В.Н. Хилинский

Учимся программировать микроконтроллеры PIC на языке PicBasicPro

MicroCode Studio - программная среда для разработки и отладки программ на языке BASIC.

Примеры программ.

Справочник по командам PicBasic Pro.

Уфа 2007

Содержание

Предисловие	7
Введение	8
Глава 1. PIC-микроконтроллеры	10
Питание микроконтроллеров	10
Сброс и перезапуск микроконтроллеров	12
Регистры конфигурации	14
Генераторы тактовых импульсов	14
Порты ввода/вывода и регистры управления передачей данных	16
Глава 2. Компилятор PicBasicPro	17
Интерактивная среда разработки (IDE) - Microcode Studio Plus. Установка и настройка	18
Интерактивная отладка программ (ICD)	31
Программа MicroCode Loader	34
Терминал последовательного порта (The Serial Communicator)	34
Программа Easy HID Wizard	36
Язык PicBasicPro	36
Основные понятия и определения	36
Идентификаторы	36
Комментарии	37
Переменные	37
Псевдонимы	38
Константы	39
Символы	40
Метки	40
Массивы	41
Математические операторы	42
Умножение	44
Деление	44
Операторы сдвига	45
Оператор **ABS**	45
Оператор **COS**	45
Оператор **DCD**	46
Оператор **DIG**	46
Оператор **DIV32**	46
Оператор **MAX и MIN**	47
Оператор **NCD**	48
Оператор **REV**	48
Оператор **SIN**	48

Оператор **SQR** 49
Поразрядные операторы 49
Операторы сравнения 49
Логические операторы 50
Некоторые замечания о стилях программирования 51

Глава 3. Примеры программ 52
Пример № 1 – Мигающий светодиод (Hello Word) 52
Пример № 2. Работа с несколькими светодиодами 54
Пример № 3. Взаимодействие с кнопкой 58
Пример № 4. Аналого-цифровое преобразование 60
Пример № 5. Управление сервомотором 64
Пример № 6. Управление ЖКИ 66
Пример № 7. Подключение 7 сегментного светодиодного индикатора 71
Пример № 8. Работа с внутренней ЕЕПРОМ 74
Пример № 9. Работа с внешней ЕЕПРОМ 76
Пример № 10. Работа с 12 кнопочной клавиатурой 79
Пример № 11. Создание музыки 83

Глава 4. Команды компилятора PicBasicPro 86
Команда **@** 88
Команда **ADCIN** 89
Команда **ASM…ENDASM** 89
Команда **BRANCH** 90
Команда **BRANCHL** 90
Команда **BUTTON** 91
Команда **CALL** 97
Команда **CLEAR** 97
Команда **CLEARWDT** 97
Команда **COUNT** 97
Команда **DATA** 98
Команда **DEBUG** 99
Команда **DEBUGIN** 100
Команда **DISABLE** 100
Команда **DISABLE DEBUG** 101
Команда **DISABLE INTERRUPT** 101
Команда **DTMFOUT** 101
Команда **EEPROM** 102
Команда **ENABLE** 103
Команда **ENABLE DEBUG** 103
Команда **ENABLE INTERRUPT** 103
Команда **END** 104

Команда **ERASECODE** 104
Команда **FOR…NEXT** 104
Команда **FREQOUT** 105
Команда **GOSUB** 106
Команда **GOTO** 107
Команда **HIGH** 107
Команда **HPWM** 108
Команда **HSERIN** 109
Команда **HSEROUT** 112
Команда **HSERIN2** 113
Команда **HSEROUT2** 114
Команда **I2CREAD** 116
Команда **I2CWRITE** 118
Команда **IF…THEN** 121
Команда **INPUT** 121
Команда **LET** 122
Команда **LCDIN** 122
Команда **LCDOUT** 123
Команда **LOOKDOWN** 125
Команда **LOOKDOWN2** 126
Команда **LOOKUP** 127
Команда **LOOKUP2** 128
Команда **LOW** 128
Команда **NAP** 129
Команда **ON INTERRUPT** 130
Команда **OUTPUT** 131
Команда **OWIN** 131
Команда **OWOUT** 132
Команда **PAUSE** 133
Команда **PAUSEUS** 133
Команда **PEEK** 135
Команда **PEEKCODE** 135
Команда **POKE** 135
Команда **POKECODE** 136
Команда **POT** 136
Команда **PULSIN** 138
Команда **PULSOUT** 138
Команда **PWM** 139
Команда **RANDOM** 140
Команда **RCTIME** 140
Команда **READ** 141
Команда **READCODE** 141
Команда **REPEAT…UNTIL** 141

Команда **RESUME** 142
Команда **RETURN** 142
Команда **REVERSE** 142
Команда **SERIN** 143
Команда **SELECT…CASE** 145
Команда **SERIN2** 145
Команда **SEROUT** 149
Команда **SEROUT2** 151
Команда **SHIFTIN** 155
Команда **SHIFTOUT** 157
Команда **SLEEP** 158
Команда **SOUND** 159
Команда **STOP** 160
Команда **SWAP** 160
Команда **TOGGLE** 160
Команда **WHILE…WEND** 161
Команда **USBINIT** 161
Команда **USBIN** 161
Команда **USBOUT** 162
Команда **WRITE** 162
Команда **WRITECODE** 163
Команда **XIN** 163
Команда **XOUT** 164
Оператор **DEFINE** 166

Некоторые замечания по поводу обозначения выводов микроконтроллеров 169

Заключение 171

Приложение 1. Однопроводный интерфейс или интерфейс “Micro-LAN”. 172

Приложение 2. Протокол X10 180

Подборка ссылок в INTERNET 185

Литература 186

Предисловие

Написать эту книгу меня побудило то обстоятельство, что до сегодняшнего дня в нашей стране еще не было выпущено книг, рассказывающих о программировании микроконтроллеров на языке BASIC. Хотя в Интернете, да и не только в нем многие интересуются этим.

Я сам занимаюсь электроникой довольно давно. За это время в этой отрасли техники произошли значительные изменения. Я начинал работать тогда, когда широко использовались транзисторы, пришедшие на смену электронным лампам. А на смену транзисторам уже начинали приходить микросхемы. Что в это время происходило за пределами СССР в области электронной техники, нам было неведомо (к большому нашему сожалению).

Внедрение в производство первых микро компьютеров в нашей стране началось в 80 годы прошлого века. Первые промышленные компьютеры были очень дорогими для обычного человека. Поэтому большинство радиолюбителей собирало на работе и дома самодельные компьютеры такие как «Специалист», «Радио 86 РК» и т.п. Вершиной радиолюбительских сборок стали компьютеры серии «Sinkler» на базе процессора Z80.

Играя на этих компьютерах и пытаясь создавать свои программы, любители начинали самостоятельно осваивать наиболее распространенный для этих компьютеров язык программирования - BASIC.

Эта книга предназначена для студентов и инженеров, занимающихся электроникой. Здесь они смогут найти ответы на вопрос, как и чем, программировать PIC-микроконтроллеры.

Книга разбита на три части. В первой части приводится очень краткое описание устройства и работы PIC-микроконтроллеров.

Во второй части описывается язык программирования PicBasicPro.

Третья часть содержит подборку практических примеров устройств на основе микроконтроллеров PIC, а также тексты Basic-программ к этим примерам.

В четвертой части приведены справочные данные по командам языка PicBasicPro.

Благодарность.

Хочу выразить искреннюю признательность своему коллеге Виктору Никитовичу Овчинникову, прочитавшему рукопись и высказавшему ряд важных замечаний.

Также хочу поблагодарить свою супругу Ирину за детей, которых она подарила мне и за помощь в коррекции рукописи и ее издании.

Введение

С появлением в электронике микроконтроллеров многие столкнулись с проблемой, как и чем их программировать. В этой книге я не буду касаться вопроса устройства микроконтроллеров, их параметров и т. п., так как эти вопросы неплохо разъяснены во многих книгах, изданных в нашей стране. Здесь же мы будем рассматривать вопросы программирования микроконтроллеров, производимых фирмой Microchip так называемых PIC-контроллеров или на радиолюбительском сленге - «пикушек», на языке BASIC.

Известно, что программы, написанные на языке Ассемблер, дают лучшие результаты. Размер программы минимален и выполняется она значительно быстрее, чем написанная на любом другом языке. Но и писать программы на Ассемблере значительно сложнее и дольше. Команды Ассемблеров различных производителей микроконтроллеров сильно отличаются друг от друга. Поэтому, переходя от одного типа микроконтроллеров к другому типу, приходится, как бы, вновь осваивать новый язык. Да и замысловат он очень. Поэтому в настоящее время, когда технический прогресс требует скорейшего внедрения новых разработок, альтернативой Ассемблеру должны стать языки высокого уровня. В связи с тем, что сейчас многие производители выпускают огромное количество различных типов микроконтроллеров, мы во всех тех случаях, когда памяти микроконтроллера не хватает, всегда можем выбрать более емкий микроконтроллер. А если нас не устраивает скорость выполнения, то мы можем задать более высокую частоту задающего генератора. Многие специалисты утверждают, что лучшим языком программирования является - СИ. Но мы-то с Вами (почитатели других языков) знаем, что лучшим языком является тот, на котором ты грамотнее всего общаешься.

Любая программа представляет собой набор из сотен тысяч или даже миллионов инструкций процессора, каждая из которых кодируется одним или несколькими байтами (эти инструкции еще называются машинным кодом). Пытаться составлять программу, просто набирая коды инструкций, — занятие бессмысленное. Уже после десятка введенных таким способом команд человек теряет нить рассуждений, начинает путаться и допускает ошибки.

Кроме того, программа — это не просто набор вычислительных инструкций. Для общения с внешними устройствами, например, для считывания информации о изменении состояния каких либо датчиков, в наборе команд микроконтроллера есть специальная инструкция — прерывание, которая прерывает работу процессора и передает управление некоторой подпрограмме. Чтобы грамотно использовать прерывания, желательно детально разбираться в устройстве самого контроллера и понимать логику его работы.

В 50 годы прошлого века появился первый высокоуровневый язык программирования. Высокоуровневые языки программирования предназначены для написания программ с помощью привычных для человека терминов. Они манипулируют не конкретными ячейками памяти и элементарными

инструкциями, а командами естественного языка (точнее, командами, напоминающими естественный язык), например:

если ускорение равно 0

то вычислить расстояние = скорость * время;

Эти команды с помощью специальных программ автоматически переводятся в машинный код.

Для того чтобы в дальнейшем избежать любых недоразумений в тексте, я должен разъяснить несколько терминов, которые будут использоваться часто в этой книге:

Программа это некоторый текст, который состоит из последовательности команд, написанных на языке программирования. Эти команды микроконтроллер выполняет последовательно одна за другой, либо осуществляет переходы, определенные правилами языка. В главе 2 детально рассматривается структура выбранного языка.

Компилятор - программа, выполняемая на компьютере, и её задача состоит в том, чтобы транслировать первоначальную (или исходную) программу в машинный код, который затем может быть записан в микроконтроллер. Процесс трансляции BASIC - программы в исполняемую программу показан на рисунке ниже. Программа, написанная на языке BASIC и сохраненная как файл с расширением *.bas, преобразуется компилятором в ассемблерный код и сохраняется как файл программы на языке ассемблера - *.asm. Сгенерированный таким образом ассемблерный код далее будет оттранслирован в файл с расширением - *.hex или загрузочный файл. Этот файл предназначен для непосредственной загрузки в память микроконтроллера.

Программатор - устройство, которое используется, для записи файлов программ в память микроконтроллера.

Составлять программу на языке высокого уровня, конечно, удобно. Набрал текст в редакторе, записал команды в соответствии с алгоритмом решения задачи — и все. Затем запускаем программу - компилятор, которая автоматически переведет текст Вашей высокоуровневой программы в шестнадцатеричный файл. Затем этот файл можно записать с помощью программатора в микроконтроллер. При этом еще будет создан файл на языке ассемблера. Весь этот процесс представлен на рисунке 1.

Рис.1. Процесс преобразования исходного текста программы в конечный шестнадцатеричный .hex файл и запись его в микроконтроллер.

Глава 1. PIC-микроконтроллеры

Фирма Microchip в настоящее время является одним из мировых лидеров по выпуску микроконтроллеров. Распространенность этих микроконтроллеров можно сравнить с распространенностью компьютеров PC. Все выпускаемые фирмой микроконтроллеры можно условно разделить на три семейства. Это младшее семейство, в которое входят микроконтроллеры с двенадцатиразрядной шиной команд. К ним относятся микроконтроллеры: PIC 12C5xx, PIC 16C5xx, PIC 16C50x. Здесь символу x – соответствует целое число, обозначающее конкретный тип микроконтроллера.

Среднее семейство с 14 разрядной шиной команд. Это микроконтроллеры: PIC 12C6xx, PIC 1400, PIC 16C55x, PIC 16C6x(x), PIC 16C7x(x), PIC 16C8x, PIC 16F8x(x), PIC 16C9xx.

И старшее семейство, у которого шина команд имеет 16 разрядов. Это микроконтроллеры: PIC 17Cxx и PIC 18Xxx

Питание микроконтроллеров

Довольно известный популяризатор микроконтроллеров Майк Предко в книге «Handbook of Microcontrollers» утверждает, что если какой-либо микроконтроллер требует напряжения питания более +5В, а также применения сложных внешних схем сброса и тактирования, то лучше отказаться от использования такого микроконтроллера. В этом трудно с ним не согласится. Микроконтроллеры для того и создаются разработчиками, чтобы как можно меньше тратить время на электронику и больше уделить внимание разработке программного обеспечения.

У всех микроконтроллеров есть два вывода питания V_{SS} и $V_{DD.}$

- V_{SS} – общий («минус питания»);
- V_{DD} – «плюс» питания.

Для подключения PIC-микроконтроллеров к источнику питания необходимы фильтрующие конденсаторы емкостью от 0,01 до 0,1 мкФ между выводами V_{SS} и $V_{DD.}$ Следует отметить, что этот конденсатор должен иметь малые потери. Обычно для этих целей рекомендуется использовать керамические и танталовые конденсаторы.

Основная масса PIC-микроконтроллеров питается от напряжений 4,0 - 6,0 В, а некоторые типы могут работать и от 2,0 В. Они отличаются от типовых только тем, что были специально испытаны предприятием-изготовителем на функционирование при таком низковольтном напряжении питания. Для обозначения PIC-микроконтроллеров с низковольтным питанием перед буквами C или F добавляется буква L. И если Вы не проектируете какие-то специальные автономные устройства, где важен вес и объем изделия, то лучше остановить свой выбор на источнике питания +5 В. Тем более, что большинство микросхем, которые Вы будете подключать к микроконтроллеру, требуют именно такого напряжения питания. Для этих целей удобно использовать микросхему 7805 (российский аналог 142ЕН5), либо стабилитрон на 5.1 В. На рисунке 2 представлены схемы подключения таких источников.

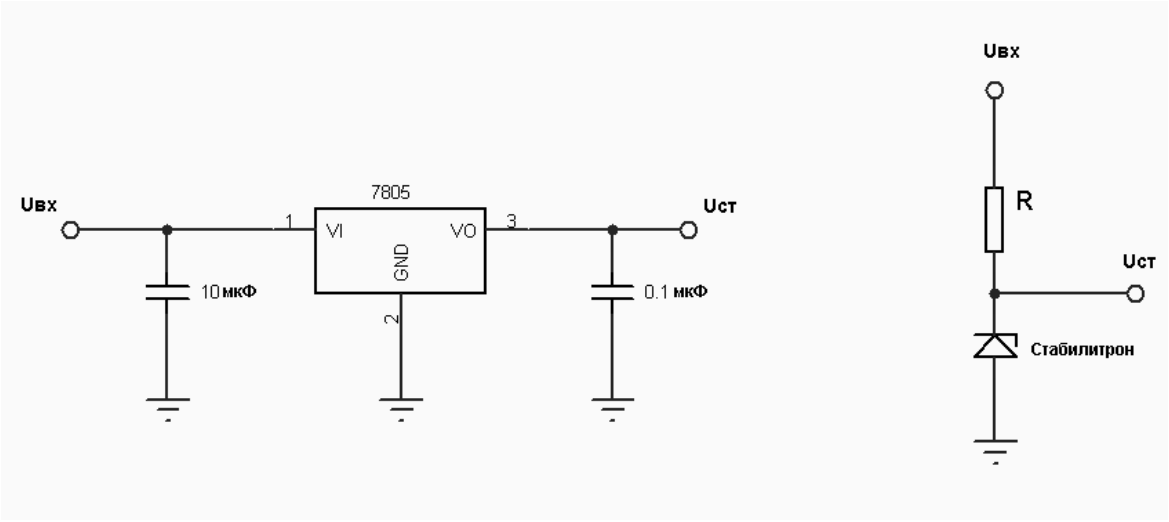


Рис. 2. Варианты схем источников питания микроконтроллеров.

Для выбора номинала токозадающего резистора можно воспользоваться формулой

R = (Uвх –Uст)/(Iвых +Iст),
где R – сопротивление токозадающего резистора,
Uвх – нестабилизированное входное напряжение,
Uст – необходимое выходное стабилизированное напряжение,
Iвых – необходимый ток питания,
Iст – номинальный ток стабилизации используемого стабилитрона.

Мощность резистора должна быть не меньше чем:

P = (Uвх –Uст)*(Iвых +Iст).

Сброс и перезапуск микроконтроллеров.

Все микроконтроллеры имеют вывод сброса, называемый MCLR. У PIC-микроконтроллеров предусмотрена внутренняя схема автоматического сброса при включении напряжения питания (рис. 3). Она устойчиво работает, если скорость роста напряжения достаточно высока (обычно 0,05 В/мс). Если напряжение питания при включении растет медленно, требуется внешняя схема сброса (так называемый ручной сброс), один из вариантов которой представлен на рис 4. Внешняя схема сброса может потребоваться, если Вы используете кварцевый задающий генератор относительно низкой частоты, с достаточно большим временем «разгона». Эта схема известна пользователям и может применяться для микроконтроллеров, выпускаемых не только компанией Microchip, но и другими фирмами. Следует обратить внимание на резистор R, значение которого может варьироваться от 100 Ом до 1 кОм. Этот резистор служит для защиты входа MCLR микроконтроллера от выбросов положительного напряжения на конденсаторе C при выключении питания.

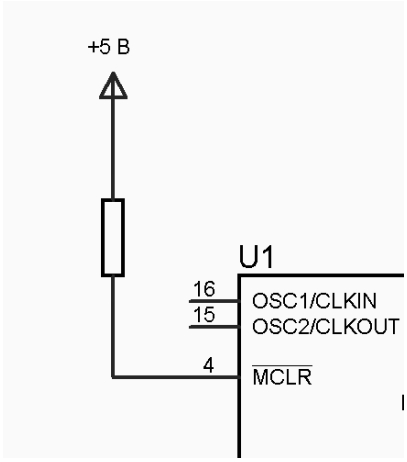


Рис 3. Схема подключения цепи MCLR для осуществления внутреннего автоматического сброса при включении напряжения питания.

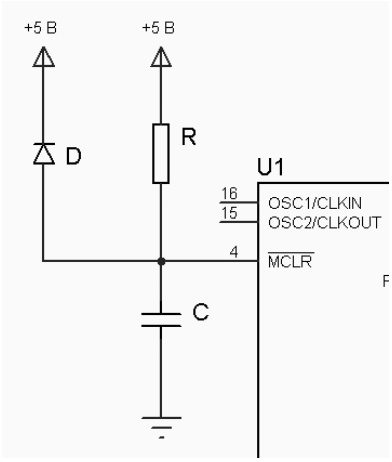


Рис.4. Схема подключения цепи MCLR для осуществления внешнего автоматического сброса при включении напряжения питания.

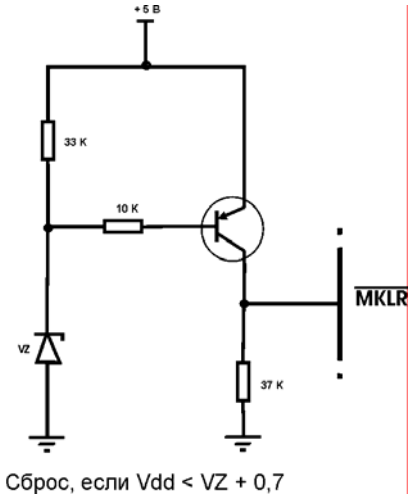
И, наконец, если напряжение питания может снижаться, до уровней способных нарушить нормальную работу микроконтроллера, целесообразно использовать схему, инициирующую сброс, когда напряжение падает ниже определенного порога. Две схемы внешнего сброса, предлагаемые фирмой Microchip, представлены на рис 5. Схема со стабилитроном, обладающая большей точностью, обеспечивает сброс как только питание опускается ниже VZ+0,7 В, в то время как схема без стабилитрона – при снижении напряжения до уровня 0,7*(1+R2/R1). В последнее время широкое распространение получили интегральные мониторы питания. Микросхемы данного типа предназначены для точного формирования сигнала сброса (RESET) в микропроцессорных системах. Принцип действия этих приборов следующий: в момент включения системы устройство тестирует величину напряжения питания и выдает сигнал сброса только тогда, когда напряжение питания достигает своей номинальной величины. Также эти микросхемы могут выполнять ряд дополнительных функций.

1. Это функция сторожевого таймера, позволяющая микросхеме отслеживать активность микроконтроллера и в случае отсутствия таковой (микроконтроллер «завис») производить принудительный сброс.

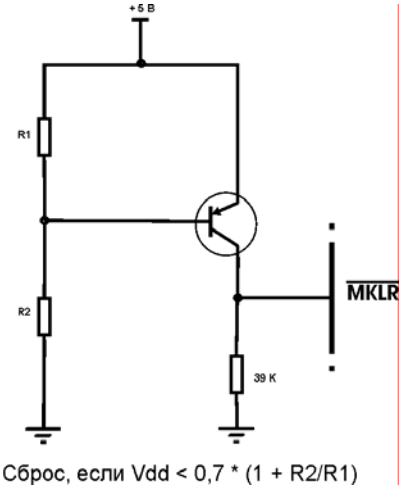
2. И другая функция это переключение на батарею, которая позволяет в случае отказа основного источника питания запитать слаботочные цепи микропроцессора (ОЗУ) от внешней резервной батареи.

В качестве примеров таких микросхем можно назвать: ADM709, DS1233, MAX809, КР1171СП47.

Следует отметить, что схемы сброса при понижении напряжения питания в современных PIC-микроконтроллерах активизируются при напряжениях ниже 4,5 В. Поэтому они, как правило, не применяются в микроконтроллерах с низковольтным питанием, за исключением специальных типов, которые имеют программируемые схемы сброса при понижении напряжения питания



Сброс, если Vdd < VZ + 0,7



Сброс, если Vdd < 0,7 * (1 + R2/R1)

Рис 5. Схемы подключения цепи MCLR для осуществления внешнего автоматического сброса при снижении напряжения питания ниже определённого уровня.

Регистры конфигурации.

Регистры конфигурации, в которых содержатся конфигурационные слова, предназначены для задания самых общих параметров работы PIC-микроконтроллеров. Они позволяют указать:

- Режим работы генераторов;
- Режим защиты памяти программ;
- Параметры системы перезапуска микроконтроллеров;
- Запуск сторожевого таймера;
- Режим отладки микроконтроллеров PIC16F87x.

Биты слова конфигурации индивидуальны для каждого подсемейства PIC-микроконтроллеров.

Генераторы тактовых импульсов.

Тактовый генератор задает периоды времени (или дисктреты), в течение которых происходит выполнение инструкций. Внутренний машинный цикл PIC-микроконтроллера (T_{CY}) состоит из четырех периодов тактового сигнала. Для формирования тактового сигнала в PIC-микроконтроллерах предусмотрен внутренний генератор.

В зависимости от типа и расположения времязадающих элементов могут использоваться следующие типы генераторов тактовых импульсов:

- встроенные генераторы;
- RC-генераторы;
- генераторы с кварцевыми резонаторами;
- генераторы с керамическими резонаторами;
- внешние генераторы.

Встроенные генераторы применяются во многих разработанных в последнее время PIC-микроконтроллерах. Понятно, что стоимость такого генератора минимальна. Этот генератор построен на основе конденсатора и программируемого резистора.

Вторым по стоимости является генератор с внешней RC-цепочкой. На рисунке 6.а показана схема подключения такого генератора. Его активным элементом является неинвертирующий буфер, который выполнен на основе триггера Шмидта и служит для открывания или запирания ключа на МОП транзисторе с N-каналом. Величины емкости конденсатора и сопротивления резистора определяются по спецификациям фирмы Microchip. Необходимо заметить, что

для сопротивлений меньше 2.2 кОм частота тактового генератора может быть нестабильной или вообще генерация может срываться. А для сопротивлений больших 1 Мом генератор становится чувствительным к внешним помехам, токам утечки и влажности. Поэтому свой выбор Вы должны осуществлять в диапазоне от 3 кОм до 100 кОм. Такой генератор может работать и без внешнего конденсатора (C=0). Но для стабильности все же рекомендуется ставить конденсатор емкостью более 20 пФ. В этом случае уменьшается влияние емкости печатных проводников.

