BRG4	BRG3	BRG2	BRG1	BRG0	0000 0000	0000 0000
SPBRGH	BRG15	BRG14	BRG13	BRG12	BRG11	BRG10	BRG9	BRG8	0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	0000 0010	0000 0010

Примечание: «x» – неизвестно, «–» – не реализовано, читается как «0». Затененные серым ячейки не используются генератором.

В табл. 3.5 представлены формулы для расчета частоты тактирования данных. Пример, приведенный ниже, показывает, как рассчитать частоту и ошибку по частоте. Некоторые типичные частоты тактирования данных и соответствующие им параметры для частоты опорного генератора 8 МГц представлены в табл. 3.6. Для уменьшения ошибки по частоте в некоторых случаях может быть преимущество в использовании повышения частоты (бит BRGH = 1), или 16-битного генератора BRG (бит BRG16 = 1). Также 16-битный режим генератора используется для достижения низких частот тактирования данных при высоких частотах тактирования системы.

Запись нового значения в регистры SPBRGH или SPBRG вызывает перезапуск (очистку) таймера BRG. Таким образом, BRG не дожидается переполнения таймера перед переходом на новую частоту.

При изменении системного тактирования в процессе приема данных может возникнуть ошибка или потеря данных. Чтобы этого избежать, можно проверить статус бита RCIDL и убедиться, что приемник находится в бездействующем состоянии.

Таблица 3.5

Формулы для расчета скорости тактирования данных

Настроечные биты			Режим работы	Формулы расчета
SYNC	SYNC	SYNC		
0	0	0	8-bit/Asynchronous	$F_{OSC}/[64\ (n+1)]$
0	0	1	8-bit/Asynchronous	$F_{OSC}/[64\ (n+1)]$
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	$F_{OSC}/[64\ (n+1)]$
1	0	X	8-bit/Synchronous	
1	1	X	16-bit/Synchronous	

Примечание: «x» – неважно, «n» – значение пары регистров SPBRGH, SPBRG.

Пример расчета частоты генератора тактирования данных для устройства с частотой тактирования системы Fosc, равной 16 МГц, желаемой скорости передачи данных (Baud Rate, BR) 9600 бод, при асинхронном режиме и 8-битном таймере BRG.

Желаемая скорость передачи данных

$$BR = \frac{FOSC}{64([SPBRGH:SPBRG])+1}.$$

Решение для 16-битного значения пары SPBRGH:SPBRG

$$X = \frac{\frac{FOSC}{BR}}{64} - 1 = \frac{\frac{16\,000\,000}{9600}}{64} - 1 = [25{,}042] = 25.$$

Рассчитанная скорость передачи данных

$$BR = \frac{16\,000\,000}{64 \times (25+1)} = 9615.$$

Ошибка

$$ERR = \frac{9615 - 9600}{9600} = 0{,}16\,\%.$$

Таблица 3.6

Параметры некоторых распространенных частот тактирования данных

BAUD RATE	SYNC=0, BRGH=0, BRG16=0			SYNC=0, BRGH=1, BRG16=0			SYNC=0, BRGH=0, BRG16=1			SYNC=0, BRGH=1, BRG16=1 or SYNC=1, BRG16=1		
	Fosc=8.000 MHz			Fosc=8.000 MHz			Fosc=8.000 MHz			Fosc=8.000 MHz		
	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)
300	–	–	–	–	–	–	299,9	-0,02	1666	300,0	0,00	6666
1200	1202	0,16	103	–	–	–	1199	-0,08	416	1200	–0,02	1666
2400	2404	0,16	51	2404	0,16	207	2404	0,16	207	2401	0,04	832
9600	9615	0,16	12	9615	0,16	51	9615	0,16	51	9615	0,16	207
10 417	10 417	0,00	11	10 417	0,00	47	10 417	0,00	47	10417	0	191
19,2k	–	–	–	19 231	0,16	25	19,23k	0,16	25	19,23k	0,16	103
57,6k	–	–	–	55 556	-3,55	8	55 556	-3,55	8	57,14k	–0,79	34
115,2k	–	–	–	–	–	–	–	–	–	117,6k	2,12	16

Блок внутреннего источника тактирования INTOSC настраивается при производстве на заводе. Однако его частота может «уплывать» вследствие изменений опорного напряжения или температуры, что напрямую влияет на асинхронный режим работы. Для подстройки генератора тактирования данных можно использовать два метода, но оба требуют некоторого опорного тактового сигнала.

Первый метод (более предпочтительный) использует регистр OSCTUNE для подстройки выхода INTOSC. Добавочное значение в регистре OSCTUNE позволяет производить точную настройку системного источника тактирования.

Второй метод заключается в подстройке задающего значения в генераторе тактирования данных BRG. Это может быть сделано автоматически, при помощи функции автоподстройки частоты тактирования данных. Однако этот метод не позволяет достичь такой же точности управления частотой, как первый.

Более подробная информация по настройке тактирования, модулю EUSART и микроконтроллеру PIC16F887 приведена в [9].

3.2.4. Обработка прерываний в микроконтроллере PIC16F887

В микроконтроллере PIC16F887 для каждого периферийного модуля назначен отдельный источник прерываний, хотя некоторые модули, например UART, имеют несколько источников прерываний.

Есть следующие источники прерываний: внешнее прерывание на ножке RB0/INT, изменение уровня сигнала на входах внешнего порта (PORTB), изменение выходного уровня двух компараторов, переполнение таймеров (Timer0 или Timer1), совпадение таймера (Timer2), завершение преобразования от АЦП, завершение записи данных в EEPROM, прием или передача по EUSART (два прерывания), прерывание от модуля синхронного последовательного порта (MSSP), расширенный набор прерываний для модуля захвата и сравнения ШИМ (CCP), прерывание при остановке внешнего тактирования (FSCM), пробуждение из режима сверхнизкого энергопотребления.

Структурная схема логики прерываний представлена на рис. 3.10.

В микроконтроллере присутствует пять регистров управления прерываниями: INTCON (общий регистр), PIE1 и PIE2 (маски, чтобы разрешить/запретить прерывания), PIR1 и PIR2 (флаги прерываний, указывающие на возникшие прерывания).

Регистр управления прерываниями INTCON содержит индивидуальные биты флагов прерываний для ядра микроконтроллера, биты маски разрешения прерываний, а также бит глобального разрешения прерываний.

Если бит глобального разрешения прерываний GIE (INTCON<7>) установлен в «1», то разрешены все немаскированные прерывания. Все прерывания запрещены, если GIE (INTCON<7>) сброшен в «0». Прерывания индивидуально запрещены сбросом соответствующего бита в регистрах INTCON или PIE. При сбросе (reset) микроконтроллера бит GIE сбрасывается в «0».

Возврат из обработки прерываний выполняется по команде RETFIE, при этом происходит установка бита GIE в «1», что позволяет обработать любое отложенное прерывание.