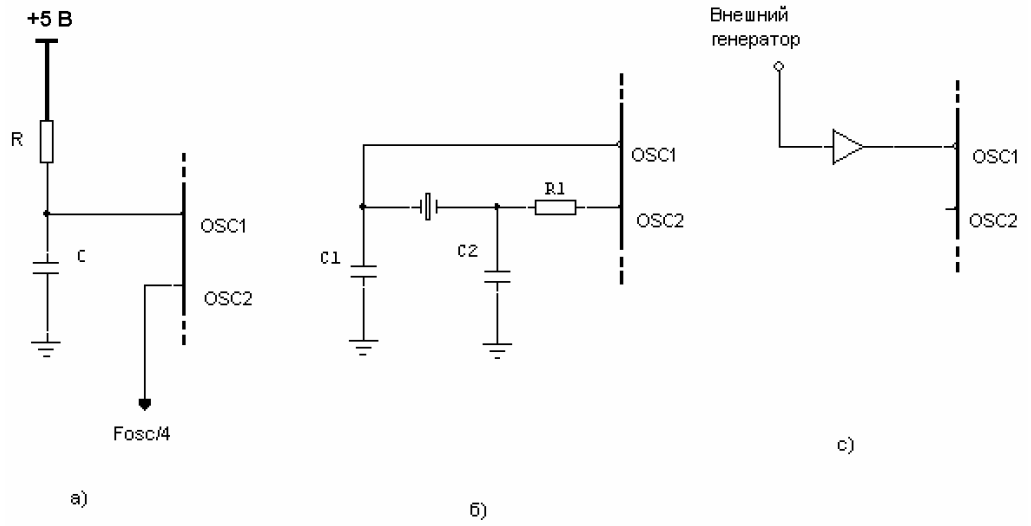


Рис 6. Схемы подключения различных типов тактовых генераторов к PIC-микроконтроллерам

Схемы генераторов с кварцевым и керамическим резонаторами практически не отличаются друг от друга. Подключение кварцевого или керамического резонатора производится согласно схеме, показанной на рисунке 6.б. При этом для нормального функционирования резонатора предусматривается подключение двух дополнительных конденсаторов. Емкость этих конденсаторов оговаривается в спецификациях фирмы Microchip.

Для каждого микроконтроллера существует три диапазона рабочих частот, каждому из которых соответствует определенный режим работы кварцевого резонатора. Частотные диапазоны кварцевых генераторов PIC-микроконтроллеров и обозначения режимов указаны в таблице 2 и задаются с помощью регистра конфигурации.

Режимы работы тактовых генераторов PIC-контролеров и их частотные диапазоны

Таблица 2.

ТИП	ЧАСТОТЫ
RC	4 МГц
LP	0 – 200 кГц
XT	200 кГц – 4 МГц
HS	4 – 20 МГц (максимальное

	значение)

Значение конденсаторов C1 и C2 выбирается в зависимости от типа резонатора (кварцевый или керамический) и частоты (таблица 3).

Таблица 3.

Тип резонатора	Частота кГц	C1, пФ	C2, пФ
LP	32*	15	15
	100	15	15
	200	0-15	0-15
XT	100	15-30	200-300
	200	15-30	100-200
	455	15-30	15-100
	1000	15-30	15-30
	2000	15	15
	4000	15	15
HS	4000	15	15
	8000	15	15
	20000	15	15

Для версии XT резистор R1 не нужен, однако иногда он требуется для микроконтроллеров версии HS. Только точное знание характеристик кварцевого резонатора позволяет определить, есть ли необходимость в резисторе R1, и каким должно быть его значение.

И, наконец, схема на рисунке 6.c показывает способ тактирования PIC-контроллера внешним генератором. Очевидно, что формируемые внешним генератором уровни должны соответствовать напряжению питания микроконтроллера. В тех случаях, когда используется только один вывод OSC1, на выводе OSC2 имеем внутреннюю рабочую частоту микроконтроллера (частота командных циклов), которая в четыре раза меньше, чем частота генератора. Это так называемый вывод CLCOUT.

Все остальные выводы микроконтроллеров выполняют функции портов ввода/вывода, с помощью которых обеспечивается взаимодействие внутренних узлов с периферией.

Порты ввода/вывода и регистры управления передачей данных.

Обычно каждый порт микроконтроллера состоит из некоторого числа однотипных структурных компонентов, каждый из которых отвечает за определенный разряд порта ввода/вывода.

Для выбора направления передачи данных через порты существуют регистры управления – TRIS (TRI-state buffer enable). Название регистра TRIS соответствует

названию порта, которым он управляет. То есть, портом PORTB управляет регистр TRISB, а портом PORTC управляет регистр TRISC.

При записи 1 в какой либо разряд регистра TRIS соответствующий вывод одноименного порта переводится в состояние входа. Если же записать в этот разряд 0 то вывод порта переводится в состояние выход. Запомнить это можно, если обратить внимание на то, что 1 (единица) похожа на латинскую букву I (Input – вход), а 0 (ноль) похож на букву O (Output – выход). Таким образом, запись команды: TRISC=%01100111 означает, что разряды порта C – 0, 1, 2, 5, 6 установлены как выходы, а разряды – 3, 4, 7 установлены как входы. Следует отметить, что команду TRIS не рекомендуется использовать при программировании PIC-микроконтроллеров среднего и старшего подсемейства, поскольку она обеспечивает доступ лишь к портам PORTA, PORTB и PORTC, тогда как порты PORTD и PORTE не могут управляться с её помощью. В этих случаях лучше использовать команды BASIC - INPUT или OUTPUT.

Разряд 4 порта PORTA (RA4) в PIC-микроконтроллерах среднего подсемейства выполняется в виде выхода с открытым стоком. Этот вывод не может самостоятельно обеспечить выдачу логической единицы без подключения резисторов к напряжению питания (подтягивающих резисторов).

Выполнение операции, связанной с «подтяжкой» входов порта PORTB к напряжению питания (PORTB weak pull-up), разрешается с помощью бита _RPBU регистра опций – OPTION. Подобное разрешение дается при сбросе этого бита. Например:

```
Option_Reg.7=0          ‘ Подключить внутренние подтягивающие
‘ резисторы PORTB
```

Величина внутреннего сопротивления переключателя, используемого для реализации режима «подтяжки», составляет приблизительно 50 кОм. Наличие подобного переключателя упрощает взаимодействие с клавиатурой и исключает необходимость в использовании внешних резисторов.

Замечание. В некоторых микроконтроллерах младшего подсемейства (PIC12C50X, PIC12C67x, 12CE67x и 12F675) названия портов изменены. Вместо PORTA он называется GPIO соответствующий этому порту регистр - TRISA. Для уточнения всегда смотрите справочные листы на используемое устройство.

Глава 2. Компилятор PicBasic Pro.

BASIC или Beginner's all purpose Symbolic Instruction Code (многоцелевой символический код для начинающих) был разработан в начале 60 годов в Дартмутском колледже для учебных целей. Благодаря своей простоте, он быстро завоевал популярность и был реализован для множества тогдашних компьютеров.

Очередная компьютерная революция совершилась в середине 1970 годов, когда на свет появились первые микрокомпьютеры для домашнего использования, по своим возможностям напоминавшие современные программируемые

микрокалькуляторы. Для одного из таких персональных компьютеров ALTAIR и был реализован BASIC, как самый простой из созданных к тому времени языков программирования. Выполнил эту работу Билл Гейтс (Bill Gates), нынешний президент Microsoft, со своим напарником Полом Алленом (Paul Allen). В дальнейшем усилиями Гейтса BASIC был перенесен на платформу IBM PC, а его наиболее доработанная версия QBasic (Quick Basic или быстрый BASIC) вошла в базовую поставку операционной системы MS-DOS.

Этот язык реализован для множества операционных систем. В частности, для операционной системы MS-DOS имеется версия QBasic и Visual Basic for MS-DOS, для Windows почти каждый год выходит новая версия Visual Basic (последней была 6-я версия). Во все офисные приложения компания Microsoft встроила язык Visual Basic for Applications (VBA), используемый в них в качестве своеобразного макроязыка. Например, в текстовом редакторе Microsoft Word на VBA можно разработать преобразователь в нужный вид текста, случайно набранного не в том регистре, или даже игру в крестики-нолики. На VBA пишутся, к сожалению, и печально известные вирусы, способные через Интернет за несколько часов распространиться по всему миру. Хорошо зная одну, базовую версию BASIC — QBasic, в дальнейшем можно свободно переходить к работе с другими версиями этого простого и очень популярного языка.

Для программирования PIC-контроллеров в 1992 - 1993 году фирмой Parallax был создан компилятор BasicStamp. Несмотря на то, что компилятор с языка BASIC фирмы Parallax, по отзывам многих пользователей, очень неплохой компилятор, он не завоевал большой популярности. Это, на мой взгляд, связано с тем, что этот компилятор предназначен для электронных плат-модулей, построенных на базе PIC-контроллеров и выпускаемых этой фирмой. А это значит, что разработчик ограничен в выборе электронных составляющих своего изделия, да и цена модуля выше, чем у отдельного микроконтроллера.

Несколько позже фирмой microEngineering Labs. Inc были созданы компиляторы PicBasic и PicBasicPro. Затем появился компилятор Basic PROTON+ фирмы Crownhill. Эти компиляторы создавались по принципу максимальной совместимости с компилятором BasicStamp или BS. Это было нужно для того, чтобы программы, написанные в BasicStamp, легко переносились в новые компиляторы. И, наконец, 1998 году появился microBasic фирмы mikroElektronika, который в настоящее время довольно интенсивно развивается**.**

Интегрированная среда разработки (IDE) - Microcode Studio Plus. Установка и настройка.

Для того чтобы писать и отлаживать программы, Вам необходим удобный текстовый редактор, который был бы способен воспринимать язык программирования. А также компилятор, способный качественно компилировать текст Вашей программы в машинный код. Также было бы очень удобно, если бы под руками находились различные средства отладки программ. Для решения этих и им подобных задач и существуют интегрированные среды разработки. Одним из

таких программных пакетов и является пакет «Microcode Studio Plus». В Интернете на страницах: www.melabs.com и www.mecanique.co.uk Вы можете найти и скачать облегченную версию интегрированного пакета программ - «MicroCode Studio», которая распространяется бесплатно. Здесь же можно купить полную версию пакета - «Microcode Studio Plus». На этих Интернет - страницах Вы сможете найти облегченную версию компилятора «PICBasic Standard» и полную версию – «PICBasic PRO». Демонстрационная версия пакета «MicroCode Studio» отличается от пакета «Microcode Studio Plus» тем, что размер программ ограничен 31 строками (при этом количество переменных, символов и констант не ограничено). Несмотря на то, что многострочные утверждения допустимы, но, тем не менее, каждая команда будет подсчитана как новая строка.

Демонстрационный компилятор поддерживает только следующие типы PIC-контроллеров: 16F84 (A), 16F627 (A), 16F628 (A), 16F870, 16F871, 16F872, 16F873 (A), 16F874 (A), 16F876 (A) и 16F877 (A), в то время как полная версия поддерживает практически все выпускаемые фирмой Microchip. В демонстрационной версии Вы также не сможете подключать файлы к основной программе с помощью команды **INCLUDE**.

Пакет программ «Microcode Studio Plus» представляет собой удобную среду разработки и отладки программ для PIC-микроконтроллеров. С помощью этого программного продукта мы можем набрать и отредактировать текст программы, откомпилировать ее в машинный код, записать его с помощью программатора непосредственно в микроконтроллер, либо осуществить внутрисхемное программирование с помощью встроенного загрузчика bootloader. В этой среде можно также обмениваться данными с реальной платой, подключенной кабелем к одному из COM портов компьютера, осуществлять интерактивную отладку программы. Обновлять программное обеспечение можно в режиме On Line через Internet.

Разработчик этой программы допускает установку её в системах Windows 98, 98SE, ME, NT 4.0 с SP 6, 2000 и XP (рекомендуется). При этом требования к компьютеру следующие:

- Процессор на 233 МГц (рекомендуется 500 МГц и выше),
  ОЗУ объемом 64 Мб (рекомендуется 128 Мб и выше),
  40 Мб свободного места на винчестере,
  16 битная видеокарта.

Для установки пакета программ открываем папку «MCSP_2300», находим и запускаем установочный файл setup.exe. После его запуска перед нами появляется заставка с окном, в котором спрашивается наше согласие на дальнейшую установку программы. Нажав кнопку «***Next***», мы перейдем к следующему окну, в котором, как обычно, нам будет предложен текст лицензионного соглашения. Если мы соглашаемся с ним, то давим на кнопку «***Yes***», ну а если нет, то «***No***». В случае согласия перед нами открывается окно, в котором нам будет предложено место установки программы. По умолчанию это - C:\Program Files\Mecanique\MCSP. Если мы не возражаем против такого варианта, то вновь кнопка «***Next***».

В следующем окне нам будет предложена папка, в которую мы пожелали бы поместить иконку для запуска программы. По умолчанию нам предлагается создать папку под названием - «MicroCode Studio Plus». И вновь кнопка «***Next***». Начинается процесс установки. После успешного окончания установки появляется окно, в котором нам сообщается, что установка успешно завершена. Здесь выбора у нас нет, это только кнопка – «***Finish***».

После того как программа была установлена на Ваш винчестер, запустим программу. Программа загрузится и перед нами откроется её рабочий стол рис. 7.

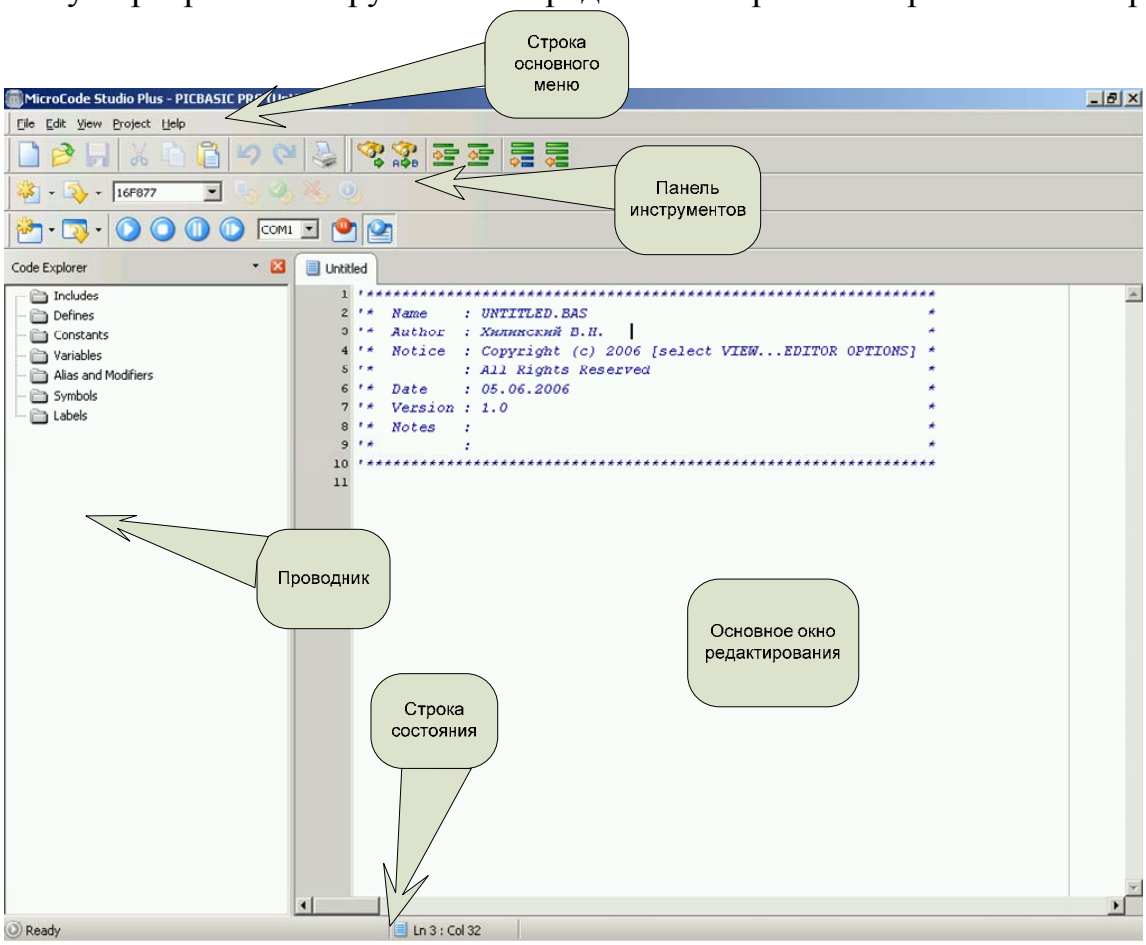


Рис. 7. Основное окно программы «Microcode Studio Plus»

В верхней части окна программы мы видим строку основного меню, состоящую из 5 пунктов: File, Edit, View, Project и Help.

Ниже строки основного меню располагается панель инструментов. Она в свою очередь состоит из следующих частей: стандартные инструменты, инструменты редактирования, компиляции и программирования, а также панель ICD отладчика.

Ниже панели инструментов размещены два окна. Слева окно обозревателя программы, а справа основное окно ввода и редактирования текста программ. В самом низу располагается строка состояния.

Пункты основного меню позволяют решать следующие задачи:

Меню File

- **New** - создает новый документ. При этом автоматически генерируется «шапка» программы, в которой указывается автор, авторские права и дата создания документа.

Когда создается новый документ, по умолчанию выбирается микроконтроллер16F877. Чтобы это изменить нужно, отредактировать файл 'default.ini'. Этот файл находится в папке, которая была создана при инсталляции программы. Это обычный текстовый файл и поэтому его можно отредактировать в любом текстовом редакторе.

- **Open** - Открывает диалоговое окно, в котором Вы можете выбрать документ для загрузки в основное окно программы.
- **Save** - Сохраняет документ на диске. Эта кнопка обычно заблокирована, если документ не был изменен.
- **Save As** – Позволяет производить сохранение как диалог, давая Вам возможность присвоить любое имя фалу и сохранять его в любое место на диске.
- **Close** - закрывает активный в настоящее время документ.
- **Close All** - закрывает все документы, загруженные в редактор, а затем создают новый документ.
- **Reopen -** отображает список документов открытых в последний раз.
- **Print Setup -** Отображает диалог установки печати.
- **Print Preview -** Отображает окно предварительного просмотра информации, выводимой на печать.
- **Print** – Выводит на печать активную в настоящее время страницу редактора.
- **Exit -** дает Вам возможность выйти из программы.

Меню Edit

- **Undo** - Отменяет любые последние сделанные изменения на активной странице документа.
- **Redo** - Полностью отменяет команду **Undo**.
- **Cut** - Вырезает любой выбранный текст на активной странице документа и помещает его в буфер обмена. Эта опция заблокирована, если никакой текст не был выделен.
- **Copy** - помещает любой выделенный текст на активной странице документа в буфер обмена. Эта опция заблокирована, если ничего не было выделено**.**
- **Paste** - Вставляет содержимое буфера обмена сразу после курсора на активной странице документа. Эта опция заблокирована, если буфер обмена пуст.

- **Delete** - Удаляет любой выбранный текст. Эта опция заблокирована, если никакой текст не был выделен.
- **Select All** - Выделяет весь текст на активной странице документа.
- **Find** - Отображает диалог поиска фрагмента текста программы.
- **Replace** - Отображает диалог поиска и замены.
- **Find Next** - Автоматически ищет следующее выбранное слово. Если никакой признак не был выбран, то используется слово в текущей позиции курсора. Вы можете также выбрать целую фразу, которая будет использоваться как термин поиска.

Меню View

- **Code Explorer** - Отображает или закрывает окно проводника программы.
- **Serial Communicator** – Загружает программу связи компьютера с Вашим устройством по протоколу RS-232.
- **EasyHID USB Wizard –** Загружает программу связи компьютера с Вашим устройством по протоколу USB.
- **Loader** - Загружает встроенную программу MicroCode Loader.
- **Loader Options** - Отображают окно диалога вариантов работы встроенного загрузчика.
- **Compile and Program Options** – Отображает окно настройки вариантов компилирования и настройки работы программатора.
- **Editor Options**- Отображают окно основных настроек редактора.
- **Toolbars** - Отображает меню инструментальной панели. Здесь Вы можете выбрать какие кнопки инструментов отображать, а какие нет.
- **Online Updates**- Проверяет обновления в режиме Online.

Меню Project

- **Compile** – При выборе этой опции начинается компилирование программы, которая в настоящее время находится на активной странице редактора.
- **Compile and Program** - При выборе этой опции начинается компилирование программы, которая в настоящее время находится на активной странице редактора, а затем автоматически запустится выбранный Вами программатор и происходит запись скомпилированной программы в микроконтроллер.
- **Program** - Автоматически запускается выбранный Вами программатор.

- **ICD Compile** - эта опция запускает компилирование активной страницы программы редактора, но в отличии от предыдущей опции здесь в скомпилированный файл будет включена информация об отладчике.
- **ICD Compile and Program** - Выбор, этой опция запускает компилирование активной страницы программы редактора с информацией об отладке и затем автоматически запускается выбранный Вами программатор.

Меню Help

Help Topics – Загружается файл справки.

- **Loader Help Topics** - Загружается файл справки встроенного загрузчика
- **Online Form** - Посещает сетевой форум PICBASIC.
- **About** - Отображает окно с информацией о версии программного пакета MicroCode Studio и компилятора PICBasic.

На панели инструментов находятся следующие пиктограммы:

 **Find**- поиск фрагмента текста. Клавиши **Ctrl+F**

 **Replace**- поиск и замена. Клавиши **Ctrl+F**

 **Indent -** Сдвигает все выбранные строки в следующую позицию табулятора. Если строки не выбраны, то перемещается только одна строка, в которой в настоящий момент находится курсор.

 **Outdent -** Сдвигает все выбранные строки в предыдущую позицию табулятора. Если строки не выбраны, то перемещается только одна строка, в которой в настоящий момент находится курсор.

 **Block Comment -** Добавляет символ комментария на каждой строке выбранного блока текста. Если не выбрано ни одной строки то добавляется знак комментария в начале строки, на которой находится курсор.

 **Block Uncomment -** Удаляет все символы комментариев из каждой строки выбранного блока текста. Если ни одной строки не выбрано, то удаляется комментарий из начала строки, в которой находится курсор.

 **Compile Only**- компилирование активного окна исходного текста. Клавиши **F9**.

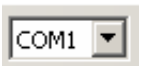
 **Compile and Program**- компилирование активного окна с последующим запуском программатора и записью кода в микроконтроллер. Клавиши **F10**.

 **Target Processor –** Окно выбора процессора

 **ICD Compile –** При нажатии этой кнопки программа компилируется точно так же как и в предыдущем случае за исключением того, что добавляется дополнительная информация для работы в режиме отладки. При компиляции создается шестнадцатеричный файл, который Вы затем можете с помощью программатора записать в Ваш микроконтроллер. **Замечание**: Нажав на эту кнопку, происходит автоматическое сохранение на диске всех открытых файлов. Кроме того, если сгенерированный таким образом *.hex файл записать в Ваш микроконтроллер, то он будет работать только под управлением ICD. И не будет работать в автономном режиме. Клавиши **Ctrl+F9**

 **ICD Compile and Program –** По нажатию этой кнопки происходит компилирование и трансляция активного в настоящий момент текста программы с добавлением в него отладочной информации и запись кода программы в микроконтроллер. Клавиши **Ctrl+F10**

 **ICD Control -** Кнопки управления дают Вам возможность запускать, останавливать, делать паузу и продолжать дальше режим отладки. Помните, когда Вы создаете свой проект, Вы должны компилировать его с помощью кнопки «**ICD Compile**» или «**ICD Compile and Program**». Клавиши **F5, F6, F7, F8**

 **ICD Communications Port –** Здесь Вам предоставлена возможность выбрать COM порт компьютера для связи в режиме ICD.

 **ICD Breakpoint -** Контрольные точки. Эта кнопка используются для того, чтобы установить в программе точки, в которых выполнение программы можно было бы остановить. Чтобы добавить контрольную точку, поместите курсор редактора на строку, в которой требуется останавливать выполнение программы, затем нажмите эту кнопку. Чтобы удалять контрольную точку, нажмите кнопку снова. Вы можете также добавить контрольную точку, используя левую кнопку мыши. Для этого установите курсор мыши в левой серой области рядом с окном редактирования напротив выбранной строки и нажмите на правую кнопку мышки.

 **ICD Animate –** При нажатии на эту кнопку программа начинает анимировать выполнение программы. При этом Вы будете видеть, как

подсвечивается строка программы в тот момент, когда выполняется команда, находящаяся на этой строке. При этом в окне проводника ICD будут обновляться текущие значения всех переменных, регистров, памяти и СППЗУ. Если же нажать кнопку “PAUSE” процесс остановится, и обновления происходить не будет. Клавиши **F2**

В программный пакет «MicroCode Studio Plus» входит очень удобное средство отладки программ – интерактивный отладчик схем - ICD (In Circuit Debugger). С его помощью Вы сможете производить интерактивно отладку программы. Работу с отладчиком мы рассмотрим позже, а пока опишем только его инструменты.