При обработке прерываний бит GIE = 0, чтобы предотвратить повторную загрузку счетчика команд PC в стек и запись в PC адреса вектора прерываний 0004h. В обработчике прерываний источник прерываний может быть идентифицирован проверкой флагов прерываний. Как правило, флаги прерываний должны быть сброшены в обработчике прерываний перед разрешением прерываний в системе, чтобы предотвратить повторный переход на обработку прерываний. Индивидуальные флаги прерываний устанавливаются независимо от состояния бита общего разрешения прерываний GIE и соответствующих битов маски.

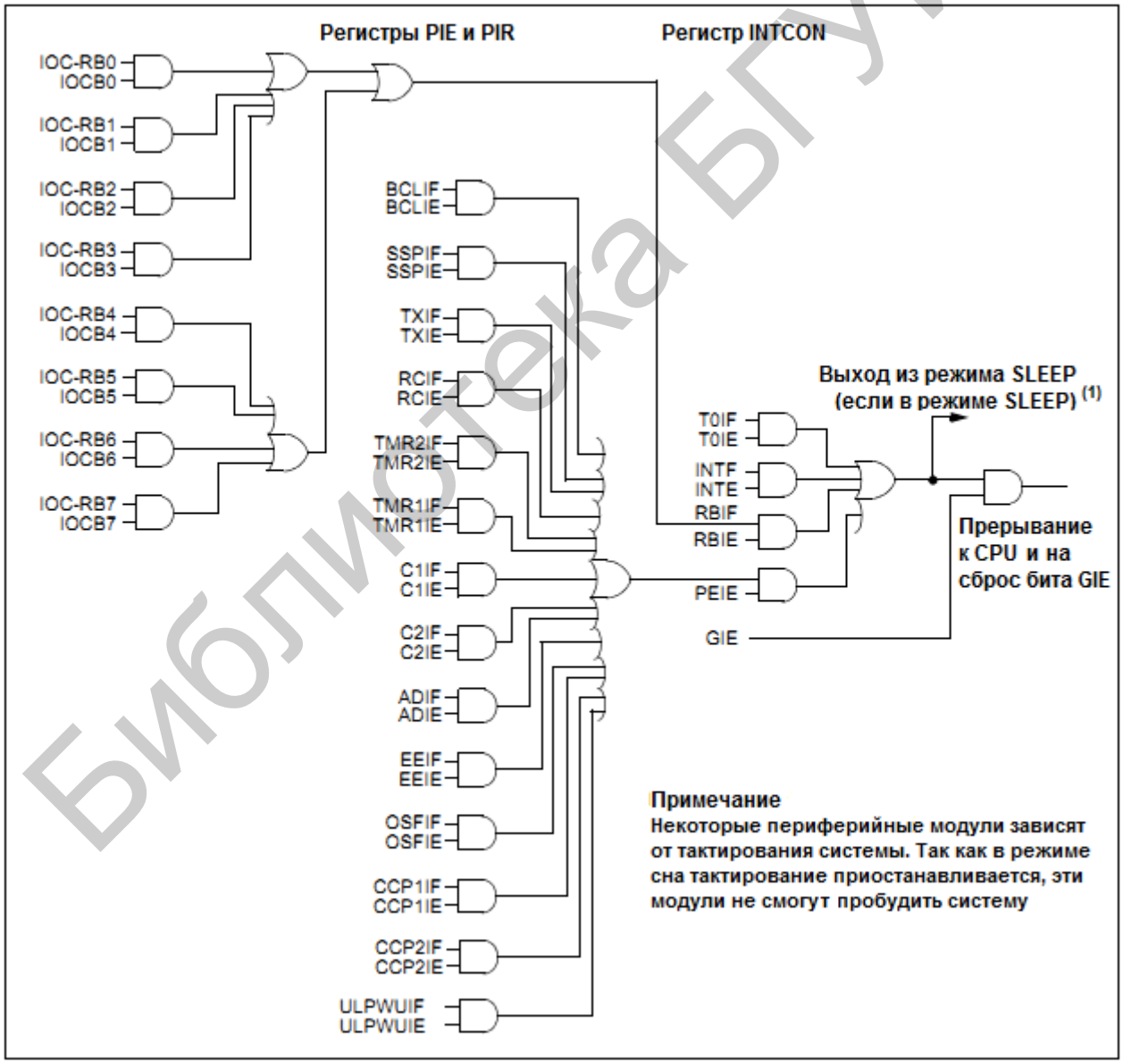


Рис. 3.10. Структурная схема логики обработки прерываний

При выполнении команды, сбрасывающей бит GIE в «0», любое прерывание, ожидающее выполнения в следующем машинном цикле, игнорируется. Микроконтроллер выполнит пустой цикл NOP в поле команды, сбрасывающей бит GIE в «0». Игнорированные прерывания ставятся в ожидание выполнения, пока бит GIE не будет установлен в «1».

Если алгоритм программы дает неправильные результаты, возможная причина заключается в неподходящей обработке прерывания.

При разрешенных прерываниях во время выполнения алгоритма необходимо гарантировать, что регистры, используемые алгоритмом, сохраняются и восстанавливаются в подпрограмме обработки прерываний. Проверьте подпрограмму обработки прерываний, так как некоторые регистры могут быть изменены.

Если выполнение программы прекращается, также возможная причина – неподходящая обработка прерывания.

Если в программе используются прерывания, то необходимо следить за тем, чтобы перед выходом из обработки прерываний (выполнения команды RETFIE) был сброшен флаг источника прерываний. Если флаг прерывания останется установленным, то после исполнения команды RETFIE выполнение программы опять перейдет по вектору прерываний и останется невыполненным разрешенное прерывание.

Пример программы обработки прерывания по изменению сигнала на порте PORTB приведен ниже. В программе при возникновении логической единицы на 4 выводе PORTB (нажатие кнопки) происходит установка на всех выводах PORTC логических единиц, а через 1 с – нулей (мигание светодиодов). Если нажатия кнопки не было более секунды, на трех младших выводах PORTC выполняется поочередный вывод логической единицы (бегающая лампочка).

```
int i;                          // Объявление переменной i типа integer
void interrupt ()               // Функция обработки прерываний
{
  if(INTCON.RBIF)               // Если прерывание произошло на выводах
  {                             // порта B, а именно если прерывание
    if(PORTB.F3==1)             // произошло на 4 выводе порта B, то
    {                           // на RC0-7 ставим 1
    PORTC=0b11111111;
    Delay_ms(1000);             // Пауза
    PORTC=~PORTC;               // Инвертируем выводы порта C
    }
    INTCON.RBIF = 0;            // Обнуляем флаг (завершаем прерывание)
  }
}
```

```
void main() {                    // Главная функция
   TRISB=0b11111111;             // RB0-7 на вход
   PORTB=0b00000000;
   TRISC=0b00000000;             // RC0-7 на выход
   PORTC=0b00000001;             // Задаем на выводах нули, кроме RC0 = 1
   INTCON=0b10001000;            // Разрешаем прерывания от PORTB и
                                 // глобально

   while(1){                     // Бесконечный цикл
      PORTC=0b00000001;          // На RC0 1, на остальных нули
      for(i=0;i<500;i++)asm{NOP}; // Пауза
      PORTC=0b00000010;          // На RC1 1, на остальных нули
      for(i=0;i<500;i++)asm{NOP}; // Пауза
      PORTC=0b00000100;          // На RC2 1, на остальных нули
      for(i=0;i<500;i++)asm{NOP}; // Пауза
   }
}
```

3.2.5. Методические рекомендации по настройке EasyPIC6

Принципиальная электрическая схема для данной лабораторной работы представлена на рис. 3.11. Общее описание методики настройки дано в п. 1.2.8. Более подробные сведения приведены в [7, 8].