Режим ICD может анимировать и на других частотах, в зависимости от установки частоты задающего генератора и в зависимости (также) от скорости работы компьютера. Вы сможете получить лучшие скорости обновления, используя параметры настройки генератора. Но не ожидайте больших скоростей анимации, если у Вас компьютер Pentium 90 с 8 Мегабайтами оперативной памяти, работающий на частоте 4MHz. В идеальном случае, Вы можете ожидать скорости обновления 10 команд в секунду. Режим ICD многофункционален. Здесь Вы можете заставить программу выполняться в режиме реального времени, а в ее критических местах останавливаться и отслеживать состояние переменных, регистров и памяти. Проводник программы позволяет Вам автоматически переходить к подключаемым файлам, константам, переменным, псевдонимам и модификаторам, символам и меткам, которые содержатся в вашем исходном тексте.

Когда Вы щелкаете мышкой по подключаемому файлу, редактор автоматически откроет этот файл для просмотра и редактирования. Если файл для включения не может быть найден или вообще не существует, для Вас будет создан новый документ.

Чтобы пересортировать узлы Проводника Программы, щелкните правой кнопкой мыши в окне проводника и выберите опцию «sort».

Окно ошибок результатов компилирования появится только тогда, когда компилятор сталкивается с ошибками, при компиляции Вашей программы. Окно дает возможность Вам быстро перейти к строкам программы, которые содержат ошибки. Для этого Вам надо выбрать мышью ошибку, о которой сообщается в окне ошибок. Если ни о каких ошибках не сообщается, то окно ошибок скрыто, и имеется только сообщение об успешной компиляции, которое отображено в строке состояния.

Прежде, чем мы продолжим изучение, настроим редактор под себя. Для этого выберем пункт View в строке основного меню. В выпавшем меню выберем пункт Editor Options.. Перед нами раскроется окно с установками конфигурации редактора. В появившемся окне имеются 4 закладки. На основной закладке мы видим 14 пунктов. Вот эти пункты:

Show Line Numbers in Left Gutter - Отображать номера строк программы в левой колонке.

Show Right Gutter - Отображать вспомогательную линию в основном окне редактора. Эта линия предназначена для выравнивания строк комментариев. И на каком расстоянии от левого края она должна находиться.

Use Smart Tabs - Использовать интеллектуальную табуляцию.

Convert Tabs to Spaces - Устанавливает количество знакомест, на которые перемещается курсор при нажатии на кнопку табуляции.

Automatically Indent - Когда выбрана данная опция, то при нажатии на клавишу ENTER курсор переходит на следующую строку и располагается точно под первым словом верхней строки. Если опция не выбрана, то курсор переходит на начало следующей строки.

Wrap Document Selection Tabs –Редактор позволяет быть открытыми одновременно множеству документов. Если документов будет открыто много, так что вкладки их уже не смогут умещаться на рабочем поле, то при выборе этой опции вкладки всех документов будут видны одновременно..

Automatically Jump to First Error – при выборе этой опции редактор автоматически переходит к первой строке программы в которой была обнаружена ошибка в результате компиляции.

Clear Undo History After Compile – Если выбрана эта опция то после успешной компиляции будет очищен буфер в который хронологически заносятся все действия, выполненные с этой программой. Это позволяет не перегружать систему.

Open Last File(s) When Application Starts - Когда выбрана эта опция, то при следующем запуске программы документы, которые были открыты, буду вновь загружены.

Display Full Filename Path in Application Title Bar - По умолчанию, MicroCode Studio в области основного заголовка отображается только имя файла документа (то есть, никакая информация о пути где он находится не показана). Если Вы хотите показать весь путь к файлу, то выберите эту опцию.

Prompt if File Reload Needed - MicroCode Studio автоматически выясняет, произведенные Вами изменения в тексте программы после компиляции файла. При попытке закрыть эту страницу, появляется диалоговое окно, где Вас спросят: «желаете ли Вы сохранить эти изменения?». Если этот пункт не выбран, то запрос выдаваться не будет.

Auto Change Identifiers - Когда выбрана эта опция, программа будет автоматически изменять форму записи идентификаторов в соответствии с тем, как это было определено вначале. Например, если было следующее объявление:

```
MyIndex VAR BYTE
```

а Вы в окне редактора набрали `myindex`, то MicroCode Studio автоматически изменит `myindex` на `MyIndex`. Необходимо отметить, что автоматическая замена фактически не изменяет само значение, а только изменяет способ, которым оно отображается в окне редактора. Если Вы скопируете вышеупомянутый пример и вставите в другой документ, то он вновь станет как `myindex`. Но в пределах блока ASM...ENDASM программа не делает

автоматической замены, даже если этот пункт выбран. Эта особенность касается только активного в настоящий момент документа, и не будет изменять идентификаторы, которые были объявлены в файлах, подключаемых командой **INCLUDE**.

Use .PBP as Default Extension - Выбор этой опции гарантирует, что по умолчанию используется расширение *.pbp, когда происходит сохранение файлов на диск. Если Вы хотите, чтобы расширение по умолчанию было *.bas, то не отмечайте эту опцию.

Default Source Folder - MicroCode Studio будет всегда автоматически входить в эту папку, когда Вы выбираете пункт меню **Open**, или **Save..as**. Чтобы отключать эту опцию уберите галочку в этом окне.

Следующая закладка - **Highlighter (**Выделение текста)

На этой закладке выбираются свойства шрифтов, используемых в программе. Здесь Вы можете изменять цвет и признаки (например, полужирный и курсив) следующих элементов:

- Комментариев
- Зарезервированных слов
- Идентификаторов
- Символов
- Строковых переменных
- Чисел
- Ассемблерных команд

Здесь же у Вас есть возможность установить, как MicroCode Studio будет отображать ключевые слова. Например, Вы можете установить так, чтобы все ключевые слова автоматически создавались в верхнем регистре, нижнем регистре или в верхнем регистре только первый символ после того, как ключевое слово будет напечатано.

Третья закладка - **Program Header** (Заголовок программы или «шапка» программы). Здесь Вы можете настроить вид шапки, которая будет загружаться при каждом запуске программы.

В последней, четвертой закладке Вы можете определить то, как Вы будете загружать обновления.

В следующей группе настроек Вы указываете программе, где находится компилятор, ассемблер и если у Вас есть программатор, то программа работы с этим программатором. Для этого вы должны выбрать в основном меню пункт **View > Compile and Program Options**.

В открывшемся окне перед Вами появятся три закладки.

На первой закладке - **Compiler**, Вы можете заставить MicroCode Studio определять местонахождение каталога компилятора автоматически, нажимая для этого кнопку «Find automatically». После этого запустится автоматический поиск. После того, как будет найден первый попавшийся файл компилятора PICBasic или PicBasicPro - автопоиск остановится. Однако, если у Вас в системе имеется несколько вариантов компиляторов (например, PBP 2.42, PBP 2.43 и т.д.), то Вам лучше установить путь к компилятору вручную, т.е. кнопка – «Find manually».

Следующая закладка – **Assembler**. По умолчанию Ассемблер, используемый MicroCode Studios, PM.EXE. Он предоставляет собой программу, разработанную фирмой microEngineering Labs для компилятора PicBasicPro. В общем случае использование этого компилятора предпочтительней. Он работает намного более быстро, чем ассемблер MPASM, предлагаемый фирмой Microchip. Однако есть некоторые случаи, когда выбор ассемблера MPASM был бы более правильным решением. Например, при использовании микроконтроллеров из ряда PIC18xxx. В настоящее время ассемблер PM.EXE пока не поддерживает этот ряд микроконтроллеров. Поэтому Вы будете обязаны использовать MPASM. Вы можете загрузить MPASM бесплатно с вебсайта Microchip.

По умолчанию, MicroCode Studio будет использовать базовый ассемблер mpasmwin.exe. Это более предпочтительно, так как он лучше использует память.

Замечание: Вы можете использовать ассемблер MPASM только с компилятором PICBasicPro.

И наконец последняя закладка – **Programmer**(программатор). Для программирования микроконтроллеров в среде MicroCode Studio Вам необходимо подключить имеющийся у Вас тип программатора. Это дает возможность Вам компилировать и затем программировать ваш микроконтроллер только несколькими щелчками мыши. Первую вещь, которую Вы должны сделать, - указать MicroCode Studio, какой программатор Вы будете использовать. Для этого, выберите кнопку **Add New Programmer**. Перед Вами откроется окно мастера установки другого либо нового программатора, рисунок 8.

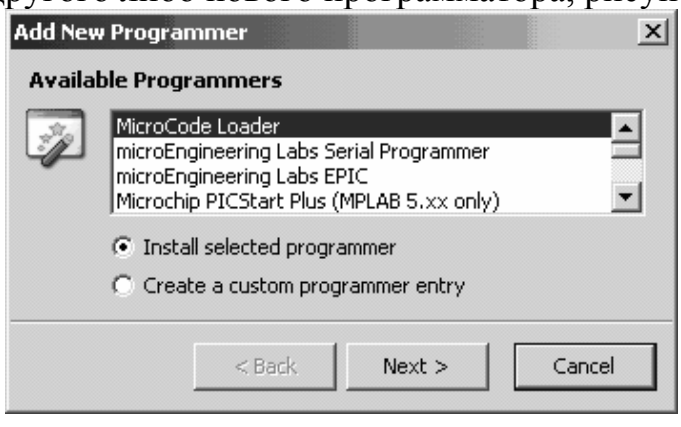


Рис. 8. Окно выбора программатора.

В этом окне Вы должны выбрать программатор и нажать кнопку **Next**. MicroCode Studio начнет искать на вашем компьютере, пока не определит местонахождение необходимого запускаемого файла. Ваш программатор теперь готов к употреблению. И теперь, когда Вы будете нажимать кнопку **Compile and Program** на основной инструментальной панели, Ваш программа будет скомпилирована, и затем записана в микроконтроллер.

В большинстве случаев, MicroCode Studio имеет ряд предварительно сконфигурированных программаторов, доступных для использования. Однако, если Вы будете использовать программатор, не включенный в этот список, то Вы должны будете добавить его. Чтобы это сделать, выберите пункт – **Create a custom programmer entry** и нажмите кнопку **Next**. Перед Вами появится следующее окно, где Вам будет предложено ввести имя нового программатора, рисунок 9. Это имя затем будет добавлено в список программаторов. В примере, показанном ниже, имя программатора – My New Programmmer.

Рис. 9. Окно установки нового программатора.

Введя имя программатора, нажимаем кнопку **Next** и переходим к следующему окну.

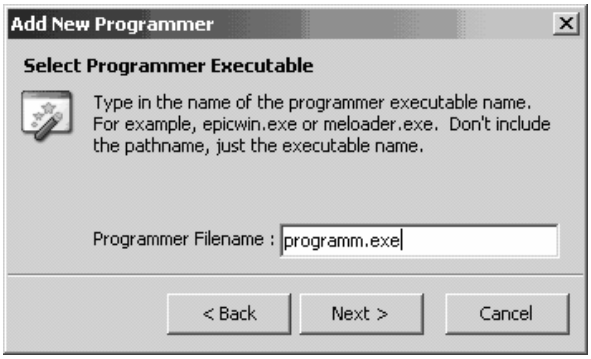


Рис.10. Окно выбора исполняемого файла

Следующее окно запрашивает название исполняемого фала программатора. Здесь не нужно давать полный путь к фалу, а ввести только название исполняемого файла. Как это сделано на рисунке 10. И вновь кнопка **Next**.

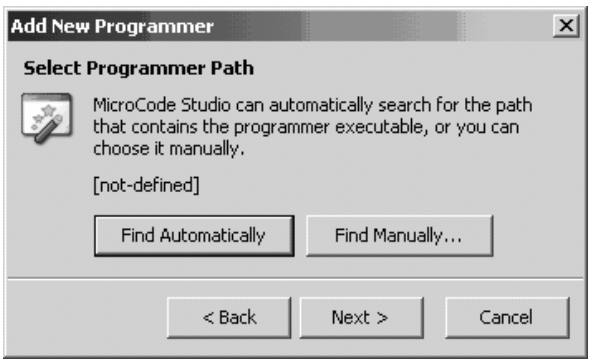


Рис.11. Окно указания пути к файлу

Перед нами появляется следующее окно рисунок 11. В этом окне нам предлагается выбор пути к файлу автоматически **Find Automatically**, либо вручную **Find Manually**.

И, наконец, последнее окно используется для того, чтобы установить параметры командной строки, которая будет передаваться вашему программатору. Некоторые программаторы, например, EPICWin позволяют Вам передавать через эту строку тип микроконтроллера и шестнадцатеричное имя файла. Таким образом MicroCode Studio дает Вам возможность при нажатии кнопок – «Compile and Program» или «Program» тотчас же связывать шестнадцатеричный файл активной в настоящий момент программы, и который Вы собираетесь записать в микроконтроллер.

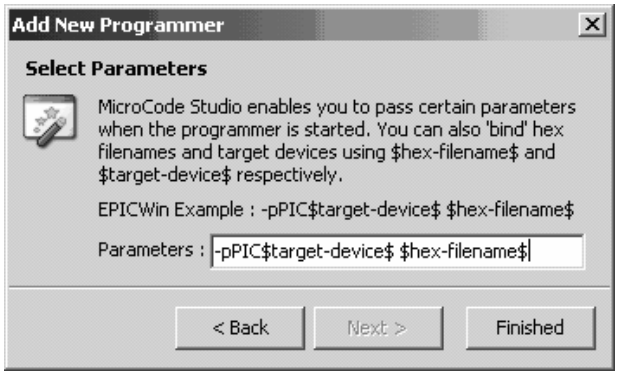


Рис.12. Окно командной строки

Например, если Вы компилируете файл blink.bas, используя микроконтроллер 16F628, то Вы хотели бы передать программатору файл blink.hex, а также название микроконтроллера, который Вы намереваетесь использовать. Вот – пример для EPICWin:

-pPIC$target-device$ $hex-filename$

Когда EPICWin запустится, имя микроконтроллера и шестнадцатеричное имя файла будут связаны и переданы программе программатора.

Пример:

-pPIC16F628 blink.hex

Интерактивная отладка программ (ICD)

Сейчас мы поговорим об очень полезном инструменте, присутствующем в пакете MicroCode Studio. Это The MicroCode Studio ICD или интерактивный внутрисхемный отладчик. Он позволяет Вам, подключив разрабатываемую схему к компьютеру через один из COM портов запрограммировать микроконтроллер, а затем, запустив программу в работу просматривать значения переменных, состояние регистров, памяти ЕЕПРОМ и памяти программ.

Для того чтобы воспользоваться интерактивным отладчиком, вам необходимо ввести в схему небольшое добавление, с помощью которого она будет осуществляться связь с компьютером. Этот узел показан на рисунке 13. В него входит микросхема драйвер COM порта – MAX232, несколько резисторов и конденсаторов. Также Вы должны уточнить в справочных данных на используемый микроконтроллер, какие выводы данного микроконтроллера используются для связи по протоколу USART.

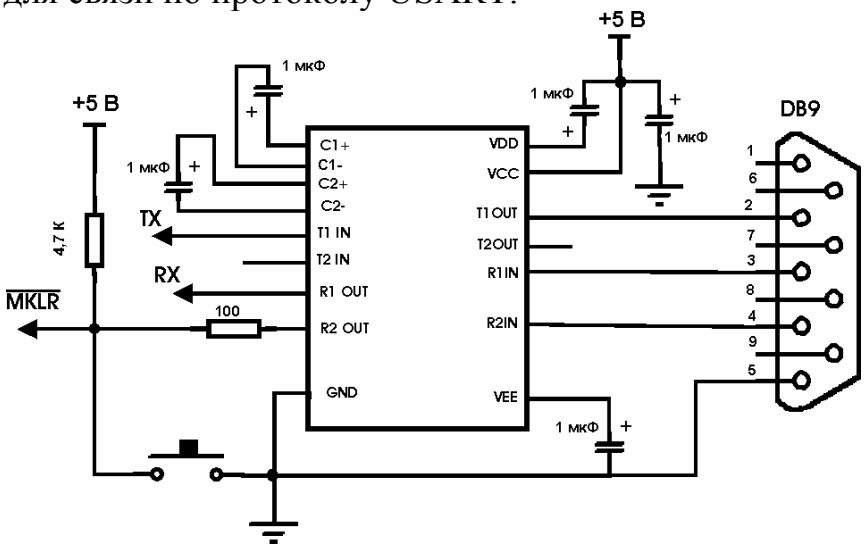


Рис. 13. Схема включения микросхемы драйвера MAX232

В случае отсутствия микросхемы-драйвера Вы можете собрать более простую схему на дискретных компонентах. Эта схема приведена на рисунке 14.

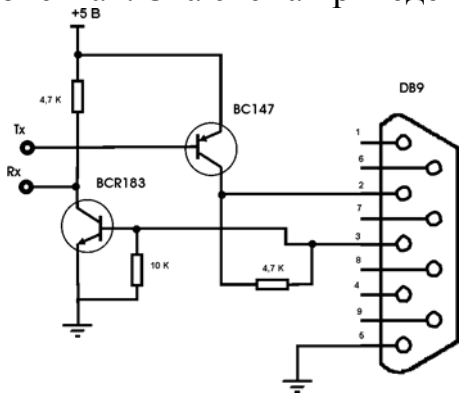


Рис. 14. Схема драйвера COM-порта построенного на дискретных компонентах.

Необходимо отметить, что облегченная версия программы MicroCode Studio поддерживает интерактивную отладку программ только с контроллером 16F628. Полная версия, которая называется MicroCode Studio Plus поддерживает следующие микроконтроллеры**:** 16F627(A), 16F628(A), 16F73, 16F74, 16F76, 16F77, 16F870, 16F871, 16F873(A), 16F874(A), 16F876(A), 16F877(A), 16F87, 16F88, 18F242, 18F248, 18F252, 18F258, 18F442, 18F448, 18F452, 18F458, 18F1220, 18F1320, 18F2220, 18F2320, 18F4220, 18F4320, 18F6620, 18F6720*, 18F8620 и 18F8720. Устройства, отмеченные «*» поддерживают только первые 64 кбайт программы. В демонстрационной версии функция ICD не поддерживается вообще.

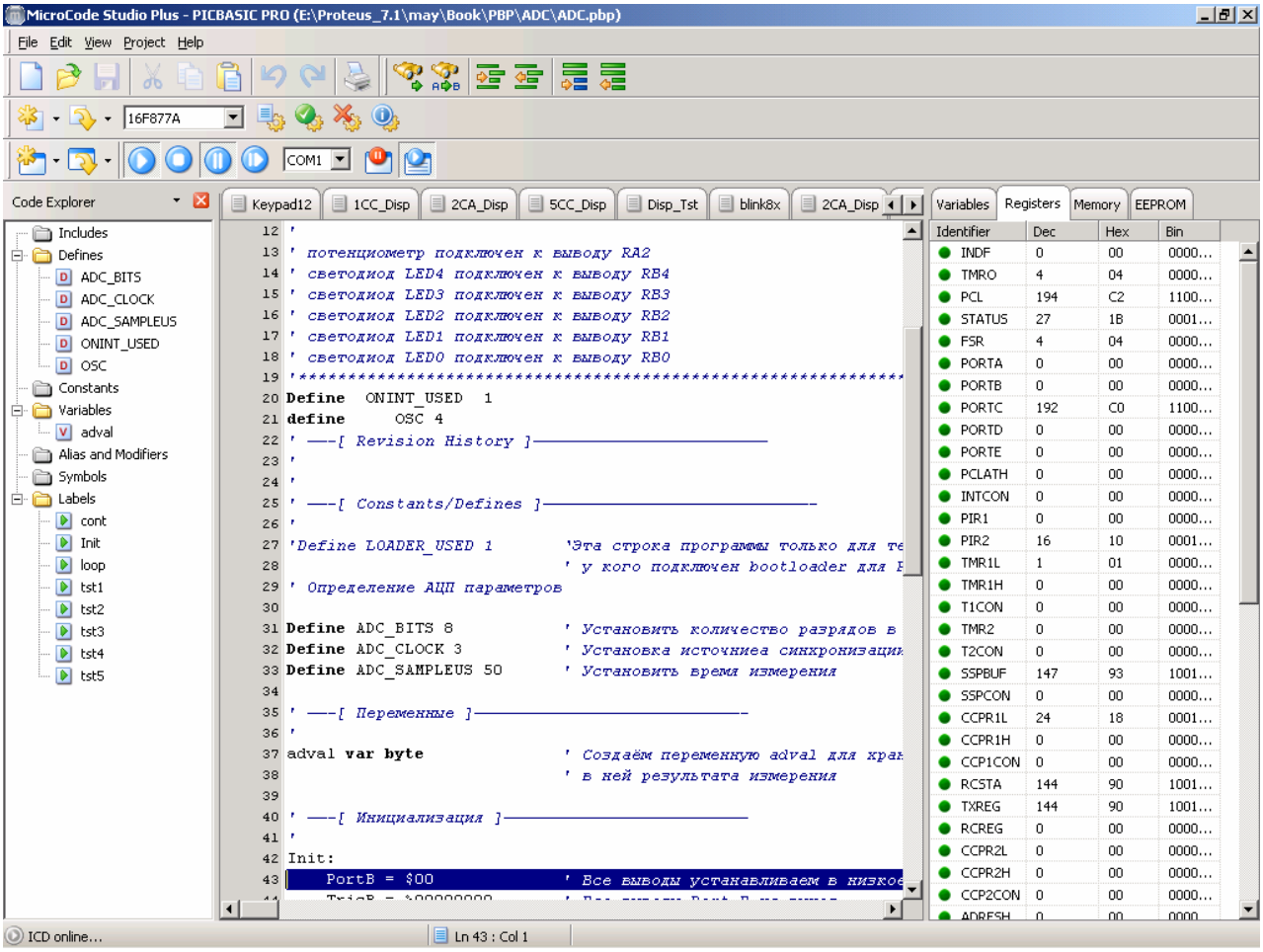


Рис. 15. Окно интерактивной отладки (ICD).

Для того чтобы работать с функцией ICD, необходимо с помощью обычного программатора записать программу – загрузчик в микроконтроллер. Только при наличии этой программы микроконтроллер «знает», как ему общаться с компьютером в этом режиме. Код этой программы находится в папке C:\Program Files\Mecanique\MCSP\MCLoader либо в папке C:\Program Files\Mecanique\MCSP\MCLoader\LoaderHEX. После того, как Вы собрали и

подключили интерфейсный узел к компьютеру и к разрабатываемой схеме, записали в микроконтроллер программу – загрузчик, можете приступать к отладке Вашей программы. Прежде всего, Вы должны указать программе порт, через который Вы будете осуществлять связь с Вашей схемой. После того, как Вы ввели текст вашей программы в окне редактора, Вы должны будете скомпилировать ее и записать в микроконтроллер. Для этого воспользуйтесь либо кнопкой «ICD Compile», либо кнопкой «ICD Compile and Program». То же самое Вы можете проделать, воспользовавшись основным меню: Project > ICD Compile либо Project > ICD Compile and Program. Перед записью Вашей программы в микроконтроллер программа, скорее всего, попросит Вас осуществить сброс Вашего микроконтроллера (Please reset the target microcontroller in order to access the bootloader process..). После того как будет нажата кнопка «сброс», начинается запись программы в память микроконтроллера и затем ее верификация. Если компиляция и запись программы в микроконтроллер прошли успешно, можно приступать к отладке программы. Нажмите для этого кнопку ICD Run. При этом окно основной программы изменится и примет вид изображенный рисунке 15.

В основном окне редактирования выполняемая строка программы будет подсвечена, синим цветом, а в появившемся справа окне будут выведены значения переменных, регистров и памяти. Вы можете в любой момент приостановить выполнение программы, нажав на кнопку «ICD Pause», а затем продолжить выполнение программы. Вы можете запустить выполнение программы в пошаговом режиме. Для этого служит кнопка «ICD Step». Вы можете в тексте программы установить контрольные точки, на которых выполнение программы будет остановлено. Для того чтобы установить контрольную точку, надо установить курсор на нужную строку и нажать кнопку «Toggle Breakpoint» либо щелкнуть курсором мышки на сером поле рядом с выбранной строкой программы. Выбранные контрольные точки подсвечиваются красным цветом. Но не на всех контрольных точках выполнение программы будет останавливаться. Такие точки во время выполнения программы подсвечиваются серым цветом. Так, например, если Вы установите контрольную точку в области комментария, программа не будет останавливаться на этой точке. Если Вы, выбрав контрольную точку, щелкните по ней правой кнопкой мышки, то в выпавшем контекстном меню Вы сможете выбрать пункт «Properties Breakpoint» (Свойства контрольной точки). В появившемся окне Вы можете задать количество проходов программы, когда данная точка остановит выполнение программы. Значение 0 говорит о том, что программа будет останавливаться каждый раз, доходя до этой точки. У интерактивного отладчика есть еще одна важная особенность - это возможность менять значения переменных, регистров и памяти в процессе работы анимации. Чтобы это сделать, в момент анимации просто щелкните мышкой по интересующему Вас значению, в нижнем справа окне введите новое значение и нажмите клавишу «ENTER». Только не забывайте что, вводя шестнадцатеричное значение, прежде ставьте префикс «$», вводя двоичное значение префикс «%».

Программа MicroCode Loader.

Программа MicroCode Loader предназначена для загрузки кода программы в микроконтроллер, чтения кода записанного в память, проверки записанного кода, стирания памяти микроконтроллера и выдачи информации об используемом микроконтроллере. Для того чтобы загрузить программу MicroCode Loader Вы должны в основном меню выбрать View > Loader. Перед Вами откроется окно представленное на рисунке 16.

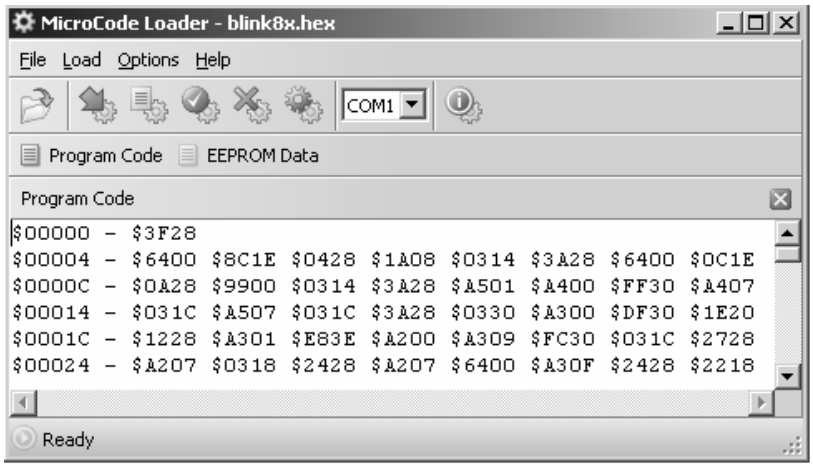


Рис. 16. Окно программы MicroCode Loader

В верхней части окна, как обычно, расположена строка основного меню. Ниже этой строки расположена панель инструментов, в которой Вы сможете найти различные кнопки. Кнопка – Open; с ее помощью вы сможете открыть нужный Вам *.hex файл. Кнопка – Program; нажатие на которую приведет к записи программы в микроконтроллер. Кнопка – Read; предназначена для считывания кода из микроконтроллера. Кнопка – Verify; служит для сравнения записанного кода с первоисточником. Кнопка – Erase; предназначена для очистки памяти микроконтроллера. Кнопка – Run User Code отключает bootloader и запускает на исполнение Вашу программу. Последняя кнопка Information сообщает информацию об используемом загрузчике и название подключенного микроконтроллера. Ниже панели инструментов расположено окно кода программы с двумя закладками. По одной закладке Вы можете просмотреть код основной программы с которой Вы работаете, а на другой - код, записанный в ЕЕПРОМ микроконтроллера. Необходимо отметить, что для того, чтобы пользоваться этой программой, в микроконтроллер, как и в предыдущем разделе, должна быть загружена программа-загрузчик. Более подробную информацию об этом читайте в предыдущем разделе.

Терминал последовательного порта (The Serial Communicator).

Этот инструмент позволяет Вам организовать и контролировать обмен информацией по последовательному каналу связи между компьютером и Вашим

устройством. Для запуска программы терминала последовательного порта Вы должны в основном меню выбрать VIEW > SERIAL COMMUNICATOR либо нажать кнопку F4. Перед Вами откроется окно представленное на рисунке 17.

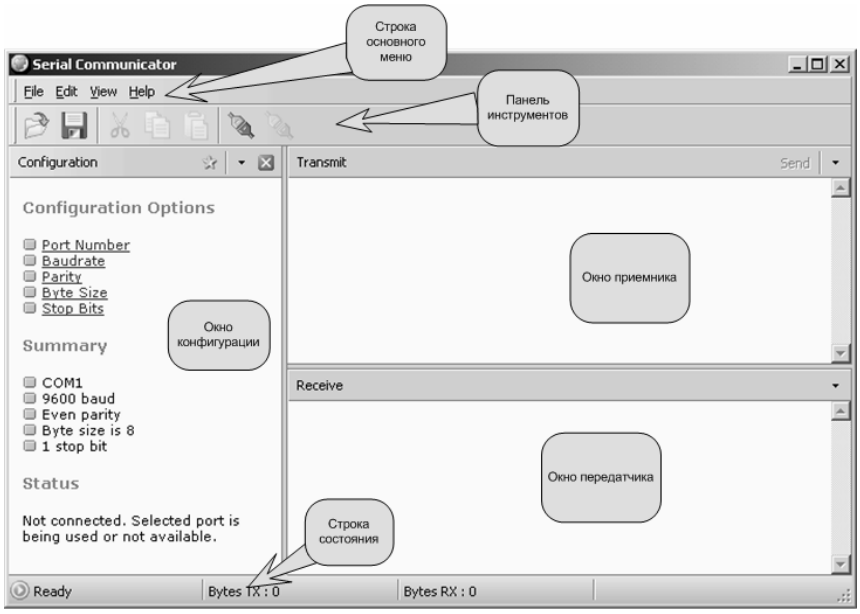


Рисунок 17. Окно программы терминала последовательного порта.

В верхней части открывшегося окна мы видим строку основного меню. Ниже расположена панель инструментов. Еще ниже и слева окно конфигурации, а справа окна приемника и передатчика. Внизу расположена строка состояния.

Прежде чем начать работу по этому каналу связи, Вы должны сконфигурировать Ваш порт для обмена данными по последовательному каналу и установить правильный протокол связи. Для этого в окне конфигурации Вы выбираете номер последовательного порта (COM1, COM2 или COM3). Затем устанавливаете скорость обмена данными в бодах (300 – 115200). В следующем пункте Вы должны будете указать компьютеру на то, будет ли осуществляться проверка на четность (нечетность) или нет. Затем Вы указываете на количество бит в информационной посылке (7 или 8) и количество стоповых разрядов в посылке. Затем вы можете устанавливать связь с Вашим устройством. Для этого Вы должны нажать в панели инструментов кнопку CONNECT либо клавишу F9. При этом окно программы изменит свой вид, пропадет окно конфигурации, и останутся только окна приемника и передатчика.

В окне передатчика Вы можете набрать текст команды и послать её по последовательному каналу связи, внешнему устройству. В дополнение к текстовым данным, Вы также можете послать и управляющие символы. Для того чтобы отобразить список вариантов передачи, щелкните правой кнопкой мышки в окне передатчика, и перед Вами откроется контекстное меню. В этом окне Вы можете выбрать удобные для Вас опции обмена информацией.

Программа Easy HID Wizard

Еще одним программным продуктом, включенным в состав MicroCode Studio Plus, является программа Easy HID (Human Interface Device) Wizard. Эта программа позволяет Вам обмениваться данными между компьютером и Вашим устройством по шине USB. При этом Вам не надо будет устанавливать специальный драйвер, он уже есть в системе. К сожалению, эта программа работает пока только с микроконтроллерами 18F2455, 18F2550, 18F4455 и 18F4550.

Язык PicBasicPro

Программа написанная на языке PicBasicPro представляет собой набор операторов и команд, которые последовательно выполняются и решают определенную задачу. С помощью операторов PBP можно производить вычисления, хранить промежуточные результаты, осуществлять те или иные действия в зависимости от состояния созданных в программе объектов. Можно также выводить информацию на экран буквенно-цифрового ЖК монитора в текстовом виде, обрабатывать действия пользователя, управлять памятью и реагировать на всевозможные исключительные ситуации и ошибки, которые могут возникнуть в ходе выполнения программы.

Предлагаемое описание не может быть полным трактатом о языке BASIC. Здесь будет описаны система команд компилятора PicBasicPro и будут приведены примеры того, как их использовать. При этом необходимо помнить, что составить серьезную программу достаточно сложно. Для этого надо хорошо знать возможности языка, выучить основные его операторы и, самое главное, постоянно практиковаться в написании новых и новых программ.

После того как мы выполнили все настройки, мы можем приступить к изучению языка PicBasic

Основные понятия и определения. Идентификаторы.

Для обозначения меток строк и имен переменных в PICBasic как и в большинстве других языков программирования используются идентификаторы. Идентификатор - любая последовательность символов, цифр и символов подчеркивания, но он не должен начинаться с цифры. Нельзя также использовать символы кириллицы. Но если использовать в качестве идентификатора русское слово записанное латинскими буквами то этого компилятор не заметит.

Идентификаторы не чувствительны к регистру, таким образом label, LABEL и Label - всё будет обработано компилятором одинаково. Хотя идентификатор Вы можете написать любой длины, PICBasic распознает только первые 32 символа.

Комментарии.

Программу очень полезно сопровождать комментариями. Они нужны для того, чтобы в дальнейшем сразу понимать, что происходит в различных ее частях. Причем эти комментарии необходимы не только для стороннего читателя, но и для Вас самих**,** поскольку через некоторое время многое может забыться. Комментировать нужно не операторы BASIC, а выполняемое ими действие. Например, оператор

```
Speed = 60
```

правильно снабдить не очевидными словами “в переменную Speed записывается число 60”, а осмысленным текстом: “значение скорости делаем равным 60 км/час”.

Комментарии желательно располагать на той же самой строке, где находится и описываемый ими оператор. Он начинается с символа «'» (одиночная кавычка), либо с символа «;» (точка с запятой), либо оператора REM. Все символы, следующие за знаком комментария и до конца текущей строки, компилятором просто игнорируются.

```
Speed = 60  ' Значение скорости делаем равным 60 км/ч
```

Комментарий можно начинать и с первой позиции:

```
' —— вся эта строка - комментарий ——
```

Вместо символа «'» в первой позиции может использоваться оператор REM. Перед ним в строке могут идти только пробелы.

Кроме того, при записи операторов BASIC в тех местах, где допускаются пробелы, их можно использовать в неограниченном количестве, как правило, с целью повышения удобочитаемости исходного текста программы. Например:

```
Speed    =    60    ' пробелы хороши в меру
```

Переменные.

Переменная — ключевое понятие любого языка программирования. Переменные используются для хранения промежуточных значений, получаемых в процессе выполнения программы. Чтобы программа понимала, какие данные хранятся в той или иной переменной, они делятся на типы. В PICBasicPro существуют три типа переменных – BIT, BYTE, WORD. Из названия типов ясно, что в памяти микроконтроллера эти переменные будут занимать соответственно 1, 8, 16 разрядов. Переменные конкретного типа могут содержать только корректную, соответствующую этому типу информацию. При попытке записать, например, в байтовую переменную шестнадцатиразрядное значение, компилятор сообщит об ошибке и прервет работу.

Для того чтобы добавить переменную в программу используется ключевое слово VAR. Переменные могут быть битовыми (BIT), байтовыми (BYTE) или двухбайтовыми (WORD). Пространство для каждой переменной компилятор автоматически распределяет в оперативной памяти микроконтроллеров. При определении переменной используется следующий формат записи:

```
Label VAR Size,
```

где

Label - имя переменной в качестве которого Вы можете использовать любой идентификатор, исключая зарезервированные слова.

. Size – тип переменной **(**BIT, BYTE или WORD).

Например:

```
dog VAR byte
cat VAR bit
w0 VAR word
```

В качестве имени переменной может использоваться любая комбинация английских букв и цифр (без пробелов), начинающаяся с буквы. Большинство операторов BASIC записываются с помощью специальных слов (в данном случае — VAR), пробелы внутри которых не допускаются. Эти слова называются ключевыми или зарезервированными. Использовать ключевые слова в качестве названий переменных или в любых других целях, помимо их прямого назначения, запрещено.

Псевдонимы.

Оператор **VAR** может также использоваться для того, чтобы создать псевдоним (другое название) для переменной. Использование псевдонимов может быть полезно при обращении к внутренним частям переменной.

```
Label VAR Label1 {.Modifiers},
```

где

Label - любой идентификатор или имя псевдонима, исключая ключевые слова.

Label1 – имя ранее определенной переменной.

Modifiers – модификатор с помощью которого Вы выделяете часть переменной. Список используемых модификаторов приведен в таблице 4.

Примеры:

```
fido VAR dog        ' fido другое название для переменной dog
b0 VAR w0.byte0     ' b0 - первый байт слова w0
b1 VAR w0.byte1     ' b1 - второй байт слова w0
flea VAR dog.0      ' flea это нулевой бит переменной dog
```

Таблица 4.

Модификатор	Описание
BIT0 или 0	Создаёт псевдоним для 0 разряда байта или слова

BIT1 или 1	Создаёт псевдоним для 1 разряда байта или слова
BIT2 или 2	Создаёт псевдоним для 2 разряда байта или слова
BIT3 или 3	Создаёт псевдоним для 3 разряда байта или слова
BIT4 или 4	Создаёт псевдоним для 4 разряда байта или слова
BIT5 или 5	Создаёт псевдоним для 5 разряда байта или слова
BIT6 или 6	Создаёт псевдоним для 6 разряда байта или слова
BIT7 или 7	Создаёт псевдоним для 7 разряда байта или слова
BIT8 или 8	Создаёт псевдоним для 8 разряда слова
BIT9 или 9	Создаёт псевдоним для 9 разряда слова
BIT10 или или 10	Создаёт псевдоним для 10 разряда слова
BIT11 или 11	Создаёт псевдоним для 11 разряда слова
BIT12 или 12	Создаёт псевдоним для 12 разряда слова
BIT13 или 13	Создаёт псевдоним для 13 разряда слова
BIT14 или 14	Создаёт псевдоним для 14 разряда слова
BIT15 или 15	Создаёт псевдоним для 15 разряда слова
BYTE0 или LOWBYTE	Создаёт псевдоним для младшего байта слова
BYTE1 или HIGHBYTE	Создаёт псевдоним для старшего байта слова

Константы.

Представьте, что Вы пишете большое приложение, выполняющее сложные математические расчеты, в ходе которых неоднократно приходится прибегать к использованию одного и того же числа, несущего определенную смысловую нагрузку (например, числа 0,001, задающего точность вычислений). После того как программа разработана и проверена на контрольных примерах, вполне может возникнуть желание использовать ее в более широких целях. При этом, в частности, наверняка потребуется изменить величину точности вычислений с более высокой на более низкую или наоборот. Но тогда придется искать все операторы, где встречается это число, и каждый раз менять его на новое значение вручную. При этом легко и ошибиться, введя лишний или не дописав ноль.

Есть хороший способ — всем числовым или строковым константам, которые встречаются в программе более одного раза, присвоить имена также как и переменным. Для этого предназначен оператор **CON**.

```
porog CON 0.003
```

Размещаются операторы описания констант обычно в самом начале программы.

Теперь, если потребуется изменить значение уровня точности, достаточно будет только поменять одно число в одном месте исходного текста. Переменные данные не могут быть сохранены в константе.

Для записи числовых констант в PicBasicPro используются три формы: десятичное число, двоичное и шестнадцатеричное. Двоичные значения определяются с использованием префикса «%» а шестнадцатеричные значения с использованием префикса «$». Десятичные значения - по умолчанию не требуют никакого префикса.

Примеры:

```
100       'десятичное число
%100      'двоичное число
$100      'шестнадцатеричное число
```

В языке PicBasicPro не предусмотрен такой тип переменных, как строковые переменные, но, тем не менее, строковые переменные могут использоваться в некоторых командах. Строковая переменная содержит один или более буквенно-цифровых символов и заключена в кавычки.

Пример:

```
LCDOUT "Hello"     'Вывести на экран ЖК монитора слово Hello
```

Строковые переменные обычно обрабатываются как список индивидуальных символьных значений.

Символы.

Для совместимости с BasicStamp в PicBasicPro был введен оператор **SYMBOL**. Он обеспечивает еще один метод для совмещения имен переменных и констант. Оператор **SYMBOL** не может использоваться для того чтобы создавать переменные.

Пример:

```
SYMBOL lion = cat 'cat был предварительно создан с использованием
оператора VAR
SYMBOL mouse = 1        'тоже самое что  - mouse = 1
```

Метки.

Для того, чтобы в программе можно было отметить те места, куда программа должна перейти в результате выполнения какого либо условия, существуют

метки. В отличие от многих ранних версий BASIC, PicBasicPro не требует нумерации строк. Вместо этого любая строка может начаться с метки. В данном случае она является просто идентификатором. Заканчивается метка двоеточием.

```
LAB:
     PRINT "Hello World"
     GOTO LAB
```

Имена меток могут быть длиной до 32 символов и могут содержать любые алфавитно-цифровые знаки, но они не должны начинаться с цифры. Например:

```
LABEL1:
```

является совершенно правильным, однако выражение:-

```
1LABEL:
```

произведет синтаксическую ошибку, потому что метка начинается с цифры - 1. Символы подчеркивания также разрешаются как часть знаков метки. Это помогает создавать более ясные имена меток. Например:-

```
THISISALABEL:
```

Эта метка менее понятна чем:

```
THIS_IS_A_LABEL:
```

Массивы.

Одним из основных понятий в программировании является понятие массива.

Массив — это набор однородных данных (BIT, BYTE или WORD), имеющий имя и последовательную нумерацию его элементов. Если мы знаем, что в программе предстоит работать с большим объемом каких-то данных, то мы должны объявить этот массив в программе. Массивы переменных могут быть созданы, так же как и переменные.

```
Label VAR Size[Number of elements]
```

Где

Label - любой идентификатор, исключая ключевые слова, как описано выше.

Size – формат переменных массива (BIT, BYTE или WORD).

Number of elements – количество элементов массива. Примеры создания массивов:

```
sharks VAR byte[10]
fish VAR bit[8]
```

Первый элемент массива - элемент 0. В массиве рыбы, определенном выше, элементы пронумерованы от fish [0] до fish [7] всего приведено 8 элементов. Поскольку массивы распределены в памяти, есть пределы размера для каждого типа:

Таблица зависимости объема занимаемой памяти массивом от типа массива

Таблица 5

Тип	Максимальное количество элементов
BIT	256
BYTE	96*
WORD	48*

- Зависит от микроконтроллера.

Математические операторы.

В PicBasicPro как и в любом другом языке программирования необходимо производить математические вычисления. Для этих целей используются математические операторы. К математическим операторам мы здесь относим как обычные арифметические операторы, так и тригонометрические операторы, операторы сравнения, логические операторы, операторы сдвига. Ниже приведен полный список математических операторов для версии компилятора 2.47.

Замечание: Все математические операторы выполняются для операндов без знака и с 16 разрядной точностью.

Таблица математических операторов языка PicBasiPro

Таблица 6

Математический оператор	Описание
=	Оператор присваивания
+	Сложение
-	Вычитание
*	Умножение
**	Старшие 16 разрядов результата умножения
*/	Средние 16 разрядов умножения
/	Деление

//	Остаток (Модуль)
<<	Сдвиг влево
>>	Сдвиг вправо
ABS	Абсолютное значение
COS	Косинус
DCD	2n декодер
DIG	Цифра
DIV32	Деление 32 разрядного числа
MAX	Максимум
MIN	Минимум
NCD	Енкодер
REV	Реверсирование разрядов
SIN	Синус
SQR	Квадратный корень
&	Поразрядное И
|	Поразрядное ИЛИ
^	Поразрядное исключающее ИЛИ
~	Поразрядное НЕ
&/	Поразрядное НЕ И
|/	Поразрядное НЕ ИЛИ
^/	Поразрядное НЕ исключающее ИЛИ

Итак, компьютер умеет вычислять элементарные арифметические выражения. Но для того, чтобы он смог это сделать, мы должны представить это самое выражение в понятном ему виде, а именно:

1. В отличие от арифметики, выражение должно быть записано в одну строку безо всяких числителей и знаменателей;
2. Для записи арифметических действий допустимо использовать только перечисленные ниже знаки. Недопустим пропуск знака умножения между коэффициентом и переменной, как это возможно в алгебре (например, нельзя писать 2x, а надо 2*X, или нельзя 5d, а надо 5*D);
3. Чтобы компьютер вычислил выражение правильно, необходимо помнить о приоритете выполнения действий. Тут все как в элементарной математике, сначала выполняются действия в скобках (в BASIC в математических операциях скобки используются только круглые, в

сложных выражениях они могут быть двойные, тройные и т. д.). Далее вычисляются функции, если они есть. Потом умножение и деление. И, наконец, в последнюю очередь - сложение и вычитание;

4. Действия одинаковой очередности выполняются слева направо.

Теперь поговорим о каждом из этих операторов отдельно.

Умножение.

Оператор «*» возвращает младшие 16 разрядов 32-битового результата. Это - типичное умножение, характерное для большинства языков программирования.

Оператор «**» возвращает старшие 16 разрядов 32-битового результата.

Эти два оператора могут использоваться в соединении, чтобы выполнить полное умножение двух шестнадцатиразрядных чисел, в результате перемножения которых выдается 32-разрядный результат.

Пример:

```
W1 = W0 * 1000     ' Умножить  W0 на 1000 а результат(младшие 16
‘ разрядов) записывается в W1
W2 = W0 ** 1000   'Умножить W0 на 1000 а результат (старшие 16
‘ разрядов, которые могут быть  и 0) записывается в W2
```

Оператор - * / возвращает средние 16 разрядов 32-разрядного результата.

```
W3 = W1 */ W0  'Умножить W1на  W0 результат(срединные 16 разрядов 32
‘ разрядного числа) поместить в W3
```

Деление.

Оператор – «/» возвращает 16-разрядный результат деления одного 16-разрядного числа на другое 16-разрядное число. В то же самое время оператор – «//» возвращает остаток от деления 16-разрядных чисел. Иногда это называется - модуль числа.

Пример:

```
W1 = W0 / 1000  ‘Разделить  W0 на 1000 и результат поместить в W1
W2 = W0 // 1000 ‘Разделить  W0 на 1000 и остаток  поместить в W2
```

Операторы сдвига.

Операторы «<<» и «>>» сдвигают влево или вправо двоичное значение числа на указанное количество разрядов, т.е. от 0 до 15.

Пример:

```
B0 = B0 << 3  'Сдвинем двоичное значение переменной B0 на 3 разряда влево (все равно что умножить на 8)
W1 = W0 >> 1  'Сдвинем двоичное значение переменной W0 на 1 разряд вправо и результат запишем в W1 (все равно что разделить на 2),
```

Объясним все это, как говорится, на пальцах. Пусть мы имеем число 1000 или в двоичном представлении 1111101000, тогда операция 1000<<3 выдаст результат 1111101000000 или 8000 (1000*8), а операция – 1000>>2 выдаст – 11111010 или 250(1000/4).

Оператор ABS.

Оператор ABS возвращает абсолютное значение числа. Если Вы имеете дело с переменной размером в байт и она будет больше чем 127 (старший разряд =1), то тогда результатом команды ABS будет разность = 256 - значение. Если же Вы имеете дело с переменной размером в слово и она будет больше чем 32767 (старший разряд =1), то тогда результатом команды ABS будет разность = 65536 - значение.

Пример:

```
B0 VAR BYTE
B0 = 200
B1 = ABS B0          ' Результатом этой команды будет 50 (255-200)
```

Оператор COS.

Оператор COS возвращает 8-разрядное значение косинуса угла. Результат представляется в дополнительном коде. Это значит, что восьмой разряд результата в двоичном представлении говорит о знаке результата. Если этот разряд равен 0, то результат положительный, а если равен единице – отрицательный (модуль отрицательного результата может быть получен с использованием команды **ABS**). В PicBasicPro значения аргумента и значения функции отличаются от значений принятых в обычной математике. Так в PicBasicPro полный угол для функции COS равен 255 в двоичных радианах, в отличие от обычных 360 градусов. Для перевода градусов в двоичные радианы

воспользуйтесь формулой α*255/360 где α - угол в градусах. А максимальное и минимальное значение функции лежит в интервале от +127 до -127 (в обычной математике этот интервал от +1 до -1).

Пример:

```
B1 = COS B0
```

Оператор DCD.

Оператор DCD дешифрирует значение десятичного числа таким образом, что в результате образуется двоичное число, все разряды которого равны 0 кроме разряда, порядковый номер которого равен дешифрируемому числу. Это десятичное число может быть любым от 0 и до15.

Пример:

```
B0 = DCD 2 'Переменная B0 равна %00000100
```

Оператор DIG.

Оператор DIG возвращает значение цифры десятичного числа. Просто укажите порядковое значение нужной цифры (0 - 4 где 0 является самой правой цифрой) значение которой Вы бы хотели выбрать.

Пример:

```
B0 = 123            'Переменная B0 равняется 123
B1 = B0 DIG 1       'Переменная B1 есть вторая цифра десятичного
' значения переменной B0 (т.е. 2)
```

Оператор DIV32.

Синтаксис:

```
DIV32 Number
```

Функция умножение – «*» в PicBasicPro работает как произведение двух 16-разрядных значений, приводящее к 32-разрядному результату внутри компилятора. Но, так как компилятор выполняет математические операции с переменными, максимальный размер которых 16 разрядов, то правильный результат получается в два этапа. То есть вначале выполняется функция «*», где результатом являются младшие 16 разрядов, а затем функция «**», где в результате возвращаются старшие 16 разрядов. И нет другого пути, чтобы получить доступ к 32-разрядному результату в один прием.