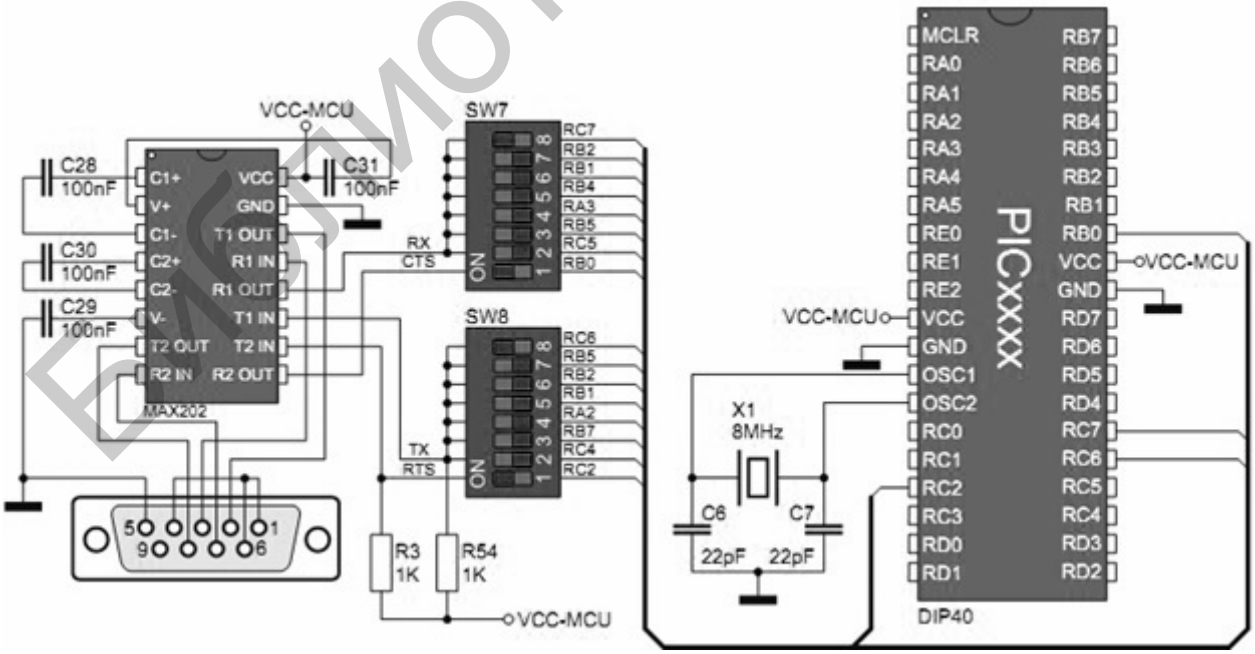


Рис. 3.11. Принципиальная электрическая схема подключения текстового ЖК-дисплея

3.2.6. Пример решения типового задания

Формулировка задания в примере: реализовать распознавание двух команд, приходящих на микроконтроллер по UART с компьютера, и текстовые ответы микроконтроллера по UART в моменты включения и получения команд при помощи встроенной библиотеки UART среды MicroC PRO for PIC.

Код программы:

```
char uart_rw_buffer;

void main()
{
   ADCON0 = 0x00;
   ANSEL  = 0;
   ANSELH = 0;
   C1ON_bit = 0;
   C2ON_bit = 0;

   TRISA=0;
   PORTA=0x0;

   // Инициализация модуля UART на скорость 9600 бит/с
   // и ожидание 100 мс для стабилизации настроек
   // с помощью библиотеки MikroE UART для PIC16 и PIC18
   UART1_Init(9600);
   Delay_ms(100);
   // передача текста через UART
   UART1_Write_Text("Ready \n\r");

   while (1)
   {
      // если получены данные по UART
      // выполняется распознавание и исполнение команды
      if (UART1_Data_Ready())
      {
         uart_rw_buffer = UART1_Read();
         if (uart_rw_buffer == 49)
         {
            PORTA=0x01;
            UART1_Write_Text("Command #1 Done \n\r");
         }
         else if (uart_rw_buffer == 50)
         {
            PORTA++;
            UART1_Write_Text("Command #2 Done \n\r");
```

```
            }
                    else
            {
               PORTA=0x00;
               UART1_Write_Text("Unknown Command \n\r");
            }
         }
      }
   }
```

3.3. Практическая часть

3.3.1. Контрольные вопросы

1. Что такое RS232, COM-порт и USART?
2. Какова структура и принцип передачи сообщений по UART?
3. Как выглядит принципиальная схема подключения микроконтроллера к ПК через аппаратный USART и зачем в ней микросхема MAX232?
4. Какова сруктура передатчика EUSART?
5. Как на уровне регистров и флагов работает передатчик?
6. Какова структура приемника EUSART?
7. Как на уровне регистров и флагов работает приемник?
8. Как использовать прерывания EUSART для приема и передачи?
9. Как рассчитать и задать частоту тактирования данных в EUSART?
10. Как работают прерывания в микроконтроллере?
11. Какие источники прерываний есть в PIC16F887?
12. Какова структура программы на языке С для обработки прерываний?

3.3.2. Варианты заданий

1. Настроить передачу данных через UART на скорости 9600 бод. Запрограммировать МК для приема сообщения в виде отдельных букв латинского алфавита и ответов порядковым номером этих букв.
2. Определить максимально возможную скорость передачи данных по UART между микроконтроллером и программой «Терминал» на компьютере.
3. Настроить передачу данных через UART на скорости 1200 бод. Запрограммировать МК для различения сообщений в виде двух символьных команд «C1» и «C2», на перую ответить «R1», на вторую не отчечать, а инвертировать состояние всех всободных портов МК.
4. Разработать фунцию инициализации UART на произвольную скорость передачи данных при произвольной частоте тактирования МК. Проверить работу функции для частоты тактирования 1 МГц и скорости 1200 бод.
5. Разработать функцию приема одиночных байтов по прерыванию от модуля EUSART. Настроить передачу данных через UART на скорости 9600 бод и про-

верить работу функции при помощи вывода принимаемых байтов на светодиоды PORTB.

6. Разработать функцию передачи одиночных байтов по прерыванию от модуля EUSART. Настроить передачу данных через UART на скорости 9600 бод и проверить работу функции при помощи ввода байтов через кнопки PORTB (исходные данные для передачи – все нажатия кнопок за последнюю секунду).

7. Разработать функцию приема строки символов. Проверить работу функции при помощи вывода принимаемых строк на текстовый дисплей.

8. Разработать функцию поиска в принимаемой строке символов. Проверить работу функции при помощи поиска фразы «PassWord», отображения приветствия на графическом дисплее в случае обнаружения и отображения отказа в доступе при других фразах.

9. Реализовать функции вывода на текстовый экран всех принятых символов между командами «C3» и «C4» и очистку экрана по команде «C5».

10. Реализовать функцию вывода на графический экран всех полученных символов (пределы 0–255) как данных столбчатой диаграммы (слева направо).

11. Реализовать текстовый арифметический («+, -, *, /») калькулятор.

12. Реализовать кодовые шестнадцатеричные часы (минуты, секунды) на светодиодах с командами «задать время» и «сообщить время» по UART.

13. Реализовать числовые часы (формат гггг-мм-дд-чч-мм-сс) на текстовом дисплее с командами «задать время» и «сообщить время» по UART.

14. Реализовать графические радиальные часы (минуты, часы) на графическом дисплее с командами «задать время» и «сообщить время» по UART.

15. Разработать робота-рисовальщика (тип «кисть») для графического дисплея. Управление роботом через UART.

16. Реализовать покадровое формирование изображений на графическом дисплее (128 × 64 точек) путем передачи по UART блоков из 1024 байт.

17. Разработать игру «Успей напечатать падающее слово» с использованием графического экрана для вывода слов и UART для ввода.

18. Разработать платформу для текстовых квестов (программа типа «меню» с навигацией посредством текстовых сообщений через UART-терминал).

19. Рализовать текстовый чат-бот UART на микроконтроллере.

20. Реализовать построитель графиков математических функций на графическом дисплее с вводом функций и параметров через UART.

Лабораторная работа №4

РЕАЛИЗАЦИЯ СИСТЕМЫ МОНИТОРИНГА ТЕМПЕРАТУРЫ

4.1. Цели работы

1. Изучение интерфейса и протокола 1-Wire.
2. Получение опыта применения и анализа интерфейса 1-Wire для обмена данными с цифровым термометром DS18B20.
3. Развитие практических навыков программирования микроконтроллеров семейства PIC и периферийных устройств путем разработки системы мониторинга температуры.

4.2. Теоретические сведения

4.2.1. Интерфейс 1-Wire

1-Wire (англ. «One Wire» – один провод) – полудуплексная линия связи.

Характеризуется невысокой скоростью (около 16 кбит/с) и значительной дальностью (до 300 м). Сеть устройств на базе 1-Wire называется MicroLan.

Полудуплексный режим передачи данных является двунаправленным, при этом одновременно возможна передача только в одном направлении.

Особенностью интерфейса является использование только одного провода (линии, порта контроллера) для передачи данных. При этом для передачи по 1-Wire также необходимо заземление, которое часто выступает второй линией. Питание может поступать как от линии данных (режим паразитного питания), так и от внешнего источника или по третьей линии.

Эта система передачи была введена фирмой Dallas Semiconductor и используется для низкоскоростных устройств вроде датчиков окружающей среды, например температуры, и электронных ключей, например «таблеток».

К линии данных подключается одно ведущее устройство (мастер) и одно или несколько ведомых. Линия данных в подключенном состоянии находится под напряжением (около +3 В, логическая единица), а сигналы по линии передаются путем ее заземления (сброс в логический нуль).

Связь по интерфейсу 1-Wire обычно осуществляется сеансами. Сеанс связи начинается с инициализации, затем происходит выбор ведомого устройства, после чего ведомое устройство готово принимать и выполнять функциональные (соответствующие функциям конкретного устройства) команды.

Инициализация выполняется мастером и представляет собой сброс (перезагрузку) всех ведомых устройств на линии. Для этого мастер заземляет линию данных минимум на 480 мкс, потом «отпускает» линию и ждет подтверждения присутствия данных от ведомых устройств в течение еще 480 мкс (минимум). Процесс инициализации и передачи данных показан на рис. 4.1.

Ведомое устройство, зафиксировав длительное заземление линии, перезагружается и, выждав 15-60 мкс, снова заземляет линию на 60-240 мкс, подтверждая тем самым свое присутствие на линии. После этого сеанс связи считается установленным. Завершается сеанс связи в любой момент времени (сбросом или отключением питания). При этом все несохраненные данные будут потеряны. Процедура инициализации изображена на рис. 4.1.

В течение сеанса связи данные передаются побитно, начиная с младшего бита младшего байта. Передача каждого бита инициируется мастером путем формирования тайм-слота (ТС) – отрезка времени для передачи бита. Длительность ТС составляет 60–120 мкс, пауза между ними минимум 1 мкс (это время также связано с восстановлением напряжения на линии).