Во многих случаях желательно иметь возможность разделить 32- разрядный результат умножения на 16-разрядное число для того, чтобы вычислить среднее значение или провести некоторые измерения. С этой целью и была добавлена

новая функция, названная **DIV32**. Работа функции **DIV32** фактически ограничена делением 31-разрядного целого числа без знака (максимальное значение 2147483647) на 15-разрядное целое число без знака (максимальное значение 32767). Но поскольку компилятор во всех случаях работает только с 16-разрядными переменными, то функция **DIV32** должна начинать работать сразу же, как только выполнено умножение и внутренняя переменная все еще содержит 32-разрядный результат умножения. При этом никакая другая операция не должна произойти между операцией умножения и **Div32**, так, в противном случае, внутренняя переменная может быть изменена, уничтожив 32-битный результат умножения. Это означает, между прочим, что в это время и функции обработки прерывания должны быть отключены перед операцией умножения и до окончания работы функции **DIV32**. Прерывания же на ассемблере не влияют на внутренние переменные, и таким образом они могут использоваться вне зависимости от использования функции **DIV32**.
Пример:

```
a VAR WORD
b VAR WORD
c VAR WORD
dummy VAR WORD
b = 500
c = 1000
DISABLE          'Необходимо отключить, если используются прерывания
dummy = b * c    ' Можно было бы также использовать ** или */
a = DIV32 100
ENABLE           ' Включаем обработку прерываний
```

В этом примере переменной b присваивается значение 500, а переменной c - значение 1000. Если эти два числа перемножить, то результат был бы 500000. Но это число превышает максимально возможный размер переменных в PicBasicPro т.е. - WORD размер, которой 16-разрядов (65535). Таким образом, перемножив эти два числа и выведя результат, мы бы получили только младшие 16 разрядов. Однако функция **Div32** предоставляет нам доступ к внутренней 32 разрядной переменной компилятора как к операнду. В этом примере 32-bit результат умножения (b*c) делится *на* 100 с помощью функции **Div32**. В результате чего 16-разрядный результат деления (5000) сохраняет*ся* в переменной - a.

Операторы MAX и MIN.

Операторы MAX и MIN возвращают максимальное или минимальное значение соответственно, двух чисел. Это обычно используется для того, чтобы ограничить значения некоторой величиной.
Пример:

```
B1 = B0 MAX 100 'Сделать B1 как набольшее из двух значений B0 и 100 (B1
‘ примет значения от 100 до 255
B1 = B0 MIN 100 'Сделать B1 как наименьшее из двух значений B0 и 100
‘ (B1не может быть больше100)
```

Оператор NCD.

Оператор NCD возвращает порядковый номер старшего разряда шестнадцатиразрядного двоичного числа имеющего значение 1. Он используется для того, чтобы найти в значении самый старший разряд равный 1. Он возвращается в 0, если никакой разряд не равен 1.

Пример:

```
B0 = NCD %01001000 ' Переменная B0 равняется 7
```

Оператор REV.

Оператор REV полностью меняет значения разрядов двоичного числа на противоположное. Число разрядов, которые будут полностью изменены – может быть от 1 до 16. Отсчет изменяемых разрядов ведется, начиная с самых младших разрядов.

Пример:

```
B0 = %10101100 REV 4 'Преобразовать значение %10101100 в переменную
B0 равную %10100011
```

Оператор SIN.

Оператор SIN возвращает 8-разрядное значение косинуса угла. Результат представляется в дополнительном коде. Это значит, что восьмой разряд результата в двоичном представлении говорит о знаке результата. Если этот разряд равен 0, то результат положительный, а если равен единице – отрицательный (модуль отрицательного результата может быть получен с использованием команды **ABS**). В PicBasicPro значения аргумента и значения функции отличаются от значений принятых в обычной математике. Так в PicBasicPro полный угол для функции SIN равен 255 в двоичных радианах, в отличие от обычных 360 градусов. Для перевода градусов в двоичные радианы воспользуйтесь формулой α*255/360 где α - угол в градусах. А максимальное и минимальное значение функции лежит в интервале от +127 до -127 (в обычной математике этот интервал от +1 до -1).

Пример:

```
B1 = SIN B0
```

Оператор SQR.

Оператор SQR возвращает квадратный корень значения. Так как PicBasic работает только с целыми числами, результатом всегда будет 8-разрядное целое число, не большее чем фактический результат.

Пример:

```
B0 = SQR W1          ' B0  квадратный корень W1
```

Поразрядные операторы.

Одним из важных свойств языков высокого уровня есть возможность перехода к языкам низкого уровня типа ассемблер. Поразрядные операторы осуществляют доступ к регистрам и памяти МК на уровне отдельного разряда. Они действуют на каждый разряд значения булевым способом *и* могут использоваться для того, чтобы ‘обнулить’ какие либо разряды или установить в 1 нужные. Например,

```
B0 = B0 & %00000001   ' Обнулить разряд 0 переменной B0
B0 = B0 | %00000001   ' Установить в переменной  B0 разряд 0 в 1
B0 = B0 ^ %00000001   ' Осуществить реверсирование состояние разряда 0 ‘ в переменной B0
```

В следующей таблице представлены поразрядные операторы.

Таблица 7

ПОРАЗРЯДНЫЕ ОПЕРАТОРЫ	
Операторы	**Описание**
&	Логическое И
|	Логическое ИЛИ
^	Логическое Исключающее ИЛИ
~	Логическое НЕ (отрицание)
&/	Логическое НЕ И
|/	Логическое НЕ ИЛИ
^/	Логическое НЕ Исключающее ИЛИ

Операторы сравнения.

Операторы сравнения используются в утверждениях типа IF..THEN, для того чтобы сравнить одно выражение с другим. PicBasicPro поддерживает следующие операторы сравнения:

Таблица операторов сравнения

Таблица 8

Операторы сравнения	
Оператор сравнения	**Описание**
= или ==	равно
<> или ! =	Не равно
<	Меньше чем
>	Больше чем
<=	Меньше или равно
>=	Больше или равно

Логические операторы.

Логические операторы отличаются от поразрядных операторов. Они приводят к истинному или ложному следствию их действия. Значение 0 рассматривают как ложное. Любое другое значение рассматривается как истинное. Они по большей части используются в соединении с операторами сравнения в утверждениях IF..THEN. PicBasicPro поддерживает следующие логические операторы:

Таблица логических операторов

Таблица 9.

Логические операторы	
Логический оператор	**Описание**
AND или **&&**	Логическое И
OR или ||	Логическое ИЛИ
XOR или ^^	Логическое исключающее ИЛИ
NOT AND	Логическое НЕ И
NOT OR	Логическое НЕ ИЛИ
NOT XOR	Логическое НЕ исключающее ИЛИ

Пример**:**

```
IF (A == big) AND (B > mean) THEN run
```

Убедитесь, что Вы использовали круглые скобки для указания PicBasicPro на нужный порядок выполнения операций.

Некоторые замечания о стилях программирования.

Создание и сопровождение программы это искусство. Здесь хотелось бы привести несколько простых методик следование, которым могло бы помочь Вам стать художником в этом деле.

Комментарии. Старайтесь использовать как можно больше комментариев. Даже притом, что программу пишете Вы, и ее работа совершенно очевидна для Вас. Однако через большой промежуток времени Вы не сможете вспомнить ее идеи, не говоря о постороннем человеке. Конечно, комментарии могут занимать много места в исходном тексте программы, но они не занимают память микроконтроллера, а поэтому используйте их свободно. И это поможет Вам в дальнейшем.

Делайте так, чтобы комментарии говорили Вам о том, что делает программа. Комментарий вроде: "Установить Pin0 в 1" - просто объясняет синтаксис языка, но ничего не говорит о том, почему Вы это делаете. Однако такой комментарий: "Включить светодиод, когда батарея разряжена" мог бы сказать Вам многое.

Блоки комментариев в начале программы и перед каждой секцией могут подробно описать то, что может случиться, вместо коротких заявлений в конце строки. Но не увлекайтесь одними только блоками, используйте оба вида комментариев.

В начале программы обычно описывается то, для чего программа предназначена, кто написал ее и когда. Может быть, также полезно внести в список дату и информацию о проверке, определения, что подключено к каждому контакту микроконтроллера, на какой микроконтроллер она рассчитывалась и тому подобное.

Контакты и имена переменных. Объявляя названия контактов или присваивая имена переменным старайтесь делать так, чтобы это было чем-то более понятным, нежели просто Pin0 или B1**.** В дополнение к либеральному использованию комментариев, наглядные имена переменных и контактов могут значительно повысить удобочитаемость программы. Следующий фрагмент программы демонстрирует это:

```
BattLED var PORTB.0          'Светодиод «Батарея разряжена»
level   var byte             'Переменная в которой хранится уровень
' зарядки батареи

    If level < 10 Then       'Если уровень зарядки батареи низок,
        High BattLED 'Включить светодиод
    Endif
```

Метки. Метки также должны нести смысловую нагрузку. Название метки должно говорить о том, какая задача решается в подпрограмме начинающейся с

этой метки. Так как обычно строка или подпрограмма, к которой переходит программа, делают, что-то уникальное.

Оператор **GOTO**. Старайтесь не использовать слишком часто команду **GOTO**. Оператор **GOTO** является, может быть иногда необходимым, но злом, старайтесь минимизировать его использование в максимально возможной степени. Делайте так, чтобы Ваша программа логически не скакала, была бы более последовательной. Лучшей заменой команды **GOTO** для достижения цели может быть команда **GOSUB.**

Глава 3. Примеры программ.

Основной задачей данной книги является обучение читателя программированию микроконтроллеров на языке BASIC. До настоящего момента мы рассматривали теоретическую часть этого вопроса, но, как известно теория без практики это необходимое, но далеко не достаточное условие освоения какого либо предмета. Поэтому в этой части книги мы перейдем непосредственно к разработке программ для реальных целей. Во всех описанных проектах мы будем использовать микроконтроллер PIC16F877A. В этом микроконтроллере имеется все необходимое для решения наших задач. Хотя Вы можете взять и другие микроконтроллеры, имеющие в своем составе необходимые аппаратные средства. Но только не забывайте в панели MicroCode Studio в окне выбора микроконтроллера выбрать соответствующий тип контроллера.

Начнем мы с самого простого примера, который часто называют «Hello Word».

Пример № 1 – Мигающий светодиод («Hello Word»).

Наша первая программа будет включать и выключать светодиод, подключенный к одному из выводов микроконтроллера (PORTB.0). Таким образом, мы сделаем нечто типа маяка, который будет работать так долго, пока это кому-то не надоест. На Рис. 18 приведена схема этого устройства. Выбрали мы этот микроконтроллер (достаточно мощный) не потому, что будем решать архи-сложную задачу, а только потому, что его можно использовать во всех примерах этой книги. А зачем нам покупать несколько разных контроллеров, когда все примеры мы можем изучить на одной этой микросхеме.

А вот и текст нашей первой программы:

```
Symbol LED = PORTB.0  ‘ Присвоим выводу  PORTB.0 имя LED
TRISB = %11111110     ‘ Установим вывод B.0 на выход, а остальные на
‘ вход
main:                 ‘ Метка начала основной программы
```

```
LED = 1              ' Включаем светодиод
Pause 500            ' Пауза в 0,5 секунды
LED = 0              ' Выключаем светодиод
Pause 500            ' Пауза в 0,5 секунды
goto main            ' Возврат на начало программы (цикл замыкается)
```

В первой строке нашей программы выводу микроконтроллера, к которому подключен светодиод (PORTB.0), мы присваиваем символ LED. Вообще этого можно и не делать и везде, где далее в программе встречается символ LED, писать полностью название вывода. Но, согласитесь, что это будет не так наглядно. Во второй строке программы мы указываем компилятору, что мы устанавливаем все выводы PORTB на ввод информации (хотя в нашем случае это, вообще говоря, безразлично), а вывод B.0 на вывод. Как было написано ранее, регистр TRIS управляет направлением передачи информации соответствующего порта. Значение порта здесь записано в двоичной системе, об этом указывает модификатор «%». Кстати, для того, чтобы легче понимать, в какое состояние мы устанавливаем вывод порта, запомните такую подсказку 1- Input (ввод), а 0 – Output (вывод). Для того чтобы установить контакт B.0 на выход, можно было бы воспользоваться оператором **OUTPUT** т.е.:

```
OUTPUT PORTB.0
```

Но такая запись мне не кажется проще. Особенно, когда нам надо одновременно переключить состояние нескольких контактов порта. С помощью непосредственного управления регистром TRIS мы это можем сделать одной строкой в программе.

В третьей строке стоит метка main, которая указывает начало нашей программы. В следующей строке мы устанавливаем на выводе PORTB.0 высокий логический уровень (1), а это означает, что светодиод начинает светиться. Затем мы выдерживаем паузу в течение 0.5 секунды для того, чтобы все успели это заметить. В шестой строчке программы мы выключаем светодиод, подавая на него низкий логический уровень (0). И затем вновь выдерживаем секундную паузу. Последняя строка нашей программы возвращает программу на начало, т.е. на метку main. Таким образом, программа будет работать циклически до тех пор, пока это Вам не надоест.

В этом первом примере все просто и понятно. Теперь Вы можете написать ее в редакторе MicroCode Studio и затем откомпилировать. Полученный в результате компиляции файл с расширением *.hex записать в микроконтроллер и спаять схему.

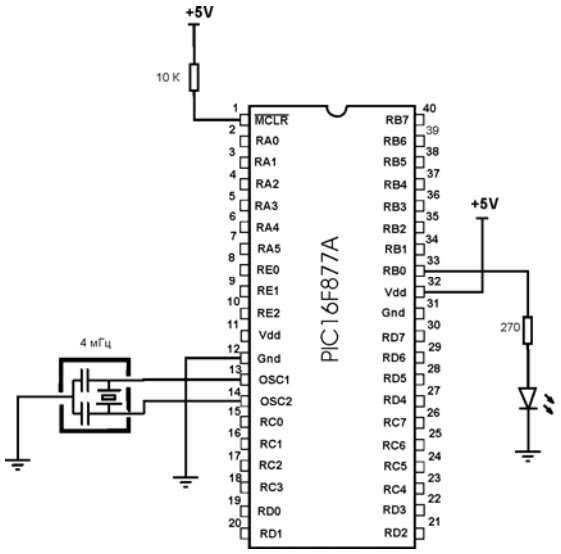


Рис. 18. Схема к примеру «Hello Word».

Замечание. Перед началом компиляции не забудьте в редакторе установить в отведенном для этого окне тип микроконтроллера, используемого в Вашем случае. Если же Вы собираетесь использовать кварцевый резонатор на другую частоту, то поместите в начале Вашей программы строчку:

```
DEFINE  OSC    X
```

Где вместо X надо будет указать частоту Вашего кварца. По умолчанию компилятор все делает для частоты 4 МГц.

Пример № 2. Работа с несколькими светодиодами.

Этот пример в большей степени основан на предыдущем примере. Добавим в предыдущую схему еще семь светодиодов и соберем ее, так как это показано на рис. 19.

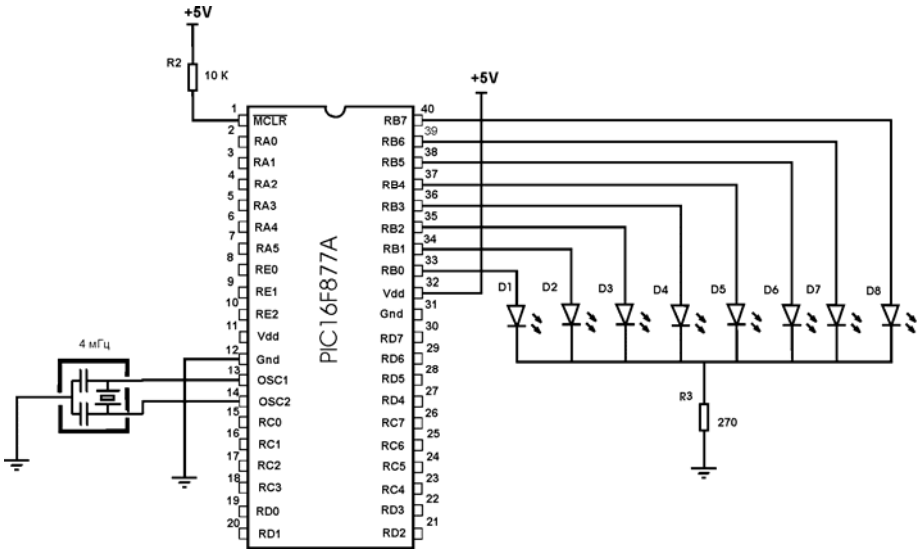


Рис. 19. Схема поочередного включения светодиодов.

А теперь рассмотрим следующую программу:

```
Define  LOADER_USED 1      ' Эта строка в программе необходима если Вы
‘ используете  интерактивную среду разработки (ICD)
LEDS    var     PORTB      ' Введем псевдоним для порта B – LEDS, под
‘ которым мы будем подразумевать все множество светодиодов подключенных
‘ к этому порту
TRISB = %00000000          ' Устанавливаем все выводы PORTB в состояние
выхода
LEDS = 0                   ‘ Определяем начальное состояние выводов PORTB

Main:                      ‘ Начало основной программы
LEDS = LEDS + 1            ‘ Увеличиваем значение переменной на 1
pause 500                  ‘ Задержка на 0,5 секунды
goto main                  ‘ Вернуться в начало программы
```

В первой строке программы мы поместили указание компилятору на то, что мы собираемся работать в интерактивной среде разработчика или ICD. Если Вы не нуждаетесь в ней или у Вас нет возможности работать в этой среде эту строку программы можно убрать.

Далее мы вводим определение переменной `LEDS`, которая будет относиться к байтовому типу. Под этой переменной мы будем подразумевать все множество светодиодов подключенных к PORTB. Далее мы записываем все нули в регистр TRISB. Это приведет к тому, что все выводы PORTB будут установлены в состояние выхода. В пятой строке программы мы присваиваем начальное значение переменной `LEDS`. Понятно, что в двоичной форме оно будет выглядеть как

```
LEDS = %00000000
```

Это означает, что на всех выводах PORTB в начале работы программы будет логический 0. Далее начинается собственно сама программа. Об этом нам говорит метка `Main:`. В самом начале основной программы мы увеличиваем значение переменной `LEDS` на единицу. Это значит, что на выводе 0 PORTB появится логическая единица и первый светодиод зажжется. Для того чтобы заметить это следующим стоит оператор паузы, который позволит программе остановится на время 0.5 секунды. И в конце программы мы снова сталкиваемся с оператором **`goto`** возвращающим нашу программу в начало. Вернувшись в начало, программа снова увеличивает значение переменной `LEDS` на единицу и тогда загорается второй светодиод, а первый гаснет. Таким образом, программа, работая

в цикле, постоянно увеличивает значение указанной переменной на единицу. В результате этого светодиоды будут отображать значение переменной LEDS в двоичном формате. Это будет происходить до тех пор, пока значение переменной не станет равным 255 (%11111111). Затем все повторится сначала.

Теперь рассмотрим другую программу, которая будет работать с этой же схемой.

```
I var Byte                    ' Вводим переменную цикла - I
Symbol LEDS = PORTB   ' Введем псевдоним для PORTB, под которым мы
‘ будем подразумевать все множество светодиодов подключенных к этому
‘ порту
Pause 500
TRISB = %00000000          ‘ Устанавливаем все выводы порта B в
‘ состояние выхода

Loop:                        ‘ Метка начала цикла
LEDS = 1                  ' Устанавливаем все выводы PORTB в низкое логическое
‘ состояние кроме вывода 0.
Pause 500
For I = 0 To 6              ‘ Цикл в котором происходит управление каждым
‘ светодиодом
 LEDS = LEDS << 1       ' При каждом проходе цикла происходит смещение
‘ горящего светодиода
     Pause 500             ' Вводим задержку на 0,5 секунд
Next                        ' Цикл продолжается пока все 8 светодиодов не мигнут
GoTo Loop                   ' Вернуться на метку Loop
```

В первой строке стоит определение вводимой в программу переменной I. Размер этой переменой определяется в байт. Эта переменная позже нам понадобится для организации цикла. Вторая строка нам уже знакома по предыдущему примеру. В следующей строке программы мы вводим задержку, для того чтобы дать возможность микроконтроллеру стабилизироваться после включения питания. Далее, как и в предыдущем примере, мы непосредственно записываем в регистр TRISB все 0, это должно привести к тому, что все выводы PORTB будут находиться в состоянии выхода. Далее мы вводим метку – Loop, которая позволит нам постоянно возвращаться к этой строке в программе. Затем мы присваиваем конкретное значения псевдониму LED = 1. Это же самое мы

могли бы записать в двоичной системе LED = %00000001, что в данном случае, наверное, было бы наглядней. Из этой записи мы видим, что на все выводы этого порта мы подали логический 0 (все эти светодиоды не горят), но на нулевой вывод логическую единицу (светодиод подключенный к этому выводу должен загореться). Далее мы опять вводим оператор задержки длительностью в 0,5 секунды. Это сделано для того, чтобы мы успели увидеть, как этот первый светодиод горит.

В следующей строчке начинается циклический оператор **FOR…NEXT**. Этот оператор работает таким образом: все, что стоит между строками **FOR** и **NEXT** выполняется циклически последовательно до тех пор, пока значение переменной, меняясь от начального значения (I = 0), не станет равным ее конечному значению (7). Причем шаг изменения определяется параметром – **step**. Если же этот параметр опущен, то по умолчанию этот шаг равен 1.

Далее, первым внутри цикла **FOR…NEXT** стоит так называемый оператор сдвига (<<), с помощью которого при каждом циклическом проходе будет происходить смещение единицы в регистре PORTB (%00000001) на один разряд влево.

I = 0 LED = %00000010

I = 1 LED = %00000100

I = 2 LED = %00001000

I = 3 LED = %00010000

I = 4 LED = %00100000

I = 5 LED = %01000000

I = 6 LED = %10000000

В результате работы оператора цикла и оператора разрядного сдвига будет происходить поочередное включение светодиодов справа налево по схеме. После оператора сдвига стоит оператор паузы, который позволяет нам видеть весь процесс с «бегущим огнем». И так будет продолжаться до тех пор, пока значение переменной I не станет равным 6. При этом блок команд, находящийся между словами **FOR** и **NEXT**, выполнится в последний раз, и наша программа выйдет из этого цикла. Выйдя из этого цикла, программа сталкивается с оператором **GOTO**, который возвращает программу к метке Loop. Затем весь процесс с переключением светодиодов повторяется.

В приведенной программе Вы можете изменить временные установки, чтобы сделать передвижение света быстрее или медленнее, изменяя значения в команде **PAUSE**. Поменяв оператор сдвига влево на оператор сдвига вправо (**>>**) и сделав некоторое изменение в значении, присваиваемом псевдониму LEDS в начале цикла Loop, Вы сможете добиться того чтобы «огонь бежал» слева на право.

Сделав некоторые дополнения, как в схеме, так и в программе, Вы можете сделать на основе этого примера новогоднюю гирлянду.

Пример № 3. Взаимодействие с кнопкой.

В предыдущих примерах мы учились, как заставить микроконтроллер выводить информацию. В этом примере мы заставим вывод PORTA.0 микроконтроллера работать как цифровой вход и считывать состояние кнопки, подключенной к этому выводу. Добавим пару элементов в предыдущую схему и получим новую – рис. 20.

Пусть наша программа будет работать следующим образом. Если вывод PORTA.0 находится в высоком логическом состоянии (кнопка не замкнута), ни один из светодиодов не горит. Если кнопку нажать, то на выводе RA0 установится низкое логическое состояние, и тогда мы зажжем один из светодиодов. Пусть это будет D1. При следующем нажатии зажжется D2 и т.д. Задача, кажется, достаточно простой? На рисунке 35. представлена принципиальная схема для этого примера.

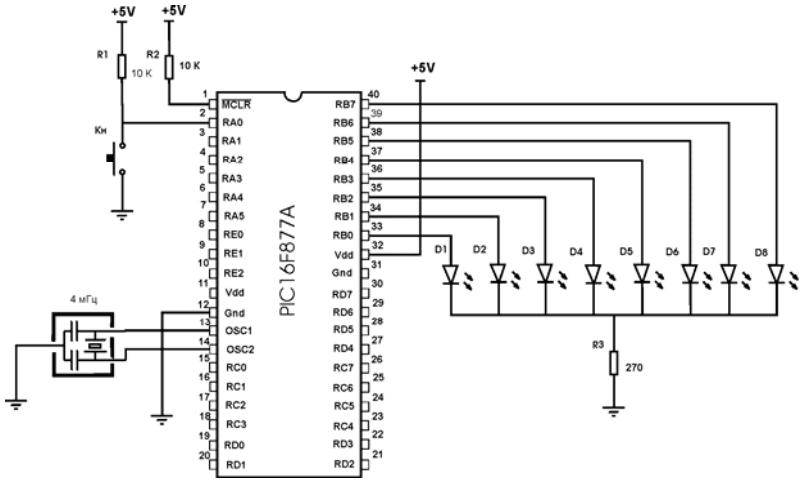


Рис. 20. Взаимодействия микроконтроллера с кнопкой.

А теперь текст самой программы:

```
‘ Вначале идут основные установки и определение переменных
    ADCON1 = %00000110     ' Установим все выводы PORTA как цифровые.
    TRISA = %00000001     ' Установим выводы RA4-RA1  как выходы,  а
‘ вывод RA0 как вход.
    TRISB = %00000000      ‘ Все выводы PORTB на выход
    PORTB = %00000000      ' Записываем все нули в PORTB ;

‘ Здесь начинается основная программа
Main:

' *** Проверяем состояние кнопки ***
IF PORTA.0 = 0 THEN LED    ' Если кнопка нажата то  перейти к
 ‘ подпрограмме LED,
```

```
GOTO Main                    ' Иначе вернуться в начало.

LED:
'*** Подпрограмма реакции на нажатие кнопки. ***
  PORTB = PORTB + 1          ' Ведем счет нажатий переключая
‘ светодиодов.
  GOTO Main                  ' Вернуться к началу цикла main.
```

Приведенная программа начинается, как обычно, с начальных установок и определений. Ввиду того, что выводы PORTA могут работать в трех разных режимах (цифровой вход/выход и режим АЦП), мы записываем в регистр ADON1 значение 6. Таким образом мы устанавливаем все выводы порта А как цифровые. Затем мы определяем направление работы выводов этого порта и PORTB и переходим к основной программе. В первой же строчке программы мы сталкиваемся с условным оператором **IF..THEN**. Здесь это единственный, еще не знакомый Вам, оператор. Согласно определению работы этого оператора он проверяет условие, стоящее после слова **IF** (PORTA.0 = 0), и если оно истинно, происходит переход программы на метку, стоящую после слова **THEN**. Если же условие ложно, то программа продолжает выполняться со следующей строки. Или говоря более подробно программа, дойдя до строки **IF … THEN**, анализирует состояние вывода PORTA.0. Если оно равняется 0, т.е. кнопка нажата, программа переходит на метку указанную после слова **THEN**. Если же состояние вывода равняется 1, то программа переходит на следующую строку. Дальше Вам все уже известно и, надеюсь, понятно.

После того как программа будет откомпилирована и записана в микроконтроллер, мы можем запустить ее и посмотреть, как она работает.

После запуска программы все происходит, так как и должно происходить, но это только до первого нажатия на кнопку. Как только мы нажмем на кнопку, увидим, что зажигается не первый светодиод, а любой из восьми светодиодов и даже, скорее всего какая либо их комбинация. В чем причина такого поведения? Причин здесь две. Первая и самая очевидная это время нажатия на кнопку. Как бы вы ни старались быстро нажать на кнопку, это время будет от нескольких десятков до сотен миллисекунд. А один цикл программы, включающий в себя опрос кнопки и переключение светодиода, займет приблизительно 10 микросекунд при указанной частоте кварца. Отсюда ясно, что пока кнопка находится в состоянии «нажато» программа успевает множество раз опросить порт, к которому она подключена, и зафиксировать это как множество последовательных нажатий. Следующей причиной этого является так называемый дребезг контактов. Более подробно о дребезге контактов написано в разделе «Команда BUTTON». Для того чтобы избавится от обоих этих явлений введите после команды переключения светодиодов (`PORTB = PORTB + 1`) задержку на 200 – 500 микросекунд. Для решения этой задачи можно также использовать специально предназначенную для этого команду **BUTTON**.

Пример:

```
    ADCON1 = 6
    TRISA = %00000001
    PORTA = %00000001
    TRISB = %00000000
    PORTB = %00000000
    B0 var  byte        ‘ Введем служебную переменную для команды
‘ Button
    b0 = 0              ‘ Эту переменную вначале надо приравнять к 0

Main:
Button PORTA.0,0,255,255,B0,1,LED       ' Если кнопка нажата то
‘ перейти на метку LED
GOTO Main                               ' Если кнопка не нажата идти в начало
‘ программы

LED:
PORTB = PORTB + 1
GOTO Main
```

Работу оператора **BUTTON** мы здесь обсуждать не будем. Об этом достаточно подробно разъяснено в соответствующем разделе.

Продолжим работу с портом А, но уже будем использовать его в качестве аналого-цифрового преобразователя.

Пример № 4. Аналого-цифровое преобразование.

В этом примере мы рассмотрим одну из самых полезных особенностей микроконтроллеров, в состав которых входит аналого-цифровой преобразователь (АЦП).

Почти все в реальном мире не цифровое, а аналоговое. Поэтому для того чтобы управлять чем-либо с помощью микроконтроллера, мы должны преобразовывать аналоговые данные реального мира в цифровую форму, которую понимает PIC контроллер. Это можно сделать с помощью аналого-цифрового преобразователя. Например, если Вы должны измерить температуру Вам будет нужен датчик, для того, чтобы преобразовать измеряемую температуру в напряжение, и АЦП, чтобы преобразовать полученное напряжение в цифровое значение.

В этом примере вместо такого датчика мы воспользуемся обычным переменным резистором, названным потенциометром (POT), который подключим к выводу PORTA.0. Поворачивая ротор потенциометра, мы будем изменять

уровень напряжения на этом выводе микроконтроллера, который в свою очередь должен преобразовать значение напряжения на выводе ротора потенциометра в цифровую форму, а результат этого преобразования вывести на светодиоды, подключенные к PORTB. При повороте ротора светодиоды будут зажигаться, точно так же, как работает линейная шкала в стерео усилителях.

К счастью, мы сейчас не обязаны слишком много понимать в работе схемы АЦП; микроконтроллер и компилятор сами знают, как это преобразование осуществляется. Схемы АЦП многообразны, но они все делают одно и тоже - преобразуют аналоговое напряжение в цифровой код. Результат такого преобразования записывается во внутренний регистр микроконтроллера. На цифровом выходе регистра АЦП будет результат этого преобразования. Он может быть 8-разрядным или 10-разрядным. PIC 16F877 имеет 10-разрядный АЦП, но может также работать и как 8-разрядный. Для упрощения программы мы будем использовать здесь 8-разрядное преобразование. Восемь разрядов вписываются в один байт, что несколько упростит нашу программу.

На рис. 21 показана принципиальная схема этого проекта. Эта схема мало чем отличается от предыдущей. Единственное отличие состоит в том, что вместо кнопки к выводу микроконтроллера подключен переменный резистор.

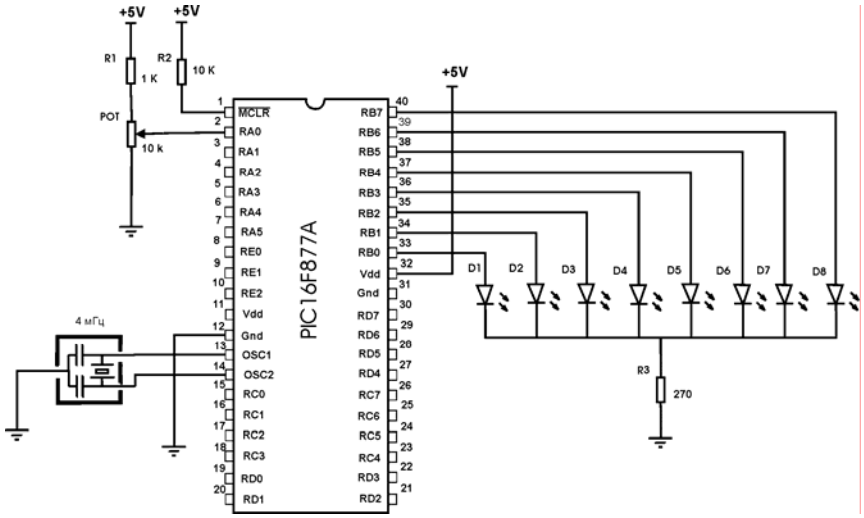


Рис. 21. Схема с использованием АЦП.

Итак, вот текст программы:

```
Define ADC_BITS 8              ' Определяем количество разрядов в
‘ результате АЦП.
Define ADC_CLOCK 3             ' Устанавливаем источник синхронизации (3
‘ = RC).
Define ADC_SAMPLEUS 50   ' Устанавливаем время преобразования.
adval var byte                 ' Создаем переменную adval для хранения
‘ результата преобразования
```

```
' ---[ Инициализация]------------------------
Init:
PortB = $00              ' Устанавливаем все выводы PORTB в 0 (все
‘ светодиоды не горят).
TrisB = %00000000        ' Устанавливаем все выводы PORTB в состояние
‘ выход.
TRISA = %11111111        ' Все выводы PORTA на вход.
ADCON1 = %00000010       ' Устанавливаем все выводы PORTA как аналоговые.
' ---[ Основная программа ]-------------------------
main:
ADCIN 0, adval           ' Читаем канал  A0, а результат преобразования
‘  сохраняем в переменную adval.
'---------- Управление светодиодами ----------------------------------------------------


 if adval < 75 then          ' Если результат преобразования меньше 75,

portb = %00000000            ' то не горит ни один из светодиодов.
endif

if adval > 100 then                ' Если результат преобразования больше
100,
portb = %00000001            ' то светится D1.
endif

if adval > 120 then                ' Если результат преобразования больше
120,
portb = %00000011            ' то светится D2.
endif

if adval > 140 then       ' Если результат преобразования больше 140,
portb = %00000111         ' то светится D3.
endif
```

```
if adval > 160 then     'Если результат преобразования больше 160,
portb = %00001111       'то светится D4.
endif

if adval > 180 then     'Если результат преобразования больше 180,
portb = %00011111       'то светится D5.
endif

if adval > 200 then     'Если результат преобразования больше 200,
portb = %00111111       'то светится D6.
endif

if adval > 210 then     'Если результат преобразования больше 210,
portb = %01111111       'то светится D7.
endif

if adval > 220 then     'Если результат преобразования больше 220,
portb = %11111111       'то светится D8.
endif

Pause 100               'пауза в 100 мс.
goto main               'Идти в начало программы.
```

В этой программе Вы встретились с незнакомым оператором **ADCIN** `0, adval`, который работает следующим образом. Программа при встрече с этим оператором начинает производить измерение аналогового сигнала на канале A0 и совершать АЦП с точностью, указанной в соответствующем определении **DEFINE**. Результат преобразования заносится в переменную (в нашем случае – `adval`).

Вы можете использовать этот пример всякий раз, когда хотите связать цифровую схему с реальным аналоговым миром. Чтение датчиков, вероятно, самое обычное дело, но не единственное. Программа для этого проекта может легко быть превращена в подпрограмму для более сложной программы. Вы можете даже изменить её так, чтобы читать больше чем один датчик, связанный с каждым контактом PORTA.

Я надеюсь, что этот пример не вызовет у Вас больших сложностей в понимании АЦП микроконтроллера.

Кроме светодиодов существует большое разнообразие других электронных изделий, управление которыми часто бывает необходимо. Одним из таких устройств, очень широко используемых в робототехнике, является сервомотор. Поэтому в следующем примере мы будем учится управлять сервомотором.

Пример № 5. Управление сервомотором.

Если Вы когда-либо строили радиоуправляемые модели самолетов или роботов, то вероятно, знакомы с сервомоторами. Сервомотор это двигатель постоянного тока с управлением направлением вращения и угла поворота. Угол поворота вала ограничен 180 градусами, но некоторые люди переделывает внутренности для того, чтобы заставить сервомотор поворачивать на полный угол - 360 градусов. Напряжение питания в большинстве случаев составляет +5 В.. Никаких сложных подключений сервоприводы не требуют. Чаще всего используются всего три внешних провода: напряжение питания, общий провод и входной сигнал управления. Серводвигатель является аналоговым устройством, управляемым с помощью широтно-импульсного модулированного сигнала (PWM). Сигнал должен быть длительностью от одной до двух миллисекунд и повторяться каждые 20 мс. Ширина импульса в 1 миллисекунду поворачивает вал на максимальный угол влево, а шириной в 2 миллисекунды - на максимальный угол вправо. Любая ширина импульса между этими значениями поворачивает вал между конечными точками в линейном масштабе. Импульс длительностью в 1,5 миллисекунды повернул бы вал в промежуточную точку лежащую на полпути к концевым точкам.

Принципиальная схема управления серводвигателем приведена на рисунке 22

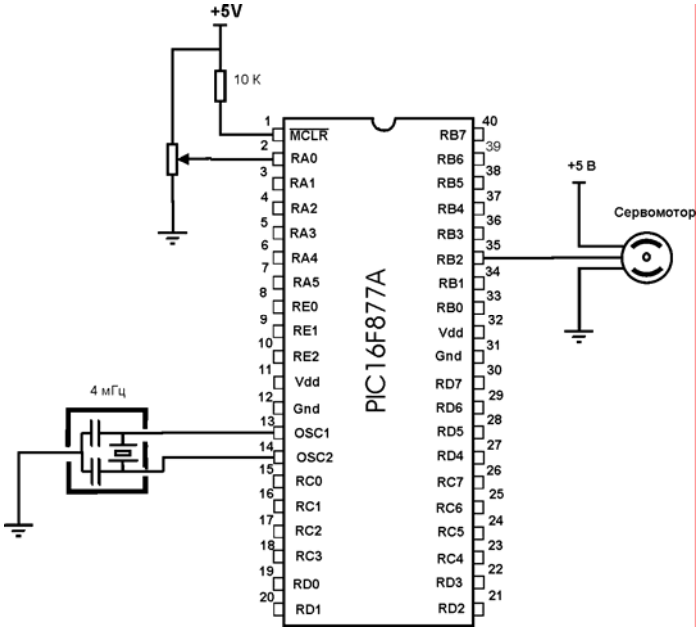


Рис. 22 Схема подключения сервомотора.

В нашем примере мы с вами будем управлять вращением сервомотора с помощью потенциометра из предыдущего примера. Как Вы, наверное, заметили, на схеме не обнаруживаются цепи питания микроконтроллера. Это совершенно не значит, что перед Вами очередной проект Perpetuum Mobile. Это сделано, для того чтобы не отвлекать Ваше внимание на вещи, само собой разумеющиеся.

Ниже приведена программа управления серводвигателем. Программа работает следующим образом: если ротор потенциометра находится в среднем положении, то и сервомотор поворачивается в среднее положение. Если ротор повернуть в сторону уменьшения сопротивления, то сервомотор должен повернуться влево (против часовой стрелки) на угол обратно пропорциональный сопротивлению потенциометра. При повороте ротора от средней точки в сторону увеличения сопротивления сервомотор должен повернуться вправо на угол пропорциональный сопротивлению потенциометра.

Пример:

```
Define ADC_BITS 8
Define ADC_CLOCK 3
Define ADC_SAMPLEUS 50

'---[ Определение переменных]----------------------------
B2 var word          'Определим переменную формирующую длительность
‘ импульса поворота.
adval var byte               'Определим переменную в которую заносится
‘ значение АЦП

'---[ Установки]-------------------------
TrisB = %00000000          'Все выводы PORTB на выход.
TRISA = %11111111          'Все выводы PORTA на вход.
ADCON1 = %00000010         'Устанавливаем выводы PORTA как
‘ аналоговые.

'---[ Основная программа ]----------------------
Main:
PORTB. 2 = 0
ADCIN porta.0, adval       'Читаем канал A0 и результат заносим в
‘ переменную adval
b2 = adval*10/256           'Рассчитываем значение B2 в зависимости от
‘ значения adval.
```

```
PULSOUT PORTb.2, 100 + b2 'Формируем импульс поворота сервомотора.
pause 20
goto main          'Возврат в начало программы
```

В этом примере появился незнакомый Вам оператор **PULSOUT** `PORTb.2,100 + b2`. Назначение этого оператора выдавать импульс указанной ширины на определенном выводе. В нашем случае этот вывод - `PORTb.2`, а длительность 100 + b2 мс. Импульс этот создается двойным изменением логического состояния на указанном контакте. Поэтому важно первоначальное состояние вывода. Поэтому в самом начале основной программы мы устанавливаем вывод PORTB. 2 в 0. Затем мы считываем значение потенциала на выводе PORTA.0 и оцифровав его сохраняем в переменной adval. Для управления сервомотором, мы должны формировать импульсы, длительность которых должна меняться от 1 до 2 мс, поэтому мы должны поставить в соответствие изменение потенциала от 0 В до 5 В изменению длительности импульса в 1 мс. Переменную adval мы определили размером в 8 бит (byte) поэтому в нашем случае она может меняться от 0 до 256. Отсюда один бит этой переменной будет соответствовать adval/256 мВ. В связи с тем, что PicBasicPro не поддерживает переменных с плавающей точкой, мы избавимся от дробной части получающегося числа. Для этого мы вначале умножим значение переменной adval на 10 и только потом его разделим на 256. Получившееся значение переменной b2 будет представлять собой изменяющуюся часть длительности импульса. Длительность импульса определяется минимально возможной длительностью для данной частоты генератора (в нашем случае при частоте 4 МГц это 10 мкс). После того как импульс будет сгенерирован, сформируем минимально необходимую паузу между импульсами в 20 мс. Затем замкнем в цикл процесс выполнения программы.

Замечание. Необходимо учитывать, что сервомотор потребляет значительно больше энергии, чем микроконтроллер. Поэтому если Вы используете общий источник питания, то он должен обладать достаточным запасом мощности. Если же у Вас разные источники, то не забудьте, что цепи “земли” обоих источников должны быть электрически соединены.

Вы можете легко изменить цикл, чтобы повернуть сервомотор в зависимости от входного сигнала, которым может быть выключатель.

Теперь, я думаю, нам пора обратить внимание на другой тип индикатора - это жидкокристаллический индикатор или ЖКИ (LCD).

Пример № 6. Управление ЖКИ.

ЖКИ или ЖК монитор часто используются для вывода информации в удобной для восприятия человеком форме. На рисунке 23 представлен один из таких индикаторов.

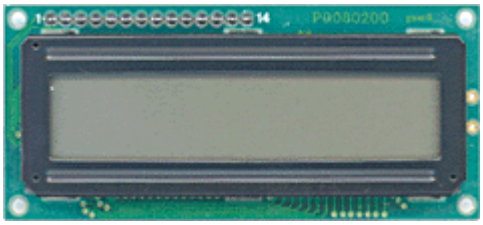


Рис. 23. Пример Жидкокристаллического индикатора.

Существует большое количество различных типов ЖКИ, но в основном (в 99%) распространены индикаторы с интерфейсом Hitachi 44780. Что же представляют собой эти индикаторы? Большинство таких индикаторов имеет 14 выводов. Назначение этих выводов приведено в таблице 27.

Таблица 30

ВЫВОД	ОБОЗНАЧЕНИЕ	НАЗНАЧЕНИЕ
1	Vdd	Общий
2	Vcc	+ Питание
3	Contrast (Vee)	Регулировка контрастности
4	R/S	_Команда/Выбор регистра
5	R/W	Чтение/_Запись
6	E	Тактовые импульсы
7-14	Data	Данные: D0 -7,…,D7 - 14

Запись информации происходит в параллельном коде по фронту тактовых импульсов E. Можно не только записывать данные в регистры ЖКИ, но и читать их. Для этого используется цепь R/W. Для подключения шины данных можно использовать 2 режима: Это режим 8-разрядной шины, когда данные в контроллер ЖКИ передаются байтами, или режим 4-разрядной шины, когда вначале передаются старшие 4 разряда байта, а затем младшие 4 разряда. Хотя понятно, что при 4-разрядной шине количество соединений меньше и загруженность портов меньше, но при этом и скорость обмена меньше. Цепь R/S предназначена для указания типа информации, которая в данный момент подается на выводы данных. Если на линии R/S установлена логическая 1, то на шине данных можно устанавливать ASCII код, символ которого будет отображаться в текущей позиции курсора. Если же на линии R/S будет установлен логический 0, а в цепи R/W логическая 1, то можно считать код символа, отображаемого в данный момент на экране. Хотя в большинстве конструкций вывод R/W подключается к общей шине питания, так как нет необходимости использовать режим чтения. На выполнение команд ЖКИ затрачивается разное время. Например, для стирания экрана или на перемещение курсора в начальную позицию требуется около 4,1 мс, а на другие команды достаточно 160 мкс.

Запись данных производится так же, как и запись команд, только в первом случае по цепи R/S должна быть подана 1. Но использование компилятора

PicBasicPro избавляет Вас от необходимости изучения интерфейса с такими ЖКИ. Он все сделает сам. Для регулировки контрастности изображения Вы можете воспользоваться схемой приведенной на рисунке 24.

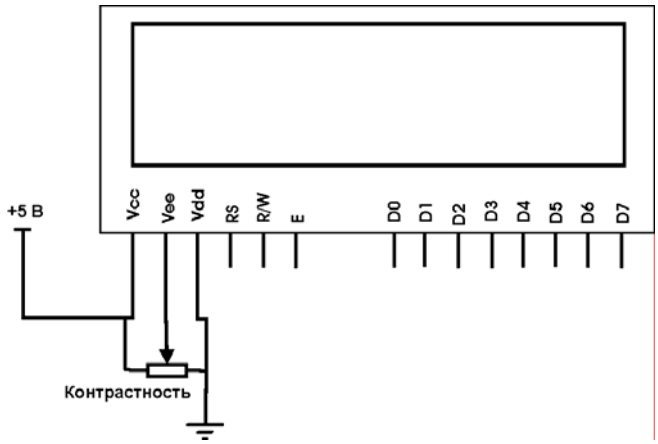


Рис. 24. Схема подключения регулятора контрастности к ЖКИ.

Следующий пример хотя и является простым, но на его основе может быть построено большое число различных устройств. В этом примере, мы будем считывать напряжение с движка потенциометра, осуществлять аналого-цифровое преобразование этого напряжения и выводить результат этого преобразования на экран ЖКИ. При этом мы будем конвертировать значение на выходе АЦП в десятичный формат. Причем значение выводимое на экран будет соответствовать измеряемому напряжению. В нашем примере мы будем использовать ЖКИ имеющий две строки по 16 знакомест в каждой (2 x16). Схема для этого примера приведена на рисунке 25.

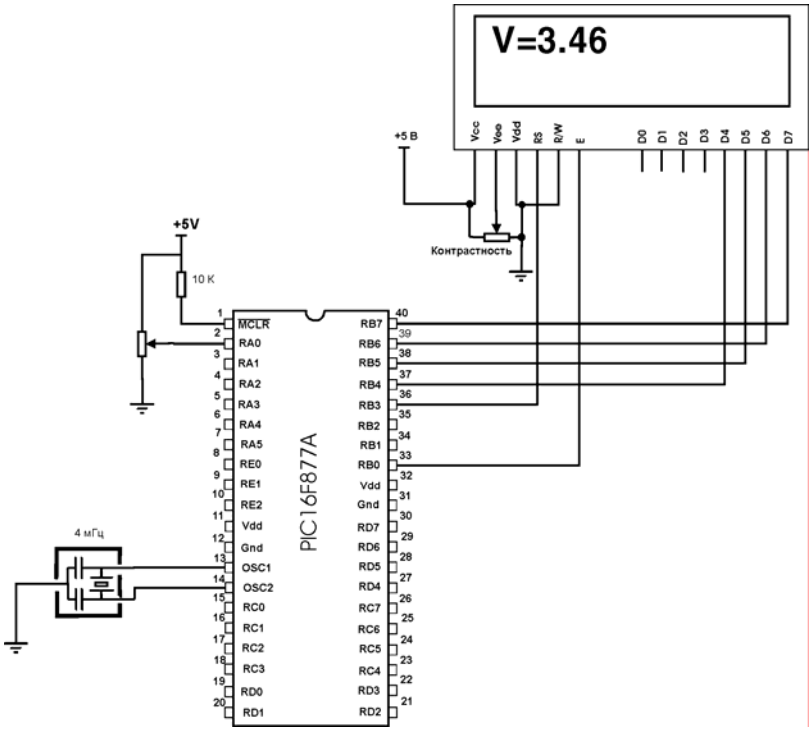


Рис. 25. Схема подключения ЖКИ к микроконтроллеру.