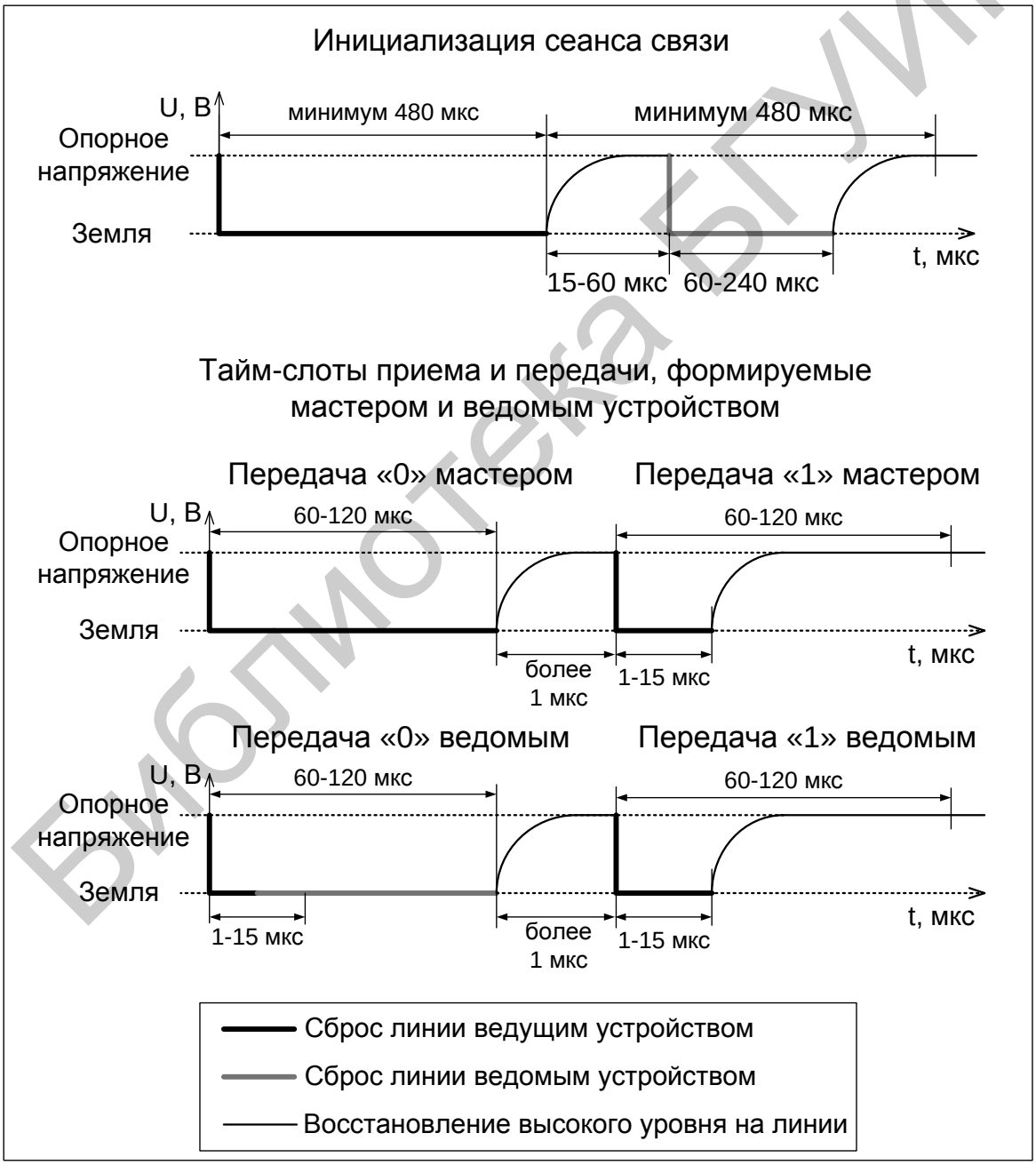


Рис. 4.1. Временные диаграммы напряжений в сеансе связи по интерфейсу 1-Wire

Начало ТС определяется мастером путем заземления линии данных на 1–15 мкс. Далее до конца ТС передается 1 бит, если это «0», то заземление линии поддерживается мастером или ведомым устройством, если это «1», то линия отпускается и на ней восстанавливается исходный уровень напряжения.

4.2.2. Цифровой термометр DS18B20

В качестве примера устройства, использующего интерфейс 1-Wire, рассмотрим цифровой термометр DS18B20, внешний вид которого представлен на рис. 4.2, структурная схема – на рис. 4.3.

***Некоторые особенности DS18B20*:**

– интерфейс 1-Wire, требующий только одну линию для передачи данных;

– уникальный 64-битный серийный номер, хранящийся во встроенной ROM;

– может быть запитан от линии данных; напряжение питания от 3,0 до 5,5 В;

– пределы измерения температуры от –55 до +125°; точность ±0,5° (от –10 до +85°);

– температурное разрешение от 9 до 12 бит при максимальном времени АЦП в 750 мс;

– контроль выхода температуры за заданные границы с командой поиска сработавших датчиков.

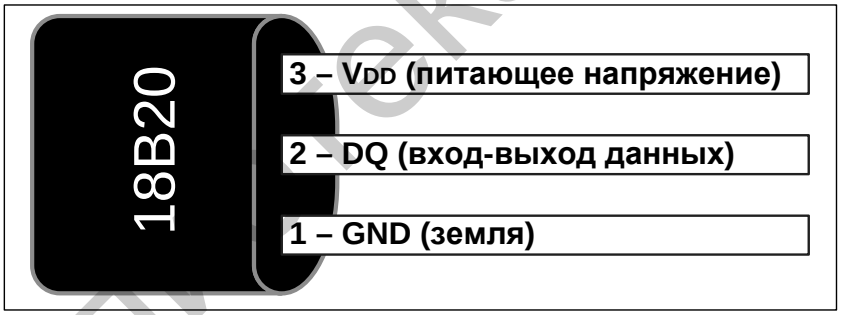


Рис. 4.2. Цифровой термометр DS18B20

Типы памяти DS18B20. Цифровой термометр оснащен памятью трех типов (структурная схема и карта памяти DS18B20 представлены на рис. 4.3, 4.4 соответственно).

1. Энергонезависимая память ROM, содержащая уникальный 64-битный идентификационный номер устройства.

2. Энергозависимая память Scratchpad, содержащая 9 байт оперативно используемой устройством информации.

3. Энергонезависимая EEPROM, дублирующая 3 байта из Scratchpad и возвращающая их при перезагрузке или по специальной команде.

Система команд DS18B20. Сеанс связи с устройством состоит из трех шагов, которые выполняются каждый раз, когда требуется обратиться к устройству.

1. Инициализация (перезагрузка и подтверждение присутствия).
2. ROM-команда (выбор устройства через уникальный 64-битный номер).
3. Функциональная команда (обращение к какой-либо из функций устройства, выполняется один раз за сеанс связи).

Каждая команда представляет собой один байт. Передача одного байта занимает восемь тайм-слотов. После команды мастеру обычно требуется продолжить формирование ТС для передачи/приема данных. Список команд DS18B20 представлен в табл. 4.1. Подробное описание термодатчика дано в [14].

Таблица 4.1

Список команд цифрового термометра DS18B20

Название команды	**Описание команды**	**HEX-код команды**	**Активность шины после команды**
ROM-команды			
Search ROM	Запрашивает уникальные ROM-номера всех устройств	F0	Используется спец. алгоритм
Read ROM	Запрашивает уникальный ROM-номер (только если на линии единственное устройство)	33	DS18B20 передает 8 байт ROM-номера
Match ROM	Устанавливает сеанс связи по точному ROM-номеру	55	мастер передает 8 байт ROM
Skip ROM	Устанавливает сеанс связи со всеми устройствами на линии	CC	Нет
Alarm Search	Запрашивает уникальные ROM-номера всех устройств, у которых превышены пределы температуры	C	Используется специальный алгоритм
Функциональные команды определения температуры			
Convert T	Инициирует измерение/преобразование температуры	44	DS18B20 передает статус выполнения
Функциональные команды работы с памятью			
Read Scratchpad	Читает Scratchpad, включая байт CRC	BE	DS18B20 передает 9 байт
Write Scratchpad	Записывает в Scratchpad байты №2, 3, 4	4E	Мастер передает 3 байта
Copy Scratchpad	Копирует байты №2, 3, 4 из Scratchpad в EEPROM	48	Нет
Recall E^2	Копирует байты №2, 3, 4 из EEPROM в Scratchpad	B8	DS18B20 передает статус
Read Power Supply	Запрашивает режим опорного напряжения (паразитное/нет)	B4	DS18B20 передает статус

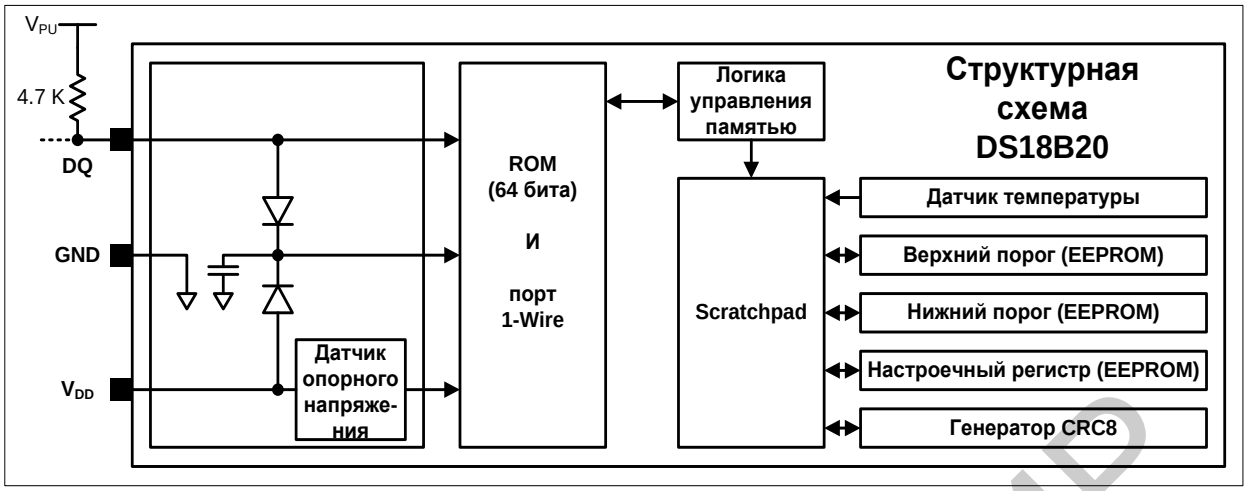


Рис. 4.3. Структурная схема цифрового термометра DS18B20

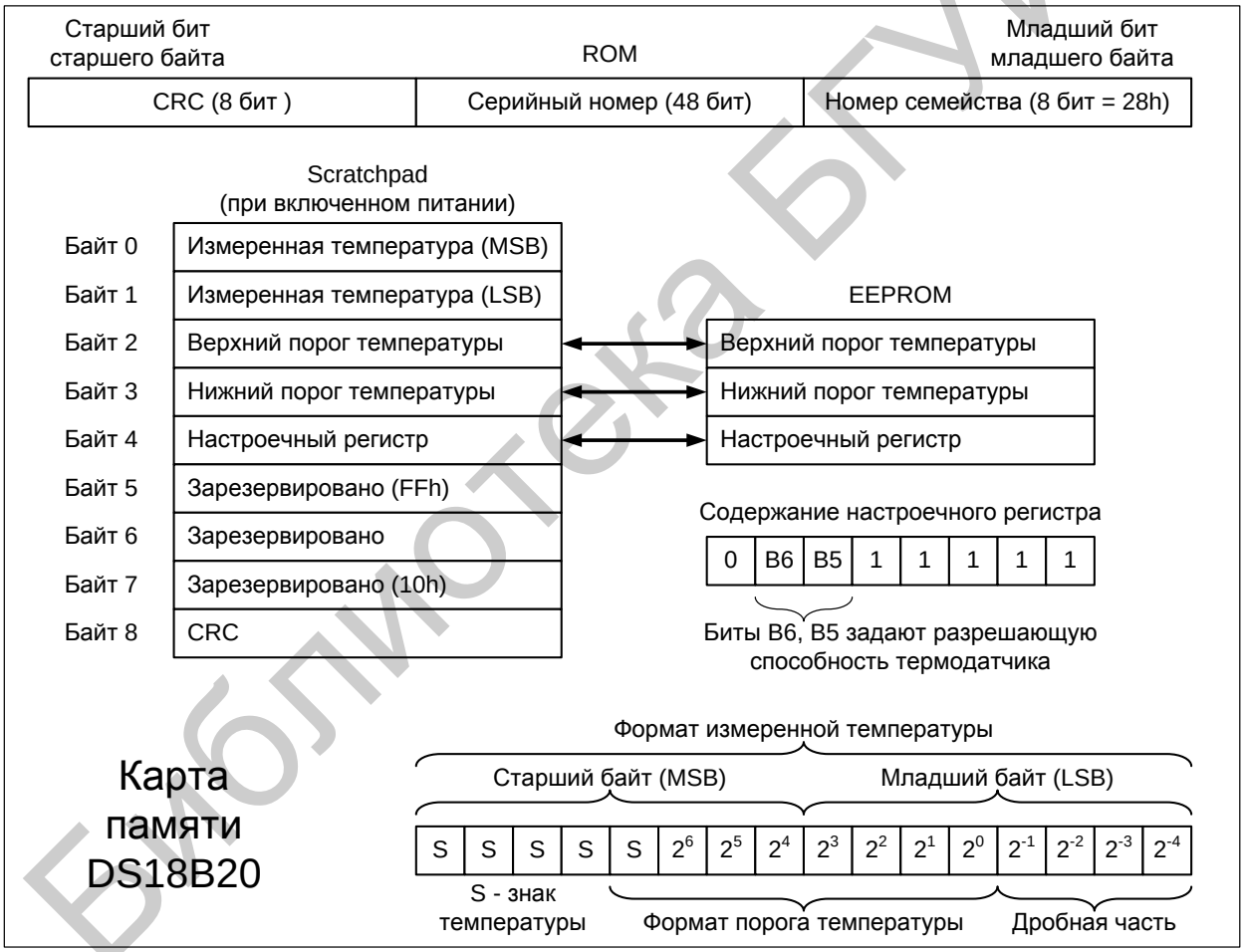


Рис. 4.4. Карта памяти цифрового термометра DS18B20

Описание специального алгоритма поиска/выбора устройства по его ROM. После команды будет произведен выбор одного устройства за утроенное (относительно размерности ROM) количество ТС.

Каждое устройство на линии первым слотом выдает первый бит своей ROM, вторым слотом выдает инвертированное значение первого бита своей ROM.

Очевидно, что если есть на линии два устройства с разными первыми битами, то в обоих ТС будет передано 0. Третьим тайм-слотом мастер подтверждает первый бит выбираемого устройства. Все устройства, чей первый бит отличается от переданного мастером, не отвечают до следующей перезагрузки.

Далее аналогичным образом определяются второй и все последующие биты.

4.2.3. Библиотека для работы с 1-Wire

Программная среда microCPROforPIC содержит набор встроенных библиотек, в том числе и для работы с интерфейсом 1-Wire. Недостатком этих библиотек является их закрытость, поэтому в учебных целях для интерфейса 1-Wire будем использовать библиотеку OneWire_Lib.h.

Описание библиотеки OneWire_Lib.h и работы с ней

1. Библиотеку можно подключить, поместив ее в папку проекта microC и добавив в код программы строчку #include «OneWire_Lib.h» до использования функций из этой библиотеки.