Программа:

```
' ——[ Определения ]————————————————————————————-
DEFINE LCD_DREG PORTB       ' Определяем порт, к которому подключены
‘ цепи данных.
DEFINE LCD_DBIT 4             ' Определяем первый вывод, к которому
‘ подключена шина данных,
DEFINE LCD_RSREG PORTB      ' Определяем порт, к которому подключена
‘ цепь RS.
DEFINE LCD_RSBIT 3          ' Определяем вывод, к которому подключена
‘  цепь RS.
DEFINE LCD_EREG PORTB       ' Определяем порт, к которому подключена
‘  цепь E.
DEFINE LCD_EBIT 0             ' Определяем вывод, к которому подключена
‘  цепь E.
DEFINE LCD_BITS 4             ' Определяем режим 4 – разрядной шины.
DEFINE LCD_LINES 2          ' Определяем тип ЖКИ.
DEFINE LCD_COMMANDUS 2000   ' Определяем время задержки между
‘ командами на ЖКИ.
DEFINE LCD_DATAUS 50          ' Определяем время задержки между
‘ посылками данных.

' Определяем параметры АЦП

DEFINE ADC_BITS 8     ' Определяем разрядность преобразования
DEFINE ADC_CLOCK 3    ' Определяем источник синхронизации для АЦП
DEFINE ADC_SAMPLEUS 50 ' Определяем время преобразования

' Назначение переменных

Res Var Word        ' Определяем переменную, в которую записывается
‘ результат АЦП
Volts1 Var Word     ' Первые два десятичных разряда результата в мВ
Volts2 Var Word     ' Вторые два десятичных разряда результата в мВ

' Константы
```

```
Conv1 Con 19        ' 5000/256 = 19.53, this is the decimal part
Conv2 Con 53        ' Это дробная часть числа


' Инициализация
TRISA = 1           ' Устанавливаем вывод RA0 (AN0) на вход
TRISB = 0           ' Весь PORTB на выход
PAUSE 500           ' Ждем 0.5 сек инициализации ЖКИ
ADCON1 = 0          ' Устанавливаем выводы AN0 - AN4 как аналоговые входа
OPTION_REG = %00000000     ‘ Подключаем подтягивающие резисторы
ADCON0 = %11000001    ' Устанавливаем внутренний источник
‘ синхронизации АЦП
LCDOUT $FE, 1           ' Очищаем экран ЖКИ

Main:

ADCIN 0, Res              ' Считываем аналоговое значение и сохраняем в
‘ переменной Res
Volts1 = Res * Conv1      ' Умножаем это значение на 19
Volts2 = Res * Conv2      ' Умножаем это значение на 53
Volts2 = Volts2 / 100
Volts1 = Volts1 + Volts2    ' Получаем результат в милливольтах
LCDOUT $FE,2,"V = ",DEC4 Volts1   ' Выведем результат на дисплей
PAUSE 500                   ' Ждем 0,5 секунд
GOTO Main
END
```

В первой части программы мы определяем порты и выводы микроконтроллера, к которым подключается ЖК-индикатор. Здесь мы также определяем тип и параметры работы индикатора. Затем идет блок определений, касающийся работы АЦП. Далее следуют блоки определений переменных и констант. Следующим блоком идет блок инициализации работы портов АЦП и портов ЖКИ. И последний блок это собственно сама программа. В первой строке программы находится уже знакомая нам команда. По этой команде происходит считывание и аналого-цифровое преобразование значения напряжения на движке потенциометра подключенного к PORTA.0. Затем мы производим преобразование полученного восьмиразрядного значения Res в шестнадцатиразрядное десятичное

значение. Так как максимальное значение напряжения, которое в нашем случае может быть подано на вход АЦП равно 5000 мВ, а максимальное значение переменной Res может быть 255, следовательно, цена одного бита информации в этом случае равна 5000/256 = 19,53 мВ. К сожалению, в компиляторе PicBasicPro отсутствуют переменные с плавающей точкой, поэтому мы вынуждены для перевода значений полученных в результате АЦП в милливольты, должны будем коэффициент перевода (19,53) разделить на две части это целая часть и дробная часть. Далее произведем отдельно конвертирование целой части и дробной части. Затем полученные результаты сложим. В следующей строке мы сталкиваемся с новым для нас оператором: **LCDOUT** `$FE,2,"V = ",DEC4 Volts1`. По этой команде микроконтроллер выводит, начиная с первой строки и первого знакоместа строку текста - «V = ». Затем после этого текста выводится десятичное значение переменной Volts, которое должно состоять из 4 разрядов. Далее в программе следуют уже известные нам команды.

Кроме жидкокристаллических индикаторов еще широко используются так называемые семи сегментные светодиодные индикаторы. В следующем примере будет рассмотрена работа микроконтроллера с одним из таких индикаторов.

Пример № 7. Подключение 7 сегментного светодиодного индикатора.

Для отображения цифр и некоторых букв часто используется 7-сегментный светодиодный индикатор. В таких индикаторах катоды или аноды всех светодиодов соединены друг с другом и имеют один общий вывод. В этом случае уменьшается количество внешних выводов. Поэтому различают светодиодные индикаторы с общим катодом или с общим анодом.

В примере, который мы будем разбирать ниже, используется такой 7-сегментный светодиодный индикатор с общим катодом. Индикаторы, подобные этому, используются в цифровых часах или в старых калькуляторах.

Управление 7-сегментным индикатором, в принципе, то же самое, что и управление семью отдельными светодиодами. Каждый сегмент 7-сегментного индикатора является отдельным светодиодом, но они имеют либо общий катод, либо общий анод, который выведен на один контакт. На рисунке 26 приведена схема соединений светодиодов и обозначение сегментов такого индикатора.

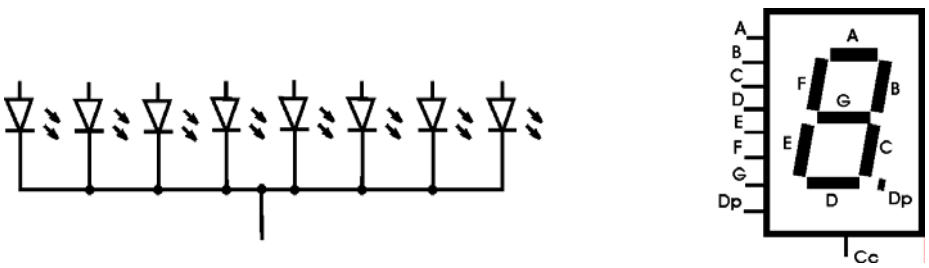


Рис. 26. Схема внутренних соединений 7-сегментного светодиодного индикатора.

Мы будем использовать PORTB микроконтроллера PIC 16F877A для того, чтобы управлять каждым сегментом индивидуально. Наша программа будет считать от 0 до 9, и затем зажжет светодиод, который является децимальной точкой в таком индикаторе. Затем программа произведет обратный счет и также в конце зажжет децимальную точку. В этой программе демонстрируется использование новой для нас команды – **Lookup**, с помощью которой происходит преобразование десятичной цифры в код вывода этой цифры индикатором. На Рис. 27. приведена схема для этого примера.

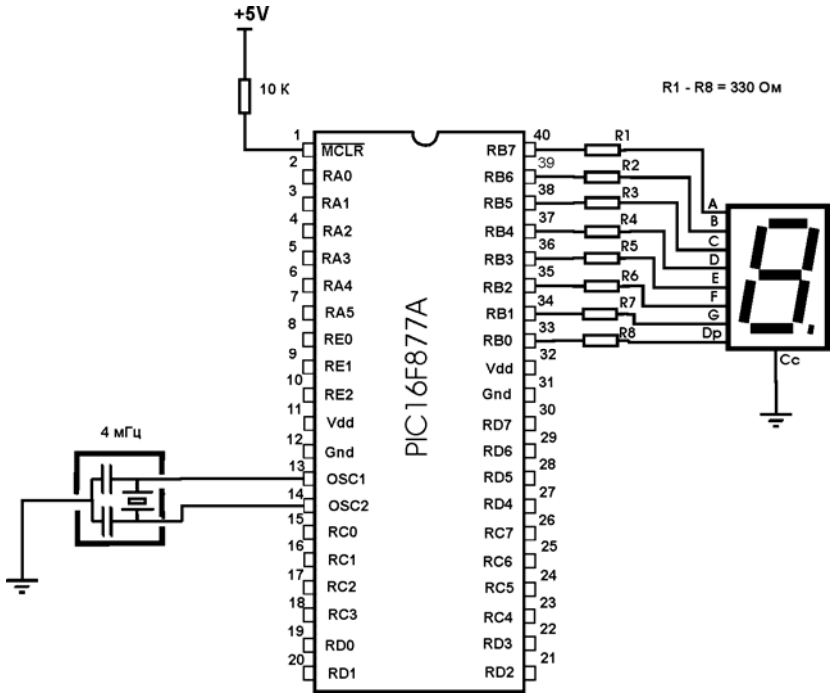


Рис. 27. Схема примера с 7-сегментным индикатором.

Программа:

```
x var byte              ' Введем переменную счета.
numb1 var byte          ' переменная, в которой хранится код вывода
                        ' на 7-с сегментный индикатор.

' Инициализация
TRISB = %00000000       ' Установить все выводы PORTB на выход.
PORTB = %00000000       ' Установить все выводы PORTB в низкое
' состояние.

' Основная программа
Main:
FOR x = 0 TO 9          ' Оператор цикла, который ведет счет от 0 до 9
GOSUB convrt        ' Переход на подпрограмму конвертации.
```

```
portb = numb1
PAUSE 1000                  ' Пауза длительностью  в 1 сек
NEXT

HIGH 0 PORTB.0              ' Счет достиг максимума - 9, поэтому зажигаем
‘ дециальную точку.
pause 1000
LOW 0 PORTB.0               ' Выключаем дециальную точку.

FOR x = 9 TO 0 STEP -1      ' Обратный счет от 9 до 0.
GOSUB convrt
PORTB = numb1       ' Установить соответствующий вывод PORTB  для
                    отображения цифры ‘на  7- сегментном индикаторе.
PASE 1000                   ' Пауза в 1 сек.
NEXT

HIGH 0 PORTB.0              ' Счет достиг 0, поэтому зажигаем дециальную
‘  точку.
PAUSE 1000                  ' Пауза длиной в 1 сек.
LOW 0 PORTB.0               ' Выключаем дециальную точку.
GOTO loop                   ' Возврат к началу.

' Подпрограмма преобразования десятичного числа в 7-сегментный код.
convrt:
lookup x[$7E,$0C,$B6,$9E,$CC,$DA,$F8,$0E,$FE,$CE],numb1
Return
```

Начало этого примера, я думаю, никаких затруднений у Вас не вызовет. Здесь все уже Вам знакомо. Мы определяем переменные, затем делаем начальные установки и далее переходим к основной программе. Программа начинается с того, что с помощью оператора цикла FOR... TO...NEXT ведется счет от 0 до 9. Но когда мы производим счет, мы постоянно обращаемся к оператору **GOSUB**, который отсылает нас к подпрограмме конвертации десятичных цифр в код 7-сегментного индикатора. Преобразование это совершается с помощью оператора **LOOKUP**. Этот оператор позволяет нам выбирать значения из таблицы - [$7E,$0C,$B6,$9E,$CC,$DA,$F8,$0E,$FE,$CE] в зависимости от индекса (x) и присваивать эти значения переменной - numb1. Так, например, если значение x = 0, то выбирается нулевое значение, т.е. $7E, а если x = 5, выбирается четвертое значение - $CC. Коды цифр в таблице даны в шестнадцатеричной системе - об этом указывает префикс - $. Хотя их можно было бы записать и в двоичной, и десятичной системах, но тогда бы на это ушло больше места в тексте программы. На рисунке 41 показано обозначение сегментов такого индикатора. Нулевое значение в таблице равно $7E или в двоичной системе %1111110. Здесь мы видим, что на все сегменты, кроме последнего – G, подается логическая единица, а это значит, что будет светиться цифра – 0.

Первое значение в таблице равно $0C или %00001100. Отбросив первый разряд, получим – 0000110, а это значит, что будут светиться только сегменты B и C, т.е. получится цифра – 1. Надеюсь теперь, Вам понятно, как происходит преобразование десятичной цифры в код 7-сегментного индикатора.

После декодирования оператор **RETURN** возвращает нас туда, откуда мы пришли, но на строчку ниже. И вот здесь происходит вывод кода на индикатор. Затем следует знакомая нам задержка в 1 секунду, и цикл продолжается, пока значение переменной x не достигнет значения 9. Далее оператор **HIGH 0** включает светодиод, который является децимальной точкой (Dp). Он светится в течение 1 секунды, затем выключается оператором **LOW** 0, и далее программа переходит к следующему циклу в котором производится обратный счет по тому же алгоритму и в конце также зажигается децимальная точка. После этого программа замыкается на бесконечный цикл переходя к начальной метке.

А теперь мы поговорим о работе с флэш-памятью. В PIC микроконтроллерах имеется внутренняя флэш-память или EEPROM (электрически стираемая и перепрограммируемая память), но она обычно меньше, чем 1 кбайт. Если Вы нуждаетесь в большей, чем 1 кбайт памяти, то Вам необходимо использовать микросхемы внешней флэш-памяти. Доступ к внутренней EEPROM в PicBasicPro весьма легок - он осуществляется он с помощью двух команд **WRITE** или **READ**. Доступ к внешней памяти сложнее. Многие внешние микросхемы памяти работают с использованием протокола разработанного фирмой Phillips - I^2C. PicBasicPro делает это легко с помощью команд **I2CIN** и **I2COUT**. В последующих примерах будет показано, как получить доступ к внутренней памяти EEPROM.

Пример № 8. Работа с внутренней ЕЕПРОМ.

Многие типы микроконтроллеров обладают собственной встроенной флэш-памятью. В частности, рассматриваемый нами контроллер PIC16F877A имеет встроенную флэш-память на 256 кб. Так вот давайте попробуем записывать в нее и затем считывать. Схемой для этого Вы можете воспользоваться из примера в котором к микроконтроллеру подключен ЖКИ. Да и программный текст, в своей основе, будет из того же примера.

Пример программы:

```
'——[ Определения ]————————————————————————————-
DEFINE LCD_DREG PORTB        'Определяем порт, к которому подключены
‘ цепи данных.
DEFINE LCD_DBIT 4             'Определяем первый вывод, к которому
‘ подключена шина данных,
DEFINE LCD_RSREG PORTB       'Определяем порт, к которому подключена
‘ цепь RS.
DEFINE LCD_RSBIT 3           'Определяем вывод, к которому подключена
цепь RS.
```

```
DEFINE LCD_EREG PORTB      ' Определяем порт, к которому подключена
‘ цепь E.
DEFINE LCD_EBIT 0          ' Определяем вывод, к которому подключена
‘ цепь E.
DEFINE LCD_BITS 4          ' Определяем режим 4 – разрядной шины.
DEFINE LCD_LINES 2         ' Определяем тип ЖКИ.
DEFINE LCD_COMMANDUS 2000  ' Определяем время задержки между
‘ командами на ЖКИ.
DEFINE LCD_DATAUS 50       ' Определяем время задержки между
‘ посылками данных.

'——-[ Переменные ]————————————————————————-
num var byte       ‘ Определяем переменную счета.
mem var byte       ‘ Вводим переменную памяти.

'——-[ Основная программа ]————————————————————
——-
Main:
FOR num = 0 TO 10          ‘ Цикл последовательной записи в память
ЕЕПРОМ
WRITE num, num             ‘ чисел от 0 до 10.
NEXT
' ***** Вывод на экран” ******
 LCDOUT $fe, 1             ' Очистим экран LCD.
 LCDOUT $fe, 2             ' Установим курсор в начальную позицию.

' Цикл последовательного вывода на экран значений считанных из внутренней
ЕЕПРОМ, начиная с 10 ячейки
FOR num =10 TO 0 STEP -1
 READ num, mem
 LCDOUT dec mem
 PAUSE 500                 ' Пауза в 0.5 сек.
 NEXT
 PAUSE 1000
GOTO Main
```

В этом примере как и в предыдущем основная программа начинается с цикла в котором постоянно производится запись во внутреннюю флэш-память. Здесь мы сталкиваемся с новым для нас оператором **WRITE** num, num , который производит запись значения переменой - num в соответствующую ячейку памяти (num). В результате работы этого цикла будет произведена запись в последовательные ячейки памяти от 0 до 10 значения, совпадающие с номером ячейки. Далее мы подготавливаем экран монитора для последующего вывода на него информации, которая будет считана из внутренней флэш-памяти. Затем

следует второй цикл, в котором производится считывание информации из внутренней флэш-памяти и вывод ее на экран ЖКИ. Считывание информации производится с помощью оператора - **READ** num, mem. Этот оператор считывает значение из ячейки под номером num, и считанное значение сохраняет в переменной mem. После окончания цикла осуществляется задержка в программе длительностью в одну секунду. Это нужно для того чтобы успеть увидеть все что было сделано. Затем все повторяется с начала.

Теперь перейдем к работе микроконтроллера с внешней памятью.

Пример № 9. Работа с внешней ЕЕПРОМ.

В дополнение к микроконтроллерам PIC, фирмы Microchip, Atmel и многие другие выпускают микросхемы внешней памяти. Для обмена информацией с этими микросхемами обычно используется протокол I^2C, а в PicBasicPro есть команды **I2CIN** и **I2COUT**, которые как раз и предназначены для работы по этому протоколу.

Хотелось бы коротко напомнить о шине I^2C. Название шины I^2C представляет собой аббревиатуру от сочетания Inter Integrated Circuit (IIC, или I^2C). Эта стандартная последовательная шина была изначально предложена фирмой Philips, но сейчас микросхемы работающие по этому протоколу выпускаются многими производителями. Шина I^2C использует только два сигнальных провода и обеспечивает обмен данными в последовательной форме со скоростью до 400 Кбит/с.

Эта шина имеет следующие основные характеристики:

- Это последовательная двухпроводная шина с цепями: SDA (Serial Data) – цепь передачи данных и SCL (Serial Clock) – цепь передачи тактовых синхроимпульсов;
- Данные могут передаваться в обоих направлениях;
- Каждый абонент шины имеет свой персональный семиразрядный служебный адрес. Следовательно, к шине может быть подключено до 128 абонентов. Каждый из абонентов может инициировать обмен и управлять им;
- Пересылка каждого байта данных должна завершиться сигналом подтверждения приема;
- Скорость шины может автоматически замедляться в зависимости от возможностей абонента;
- Максимальное число подключаемых абонентов ограничивается емкостной нагрузкой шины. Она не должна превышать 400 пФ;
- Электрические уровни шины допускают использование микросхем, изготовленных по технологиям КМОП, NMOП, ТТЛ.

В этом примере мы будем использовать микросхему 24LC00, которая имеет очень маленькую емкость памяти. Она имеет всего 16 байт. Можно было бы спросить, почему мы используем такую микросхему, когда внутренняя память PIC, больше чем у нее. Причина этого только в том, чтобы показать, как простая микросхема с EEPROM может быть добавлена к любому PIC микроконтроллеру, особенно если у этого контроллера нет вообще никакого внутреннего EEPROM. В некоторых случаях 16 байт EEPROM могут быть очень удобными за очень небольшую стоимость. На рисунке 28 показана схема этого примера. Как мы можем видеть, схема взята из предыдущих двух примеров, но добавлена микросхема памяти. А за основу программного текста был взят текст программы из предыдущего примера.

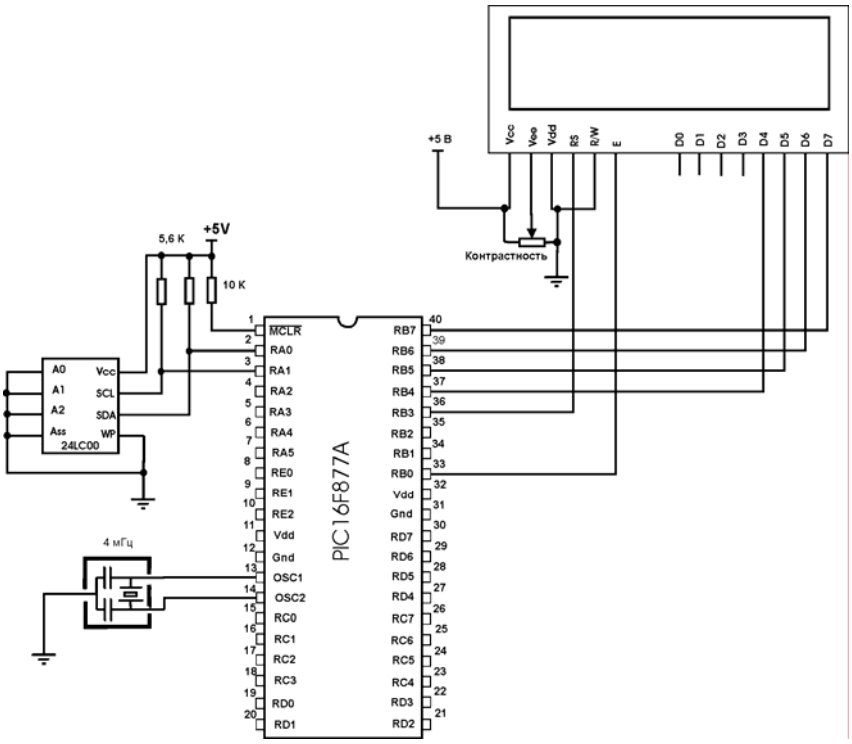


Рис. 28. Схема подключения внешней ЕЕПРОМ.

Программа:

```
' ——[ Определения ]————————————————————————————-
DEFINE LCD_DREG PORTB              ' Определяем порт,  к которому
‘ подключается шина данных ЖКИ.
DEFINE LCD_DBIT 4                  ' Определяем первый контакт PORTB, к
‘  которому подключается  шина данных ЖКИ - DB4.
DEFINE LCD_RSREG PORTB             ' Определяем порт, к которому
‘ подключается цепь RS.
DEFINE LCD_RSBIT 3                 ' Определяем контакт PORTB, к
‘ которому подключается  цепь RS.
DEFINE LCD_EREG PORTB        ' Определяем порт, к которому подключается
‘ цепь E.
```

```
DEFINE LCD_EBIT 0                ' Определяем контакт PORTB, к
' которому подключается цепь E.
DEFINE LCD_BITS 4                ' Определяем режим связи с ЖКИ (4-
' разрядная шина).
DEFINE LCD_LINES 2               ' Определяем тип ЖКИ.
DEFINE LCD_COMMANDUS 2000        ' Определяем время задержки между
' командами.
DEFINE LCD_DATAUS 50         ' Определяем время задержки между
' посылками данных.

'——-[ Переменные ]————————————————————————-
adr var byte             ' Вводим переменную адреса.
dan var byte                 ' Вводим переменную вывода на экран.
Control con %10100000            ' Вводим константу - контрольный
' байт для связи с EEPROM.
DPIN var PORTA.0                 ' Вводим переменную - цепь вывода
' данных шины I²C.
CPIN var PORTA.1                 ' Вводим переменную - цепь вывода
' синхроимпульсов шины I²C.
Address var byte                 ' Вводим переменную адреса.

'——-[ Инициализация ]————————————————————————
adcon1 = 7                   ' Устанавливаем режим PORTA  как
'цыфровой.
TRISA = %00000000            ' Устанавливаем все выводы PORTA на выход.
porta = %00000000

'——-[ Основная программа ]————————————————————————
———-
Main:
 FOR adr = 0 TO 10           ' Цикл в котором значения адреса меняются
' от 0 до 10
 I2CWRITE dpin,cpin,control,adr,[adr] ' и по этим же адресам
' записываем в память
 PAUSE 20                    ' Пауза для повышения надежности записи
 NEXT

'___ Вывод на экран ЖКИ ____
 LCDOUT $fe, 1
 LCDOUT $fe, 2
 FOR adr =10 TO 0 STEP-1      ' Цикл в котором значения адреса
' меняются от 10 до 0
 I2CREAD dpin,cpin,control,adr,[dan]  ' Считываем по этим адресам
```

```
‘ из памяти значения и сохраняем их в  переменной dan.
 LCDOUT dec dan                    ' Выводим на экран ЖКИ значения
‘  переменной dan.
 NEXT
 PAUSE 1000                        ‘ Пауза чтобы заметить изменения
 GOTO Main                         ' Вернутся в начало программы.
```

Этот пример очень похож на предыдущий пример. Разница заключается в том, что здесь для записи в память используется команда **I2CWRITE** dpin,cpin,control,adr,[adr]. По этой команде микроконтроллер записывает по адресу adr значение переменной указанной в квадратных скобках во внешнюю память. При этом для связи с внешней памятью используются выводы микроконтроллера dpin и cpin . После того как будет произведена запись последовательных значений от 0 до 10 в соответствующие ячейки памяти, программа переходит к циклу, в котором производится последовательное считывание и вывод на экран считанных значений. Считывание информации в этом случае производится с помощью команды
I2CREAD dpin,cpin,control,adr,[dan]. Понятно, что адрес считываемых данных определяется переменной adr, а считанные данные сохраняются в переменной dan. После включения схемы мы видим, как последовательно на экране появляются числа – 10,9,8…0.

В следующем примере мы с Вами разберем работу микроконтроллера со стандартной 12 кнопочной (ее еще называют телефонной) клавиатурой.

Пример № 10. Работа с 12 кнопочной клавиатурой.

Для тех случаев, когда микроконтроллер используется в панелях управления, или еще где-либо, часто возникает необходимость в оперативном вводе либо коррекции некоторой информации. Именно для этих случаев использование клавиатуры бывает очень удобно. Вариантов клавиатур существует огромное количество. Если клавиатура состоит из нескольких клавиш, то они могут быть подключены к микроконтроллеру как отдельные кнопки, то есть каждая через свой порт. Но если кнопок много то сам собой напрашивается вопрос о применении некоторого кодировщика, для того, чтобы сократить количество цепей подключения к микроконтроллеру. Для этого можно воспользоваться различными микросхемами-шифраторами, а можно и использовать так называемую матричную клавиатуру. Этот прием в настоящее время является самым распространенным. Что же собой представляет матричная клавиатура? Это клавиатура, у которой клавиши находятся на пересечении строк и столбцов матрицы. На рисунке 29 показана схема построения такой клавиатуры.

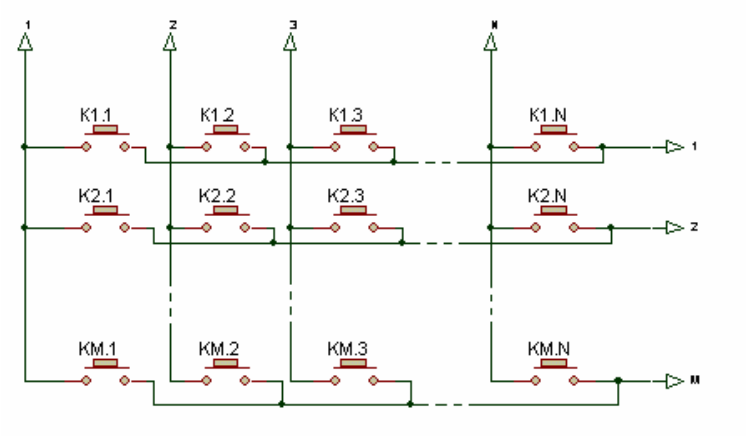


Рис. 29. Схема построения матричной клавиатуры.