2. Библиотека содержит функции для работы с интерфейсом 1-Wire и цифровым термометром DS18B20, а именно:

– OneWire_Byte_Tx() (передает один байт, указанный в скобках, по интерфейсу 1-Wire; ничего не возвращает);

– OneWire_Byte_Rx(void) (принимает один байт по интерфейсу 1-Wire; возвращает принятый байт);

– OneWire_Reset(void) (осуществляет инициализацию интерфейса 1-Wire; возвращает «1» если устройств на линии не обнаружено, иначе «0»);

– DS_Read(void) (читает оцифрованное значение температуры с датчика DS18B20 и возвращает указатель на массив (строку) символов ASCII, описывающих измеренную температуру).

Первые три функции представляют собой базис для удобной работы с интерфейсом 1-Wire. Для получения значения температуры с датчика DS18B20 достаточно воспользоваться только последней функцией и передать возвращаемый ею указатель в какую-нибудь функцию обработки текста (например, отобразить на текстовом LCD или переслать по UART).

3. Работа с функциями библиотеки требует назначения соответствующих портов микроконтроллера для передачи данных. Обычно для повышения переносимости кода все настройки реализуют с помощью макросов языка microC. Вариант настроек, подходящий для используемой отладочной платы (его можно также найти в библиотеке), включен в листинг 1.

4.2.4. Методические рекомендации по настройке EasyPIC6

Для выполнения данной лабораторной работы следует путем установки правильной конфигурации переключателей и перемычек на макете сформировать подключение узлов, соответствующее принципиальной электрической схеме на рис. 4.5. Общее описание методики настройки дано в п. 1.2.8. Более подробные сведения приведены в [7, 8].

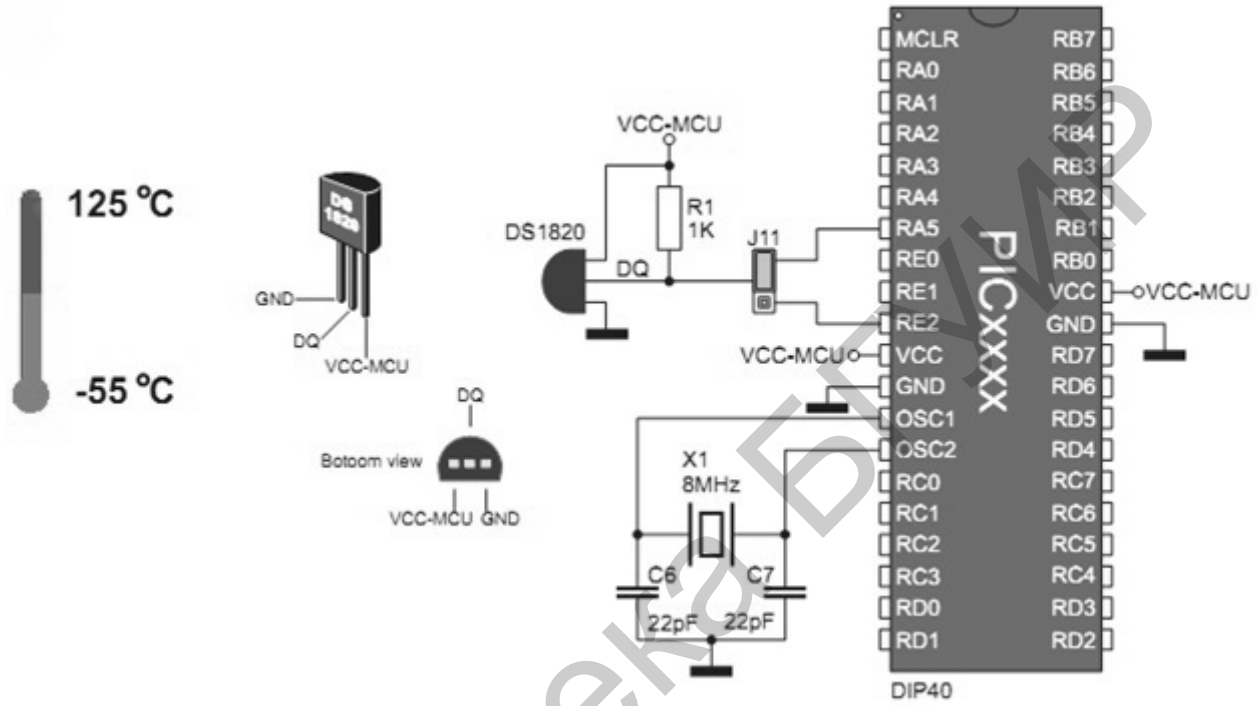


Рис. 4.5. Принципиальная электрическая схема подключения цифрового термометра

4.2.5. Пример решения типового задания

Формулировка задания в примере: реализовать отображение на текстовом дисплее в реальном времени значений температуры с цифрового термометра DS18B20, получаемых по интерфейсу 1-Wire, при помощи открытых библиотек «LCD_Lib» и «OneWire_lib».

Код программы:

```
// Настройки подключения датчика к порту через объявление макроса
#define DS_TRIS TRISA          // 8-битный регистр направлений порта A
#define DS_PORT PORTA          // 8-битный регистр значений порта A
#define DS 5                   // Номер используемой ножки порта

// подключение файлов библиотек, находящихся в директории с проектом
#include "LCD_Lib_v03.h"       // Библиотека для работы с LCD
#include "GLCD_Screen_v05_Settings.h"// Настройки для работы с GLCD
#include "GLCD_Screen_v05.h"   // Библиотека для работы с GLCD
    #include "OneWire_lib_v12.h"   // Библиотека для работы с 1-Wire
    void main() {
```

```
// Настройки для PIC16F887
ADCON0 = 0x00;                  // Отключение АЦП
ANSEL = 0;                      // Настройка пинов как
ANSELH = 0;                     // цифровых входов/выходов
C1ON_bit = 0;                   // Отключение встроенных аналоговых
C2ON_bit = 0;                   // компараторов (устройств сравнения)

// Настройки для PIC18F4550
ADCON1 = 0x0F;                  // Настройка пинов как цифровых
                                // входов/выходов
CMCON = 7;                      // Отключение встроенных аналоговых
                                // компараторов (устройств сравнения)

LCD_Init();                     // Инициализация LCD
Lcd_Goto(1,2);                  // Курсор ко второму символу первой
                                // строки
Lcd_Send_Text("Temperature:"); // Отображение текста

while(1){                       // Бесконечный цикл
Lcd_Goto(2,3);                  // Курсор к третьему символу второй
                                // строки
LCD_Send_Text(DS_Read());  // Проверка и отображение температуры
}
```

4.3. Практическая часть

4.3.1. Контрольные вопросы

1. Что такое интерфейс 1-Wire и какие у него сособенности?
2. Какова принципиальная электрическая схема сети на базе 1-Wire?
3. Как выглядят временные диаграммы работы интерфейса 1-Wire?
4. Каков алгоритм сеанса связи и выбора устройств на шине 1-Wire?
5. Каково назначение и особенности датчика DS18B20?
6. Как выглядит структура DS18B20 и организация памяти в нем?
7. Какова система команд DS18B20 и принцип использования?
8. Какие функции есть в библиотеке термодатчика и как их использовать?
9. Как могут выглядеть обобщенная структурная и принципиальная схемы системы, объединяющей все четыре лабораторные работы?
10. Каковы базовые принципы проектирования встраиваемых систем?
11. Каковы основные принципы использования аппаратных блоков МК?
12. Каковы принципы выбора и применения внешних модулей и интерфейсов при разработке встраиваемых вычислительных систем?

4.3.2. Варианты заданий

1. Реализовать автоматический сбор данных о текущем значении температуры с отображением на текстовом/графическом ЖК-дисплее с ежесекундным обновлением данных в режиме реального времени.

2. Реализовать автоматический сбор данных о текущем значении градиента (разницы между двумя датчиками) температуры с отображением на текстовом/графическом ЖК-дисплее с ежесекундным обновлением данных в режиме реального времени.

3. Реализовать автоматический расчет среднего значения температуры за 5 мин с отображением на текстовом/графическом ЖК-дисплее с ежесекундным обновлением данных в режиме реального времени.

4. Реализовать автоматический расчет среднего значения градиента (разницы между двумя датчиками) температуры за 5 мин с отображением на текстовом/графическом ЖК-дисплее с ежесекундным обновлением данных в режиме реального времени.

5. Вывести на графический ЖК-дисплей график зависимости температуры от времени (в пределах от 10 до 40 °С и 100 с) с ежесекундным обновлением данных в режиме реального времени.

6. Вывести на графический ЖК-дисплей график градиента (разницы между двумя датчиками) температуры от времени (в пределах от 10 до 40 °С и 100 с) с ежесекундным обновлением данных в режиме реального времени.

7. Реализовать систему автоматического оповещения пользователя ПК о текущем значении температуры через UART и программу «Терминал». Обновление данных производится автоматически в режиме реального времени раз в 5 с.

8. Реализовать систему автоматического оповещения пользователя ПК через UART и программу «Терминал» о превышении порога температуры, используя аппаратные возможности цифрового термометра.

9. Реализовать систему автоматического оповещения пользователя ПК через UART и программу «Терминал» о превышении порога температуры программным образом через память микроконтроллера.

10. Реализовать систему автоматического (раз в секунду) запроса уникального номера термодатчика с отображением на текстовом/графическом ЖК-дисплее.

11. Реализовать систему автоматического запроса уникальных номеров всех устройств на шине 1-Wire с отображением на текстовом/графическом ЖК-дисплее с автоматической прокруткой списка (на одну строку раз в 5 с).

12. Реализовать систему автоматизированного (по команде) запроса уникальных номеров всех устройств на шине 1-Wire с получением команды и отправкой результата по интерфейсу UART через программу «Терминал».

13. Реализовать систему автоматизированного (по различению специальных команд) извещения пользователя ПК через UART и программу «Терминал» о текущем значении температуры и среднем значении за прошедший час с добавлением к каждому ответу уникального номера термодатчика.

14. Реализовать систему автоматизированного (по различению специальных команд) извещения пользователя ПК через UART и программу «Терминал» об уникальном номере термодатчика, последнем измеренном значении температуры, обновлении (регистрации) текущего значения температуры.

15. Реализовать генератор случайных чисел на основе флуктуаций температуры с ежесекундным выбором чисел из диапазона 0 до 255. Числа и текущее значение температуры отобразить на текстовом ЖК-дисплее.

16. Реализовать автоматическое построение закона распределения случайной величины в диапазоне от 0 до 255, полученной из флуктуаций температуры, на графическом ЖК-дисплее.

17. Узнать все текущие настройки термодатчика (состояние настроечного регистра, режим 2) и сообщить их пользователю через UART и программу «Терминал» в текстовом виде.

18. Обнаружить сработавший по превышению порога температуры датчик и перенастроить его разрешающую способность.

19. Реализовать генератор CRC8 (полином $X^8+X^5+X^4+1$) и сравнить с генератором CRC-термодатчика по результатам работы.

20. Реализовать на графическом ЖК-дисплее изображение объекта, движущегося по случайной траектории, вычисляемой на основе флуктуаций температуры.

ЛИТЕРАТУРА

1. Milan Verle. PIC Microcontrollers – Programming in C. / Milan Verle // MikroElektronika, 2009. – 336 p.

2. Цифровые устройства. Лабораторный практикум : учеб.-метод. пособие / Р. Г. Ходасевич [и др.]. – Минск : БГУИР, 2010. – 112 с.

3. Левкович, В. Н. Микропроцессорные устройства. Лабораторный практикум : учеб.-метод. пособие / В. Н. Левкович, Е. Н. Каленкович, А. А. Казека. – Минск : БГУИР, 2012. – 92 с.

4. MikroC User Manual [Электронный ресурс]. – 2015. – Режим доступа : http://www.mikroe.com/pdf/mikroc/mikroc_manual.pdf.

5. MikroC PRO for PIC User Manual [Электронный ресурс]. – 2015. – Режим доступа : http://www.mikroe.com/downloads/get/30/mikroc_pic_pro_manual_v100.pdf.

6. MikroICD User Manual [Электронный ресурс]. – 2015. – Режим доступа : http://www.mikroe.com/pdf/mikroicd_manual.pdf.

7. EasyPIC6 User Manual [Электронный ресурс]. – 2015. – Режим доступа : http://ww1.microchip.com/downloads/en/DeviceDoc/easypic6_manual_v100.pdf.

8. EasyPIC6 Schematic [Электронный ресурс]. – 2015. – Режим доступа : http://www.mikroe.com/pdf/easypic6/easypic6_schematic_v101.pdf.

9. PIC16F887 Data Sheet [Электронный ресурс]. – 2012. – Режим доступа : http://ww1.microchip.com/downloads/en/DeviceDoc/41291G.pdf.

10. Документация по микроконтроллерам PIC. Переводы на русский язык [Электронный ресурс]. – 2015. – Режим доступа : http://www.microchip.ru-/lit/?mid=1x0.

11. KS0066 Data Sheet [Электронный ресурс]. – 2015. – Режим доступа : http://pdf.datasheetcatalog.com/datasheets2/55/553815_1.pdf.

12. KS0107 Data Sheet [Электронный ресурс]. – 2015. – Режим доступа : http://pdf.datasheetcatalog.com/datasheets2/16/160893_1.pdf.

13. MAX202 Data Sheet [Электронный ресурс]. – 2015. – Режим доступа : http://www.ti.com/lit/ds/symlink/max202.pdf.

14. DS18B20 Data Sheet [Электронный ресурс]. – 2015. – Режим доступа : http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

Св. план 2015, поз. 40

Учебное издание

Васюкевич Сергей Юрьевич
Давыдов Игорь Геннадьевич
Цурко Александр Владимирович

ВЫЧИСЛИТЕЛЬНЫЕ СРЕДСТВА РАДИОСИСТЕМ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. И. Герман*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *А. В. Бас*

Подписано в печать 13.07.2016. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 7,56. Уч.-изд. л. 8,0. Тираж 50 экз. Заказ 146.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6