При нажатии на одну из кнопок произойдет замыкание (соединение) определенного столбца с определенной строкой. Если строки и столбцы такой клавиатуры подключить к портам микроконтроллера и опрашивать эти порты в определенном порядке, то всегда можно определить, когда и какая кнопка была нажата. Например, если по порядку подавать на каждую цепь строки высокий логический уровень и при этом, опрашивая поочередно цепи столбцов, выяснить, где этот высокий уровень обнаружен, можно определить, какая кнопка была нажата. На рисунке 30 показана схема подключения к микроконтроллеру 12 кнопочной клавиатуры и ЖК монитора для вывода информации о нажатой клавише.

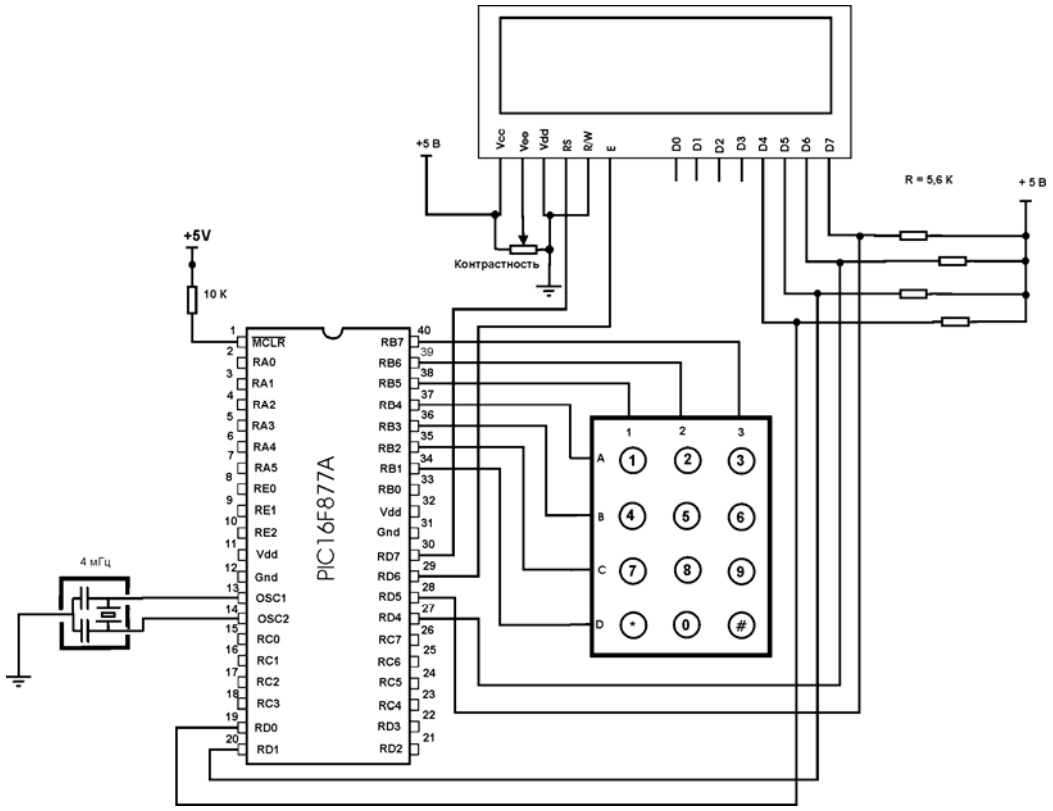


Рис. 30. Схема подключения 12-кнопочной клавиатуры.

Пример:

```
DEFINE LCD_DREG PORTD       'Определяем порт, к которому подключены
‘цепи данных.
DEFINE LCD_DBIT 0           ' Определяем первый контакт порта, к
‘которому подключены цепи данных на LCD DB4.
DEFINE LCD_RSREG PORTD      'Определяем порт, к которому подключена
‘цепь RS .
DEFINE LCD_RSBIT 7          ' Определяем контакт, к которому
‘подключается цепь RS.
DEFINE LCD_EREG PORTD       ' Определяем порт, к которому подключена
‘цепь E.
DEFINE LCD_EBIT 6           ' Определяем контакт, к которому
‘подключена цепь E.
DEFINE LCD_BITS 4           ' Определяем режим 4-разрядной шины.
DEFINE LCD_LINES 2          ' Определяем тип ЖК монитора.
DEFINE LCD_COMMANDUS 1000   'Определяем задержку между посылками
‘команд  на ЖК монитор.
DEFINE LCD_DATAUS 50        ' Определяем время задержки между
‘посланными данными.
'———-[ Переменные ]————————————————————————-
    Flag     Var Bit  ‘ Вводим переменную.
    Key      Var Byte
    Debounce     Var Bit   'Флаг показывающий нажатие клавиши.
    D_Flag   Var Bit   'Флаг Debounce flag used by Inkeys.
'———-[ Начальная установка ]————————————————————————
TRISB=%00000111
PortB=%11111111
flag=1
    LCDOUT $fe, 1        'Очистим экран ЖК монитора.
    LCDOUT $fe, 2        'Установим курсор в начальную позицию.

'———-[ Основная программа ]————————————————————————
———-

Main:
    GOSUB KeyScan
    IF FLAG =0 THEN LCDOUT $FE,1:LCDOUT $FE,2

'***** Вывод на экран код клавиши  ******
IF Flag=1 THEN LCDOUT key  'Если клавиша нажата, вывести на экран
‘ ЖК значение наж. клавиши.
    PAUSE 100                'Пауза в 100 мсек для осуществления
‘автоповтора.
    GOTO Main                'Вернуться назад в начало программы,
```

```
‘ чтобы начать цикл снова.

Keyscan:
' Эта подпрограмма сканирует 12-кнопочную клавиатуру и возвращает значение
‘  нажатой  кнопки в
‘ переменную «KEY».  Если никакая кнопка не нажималась, то возвращается
‘ значение 128.
‘Она также возвращает переменную по имени " Debounce ", которая равна  1,
‘ если клавиша  все еще  нажата,  и  0, если не нажата. Это является
‘ противодребезговой функцией для этой клавиатуры.

     Debounce=1          ' Установка начального значения для Debounce.
     Key=0               ' Очищаем переменную KEY перед просмотром.
     TrisB=%11100000     ' Устанавливаем 4 вывода PortB на выход.
                         ' и 3 вывода PortB на вход.
     Option_Reg.7=0      ' Подключаем внутренние подтягивающие
‘ резисторы PortB.
     PortB=%01111101     ' Подать на строку D клавиатуры 0.
     GOSUB Col           ' Перейти на подпрограмму сканирования колонок.
     IF Flag=1 THEN GOTO Map   ' Если клавиша нажата, то перейти на
‘ метку Map.
     PortB=%11111011     ' Подать на строку C клавиатуры 0.
     GOSUB Col           ' Перейти на подпрограмму сканирования колонок.
     IF Flag=1 THEN GOTO Map   ' Если клавиша нажата, то перейти на
‘ метку Map.
     PortB=%11110111     ' Подать на строку B клавиатуры 0.
     GOSUB Col           ' Перейти на подпрограмму сканирования колонок.
     IF Flag=1 THEN GOTO Map   ' Если клавиша нажата, то перейти на
‘ метку Map.
     PortB=%11101111     ' Подать на строку A клавиатуры 0.
     GOSUB Col           ' Перейти на подпрограмму сканирования колонок.
     IF Flag=1 THEN GOTO Map   ' Если клавиша нажата, то перейти на
метку Map.
     D_Flag=0       ' В противном случае (клавиша не нажата), сбросить
‘ флаг – debounce.
     Debounce=0
     GOTO Exit           ' Выйти из подпрограммы.
Map:
IF D_Flag=1 THEN Exit      ' Если уже было нажта клавиша то выход.
       D_Flag=1          ' Установить флаг Debounce.
      Debounce=0         ' Обнулить Debounce.
Exit:
'Lookup Key,[1,2,3,4,5,6,7,8,9,10,0,11,128],Key
```

```
 ' таблица перевода значений ‘переменной Key в числовые значения.

' **Если необходимо  преобразовать код нажатой клавиши в ascii код, то
‘ закомментируйте строку программы расположенную  выше. Если этого
‘ делать не надо, то закомментируйте строку программы
‘ расположенную  ниже
LOOKUP

' Таблица ‘перевода значений переменной Key в значения  ASCII кода.
KEY,["*","0","#","7","8","9","4","5","6","1","2","3",32],Key

'IF FLAG=1 THEN LCDOUT key       ' Отправить “Hello World” на ЖКИ
'pause 500
      RETURN              ' Выход из подпрограммы.

' Это подпрограмма сканирования колонок клавиатуры. Если какая либо клавиша
' нажата, переменная Flag устанавливается в 1, и в 0, если никакая клавиша не
‘ нажата.
' Кроме того, если никакая клавиша не нажата, то переменной - KEY,
‘ возвращается  значение 12.
Col:
Flag=1                ' Устанавливаем переменную Flag в 1.
IF PortB.5=0 THEN S_Exit   ' Если нажата клавиша в первом столбце, то
‘ перейти пп.  S_Exit(,)
Key=Key+1             ' Иначе инкремент переменой KEY и перейти к
‘ следующему столбцу.
IF PortB.6=0 THEN S_Exit   ' Если нажата клавиша во втором столбце,
‘ то перейти на пп. S_Exit(,)
Key=Key+1             ' Иначе инкремент переменой KEY и перейти к
‘ следующему столбцу.
IF PortB.7=0 THEN S_Exit   ' Если нажата клавиша в третьем столбце,
‘ то перейти на пп. S_Exit(,)
Key=Key+1             ' Иначе инкремент переменой KEY (она теперь
‘ равняется 12).
Flag=0           ' Устанавливаем переменную flag в 0.
S_Exit:          ‘ противодребезговая подпрограмма.
PAUSEUS 100
RETURN           ' выйти из подпрограммы.
```

В следующем примере мы с Вами поговорим о выводе звукового сигнала.

Пример № 11. Создание музыки.

Для того чтобы воспроизводить звуки с помощью микроконтроллера вполне достаточно подключить через конденсатор в 1 мкФ к выводу микроконтроллера,

маломощный динамик сопротивлением порядка 8 Ом либо пьезобуззер (пищалка).

Конечно, название этого примера звучит слишком громко. Нет, мы не будем здесь учиться писать симфонии и песни, мы просто сделаем простейший музыкальный инструмент, нажимая на кнопки которого будут воспроизводиться музыкальные ноты.

Для начала соберем схему представленную на рисунке 31, и запишем в микроконтроллер программу:

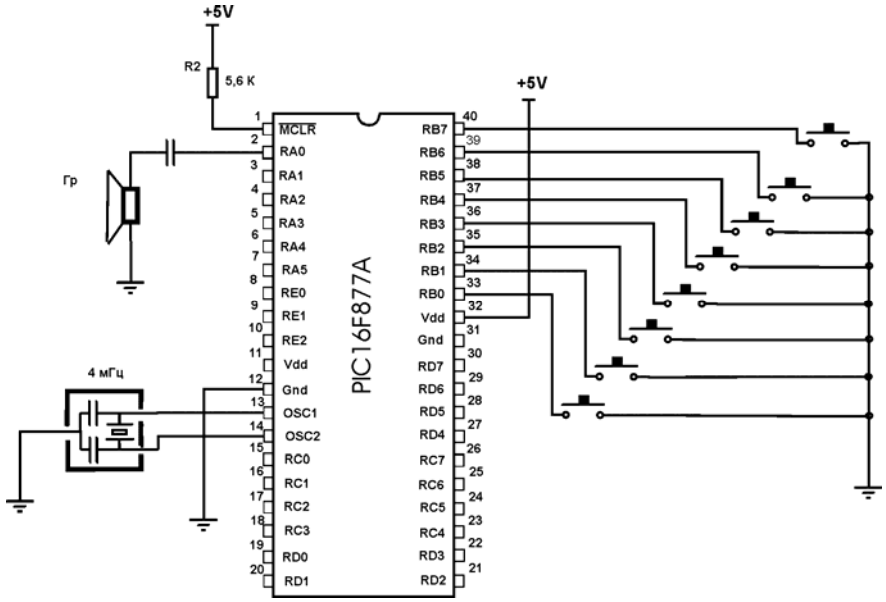


Рис. 31. Схема к примеру «Создание музыки»

```
Speaker VAR PORTA.0    ' Псевдоним для динамика подключенного к PORTA.0
Notes VAR Word[9]              ' Определение массива частот для муз. нот
Key VAR Byte                   'Код нажатой клавиши

' Список массива частот
Notes[1] = 262 : Notes[2] = 294 : Notes[3] = 330 : Notes[4]
= 349
Notes[5] = 392 : Notes[6] = 440 : Notes[7] = 494 : Notes[8]
= 524

' Конфигурируем порты
TRISB = %11111111      ' PORTB весь на вход
```

```
OPTION_REG.7 = 0            'Подключим подтягивающие резисторы в
PORTB
TRISA = 0                   'PORTA на выход
ADCON1 = %111               'Устанавливаем все выводы PORTA как цифровые
'Основная программа
Main:
Key = PORTB         'Считываем PORTB и присваиваем его переменной Key
IF Key <> 255 THEN      'Если нажата любая клавиша
key = ~key                  'Инвертируем все биты переменной Key
Key = NCD Key               'Преобразуем значение переменной Key в код
'вывода порта
FREQOUT Speaker,50,Notes[Key]  'Выдать ноту на динамик
ENDIF
GOTO Main                   'Вернуться в начало
END                     'Конец программы
```

В этой программе мы сталкиваемся сразу с несколькими новыми операторами. Во-первых, во второй и четвертой строках программы мы впервые вводим определение массива переменных. Мы указали, что массив `Notes` состоит из 9 членов. В четвертой строке мы указали все члены массива кроме нулевого члена. Это сделано потому, что нулевой член мы не используем по причине, которая будет разъяснена позже. Далее мы конфигурируем порты в соответствии с нашими задачами. Затем идет текст основной программы. В первой строке основной программы мы считываем регистр `PORTB` (состояние кнопок) и сохраняем его значение в переменной Key. Далее мы сталкиваемся с оператором условия, который говорит, что если не нажата ни одна из клавиш, программа перескакивает к предпоследней строке и затем возвращается в начало. Если же нажата любая из клавиш, т.е. значение переменной Key не равно 255 программа переходит на следующую строку стоящую после оператора **`IF...THEN`**. Здесь происходит побитное инвертирование всех разрядов переменной Key. Например, если была нажата четвертая клавиша то Key = %11110111. После побитного инвертирования эта переменная станет Key = %00001000. Далее идет строка с оператором **`NCD`**, который возвращает номер самого старшего разряда переменной равный 1. Это значит, что теперь переменная Key = 4. И, наконец, следует оператор, который подает на вывод `Speaker` в течение 50 мс частоту определяемую значением члена массива - `Notes[Key]`. Затем программа вновь возвращается в начало основного цикла. Так программа постоянно опрашивает клавиатуру и генерирует звуки (если нажата любая из клавиш).

В этой главе мы рассмотрели 11 примеров программ в которых микроконтроллер выполняет задачи, которые распространены в повседневной

практике. На основании этих примеров и тех примеров которые находятся в папке «SAMPLES» инсталлированного пакета Вы можете потихоньку начинать писать собственные программы. Для того чтобы Ваши программы были написаны грамотно в следующей главе приводятся описания команд языка PicBasicPro для компилятора версии 2.47.

Глава 4. Команды языка PicBasicPro.

Вначале хотелось бы привести полный список команд PicBasic Pro для компилятора версии 2.47:

@ Вставить одну строку на ассемблере.
ADCIN Считать аналоговое значение на выводе и осуществить АЦП.
ASM..ENDASM Вставить подпрограмму на ассемблере.
BRANCH Переход на одну из меток определяемую индексом (еквивалент **ON..GOTO**).
BRANCHL Команда **BRANCH** для многостраничной памяти (длинное **BRANCH**).
BUTTON Производит обработку нажатия кнопки
CALL Вызов подпрограммы ассемблера.
CLEAR Обнуление всех переменных.
CLEARWDT Обнуление Сторожевого таймера.
COUNT Счет импульсов на определенном выводе.
DATA Первоначальная запись констант в память ЕЕПРОМ микроконтроллера
DEBUG Передача информации в стандартном асинхронном формате
DEBUGIN Прием информации в стандартном асинхронном формате.
DISABLE Отключение процесса отладки и обработок прерываний.
DISABLE DEBUG Отключение процесса отладки.
DISABLE INTERRUPT Отключение процесса обработки прерывания.
DTMFOUT Генерирует на определенном выводе сигналы тонального набора номера
EEPROM Запись во внутреннюю ЕЕПРОМ (аналог команды **DATA**).
ENABLE Включить процесс обработки прерываний и работу отладчика.
ENABLE DEBUG Включить процесс отладки.
ENABLE INTERRUPT Включить процесс обработки прерываний.
END Остановить выполнение программы и перейти в режим малого энергопотребления
FOR..NEXT Многократно выполнить инструкции (один из операторов цикла)
FREQOUT Генерирует одну или две частоты на определенном выводе
GOSUB Вызвать подпрограмму на БЕЙСИКЕ по указанной метке
GOTO Продолжить выполнение программы с указанной метки.

HIGH Установить вывод в высокое логическое состояние.
HPWM Генерирует на выводе ШИМ сигнал.
HSERIN Аппаратный асинхронный последовательный ввод.
HSEROUT Аппаратный асинхронный последовательный вывод.
I2CREAD Чтение данных с использованием интерфейса I2C.
I2CWRITE Запись во внешнее устройство с использованием интерфейса I^2C.
IF..THEN..ELSE..ENDIF Выполнить инструкции в зависимости от условия.
INPUT Установить вывод в состояние входа.
LCDIN Чтение значений из ОЗУ ЖК монитора.
LCDOUT Вывод знаков на экран ЖК монитора.
{**LET**} Определение переменной (устаревшее).
LOOKDOWN Поиск значений в таблице констант
LOOKDOWN2 Поиск значений в таблице констант/переменных.
LOOKUP Выбрать постоянное значение из таблицы.
LOOKUP2 Выбрать значение констант/переменных из таблицы.
LOW Установить вывод в низкое логическое состояние и сделать его выходом.
NAP Отключить процессор на короткий промежуток времени.
ON DEBUG Использовать внутрисхемный отладчик.
ON INTERRUPT Выполнить подпрограмму обработки прерываний.
OWIN Принять данные по однопроводному интерфейсу.
OWOUT Передать данные по однопроводному интерфейсу.
OUTPUT Сделать вывод выходом.
PAUSE Включить паузу (на n мс.).
PAUSEUS Включить паузу (на n мкс.).
PEEK Прочитать байт из регистра.
POKE Записать байт в регистр.
POT Считать значение потенциала на выводе.
PULSIN Измерить длительность импульса на выводе.
PULSOUT Сгенерировать импульс на выводе.
PWM Сгенерировать ШИМ сигнал на выводе.
RANDOM Сгенерировать псевдо-случайное число.
RCTIME Измерить продолжительность нахождения вывода в определенном логическом состоянии.
READ Считать значение из внутренней ЕЕПРОМ.
READCODE Считать значение из памяти программ микроконтроллера.
RESUME Вернуться в основную программу после обработки прерывания.
RETURN Продолжить выполнение команд с того места откуда был выход по **GOSUB**.
REVERSE Сделать состояние вывода противоположным.
SELECT CASE Выбор значения переменной из группы различных значений.

SERIN Прием информации по асинхронному каналу связи.
SERIN2 Прием информации по асинхронному каналу связи.
SEROUT Передача информации по асинхронному каналу связи.
SEROUT2 Передача информации по асинхронному каналу связи).
SHIFTIN Прием данных по синхронному каналу связи.
SHIFTOUT Передача данных по синхронному каналу связи.
SLEEP Выключить процессор на некоторое время.
SOUND Генерировать частоту или белый шум на выводе.
STOP Остановить выполнение программы.
SWAP Обменяйте значения двух переменных.
TOGGLE Инвертировать состояние вывода.
USBIN Прием данных по каналу USB.
USBINIT Инициализация USB порта.
USBOUT Передача данных по каналу USB.
WHILE..WEND Выполнить инструкции, пока условие истинно.
WRITE Записать новое значение во внутреннюю ЕЕПРОМ.
WRITECODE Записать новое значение размером в WORD в память программ.
XIN Принять данные по интерфейсу X-10.
XOUT Передать данные по интерфейсу X-10..

А теперь опишем работу каждой из команд более подробно.

Команда @.

@ Выражение на ассемблере

Когда знак - @ используется в начале строки, то это означает, что далее на этой строке следует подпрограмма на ассемблере, которая вставлена в Вашу BASIC программу. Поэтому смело используйте этот знак, когда хотите включить команды ассемблера в программу PicBasicPro.

Пример:

```
i var byte
rollme var byte
FOR i = 1 TO 4
   @ rlf _rollme, F
NEXT i
```

Этот знак может быть также использован для включения подпрограммы ассемблера находящейся в другом файле.
Например:

```
@ Include "fp.asm"
```

Команда - **@** сбрасывает регистр выбора страницы памяти в 0 перед выполнением команд ассемблера. Страница памяти не должна меняться, при использовании команды - **@**.

Команда ADCIN.

Синтаксис
ADCIN *Channel,Var*

По этой команде считывается значение потенциала на входе выбранного канала АЦП, преобразуется в цифровой код, и результат сохраняется в указанной переменной. И хотя доступ к регистрам аналого-цифрового преобразователя можно получить непосредственно, команда **ADCIN** делает этот процесс несколько проще.

Прежде, чем Вы будете использовать команду **ADCIN**, необходимо соответствующий разряд регистра TRIS установить так, чтобы он стал входом. В регистре ADCON1 также необходимо установить соответствующий разряд так чтобы вывод канала АЦП перевести в режим аналогового входа, а в некоторых случаях установить формат результата и источник синхронизации. См. справочные листы данных фирмы Microchip для получения дополнительной информации об этих регистрах и настройке их для определенного устройства. Для этих же целей могут использоваться несколько команд DEFINE.

Пример:

```
DEFINE ADC_BITS 8       'определяем разрядность преобразования
DEFINE ADC_CLOCK 3      'Устанавливаем источник синхронизации (rc = 3)
DEFINE ADC_SAMPLEUS 50      'установим время выборки в микросекундах
TRISA = 255             'установим все разряды PORTA на вход
ADCON1 = 2              'определяем режим PORTA как  аналоговый  вход
ADCIN 0, B0             'читаем канал 0, результат записываем в
' переменную B0
```

Команда ASM...ENDASM.

Синтаксис:

```
ASM
   {assembler code}
ENDASM
```

Команды **ASM** и **ENDASM** сообщают компилятору PicBasicPro, что программа, находящаяся между этими двумя строками это программа на ассемблере и не должна интерпретироваться как команды PicBasicPro. Вы можете использовать

эти два слова, для того чтобы свободно вставлять подпрограмму на ассемблере в программу PicBasicPro.

Максимальный размер текста вставляемой программы на ассемблере не должен превышать 8 кб. Это максимальный размер, включая комментарии, на сгенерированную программу. Если блок текста программы больше чем допустимо, то необходимо разбить текст программы ассемблера на части или просто включить эго в отдельный файл. Команда **ASM** сбрасывает страницу памяти в 0.

Пример:

```
ASM
   bsf PORTA, 0 ; установить разряд 0 PORTA в 1
   bcf PORTB, 0 ; обнулить разряд 0 PORTB
ENDASM
```

Команда BRANCH.

Синтаксис:
BRANCH *Index*,[*Label*{,*Label...*}]

Команда **BRANCH** заставляет программу перейти на одну из указанных меток в программе, на основании переменного индекса. Индекс выбирается один из списка меток. Выполнение программы продолжается от указанной метки.

Например, если индекс - нуль, программа переходит к первой в списке меток, если индекс равен единице, программа переходит ко второй метке, и так далее. Если индекс больше или равен числу меток, то никакое действие не будет предпринято, и выполнение продолжается со следующей строки после команды **BRANCH**. В команде **BRANCH** могут быть до 255 меток (256 для 18Cxxx). Метка должна находиться в той же самой странице памяти, что и команда.

Если Вы не уверены в этом, то используйте команду **BRANCHL**.

Пример:

```
BRANCH B4,[dog,cat,fish]
```

Или то же самое:

```
If B4=0 Then dog          ' перейти на метку dog
If B4=1 Then cat          ' перейти на метку cat
If B4=2 Then fish         ' перейти на метку fish
```

Команда BRANCHL.

Синтаксис:
BRANCHL *Index*,[*Label*{,*Label...*}]

BRANCHL (или BRANCH длинный) работает точно также, как и команда **BRANCH**. То есть она заставляет программу переходить к различным меткам, в зависимости от переменной индекса. Основные различия заключаются в том, что этот оператор может перейти к метке, которая находится на другой странице памяти, чем та, в которой находится команда **BRANCH**. Но при этом компилятор создает код программы, который будет вдвое длиннее, чем код программы, созданный командой **BRANCH**. Если Вы уверены, что метки находятся на той же странице памяти, где находится и команда **BRANCH** или если микроконтроллер не имеет больше, чем одна страница памяти (ПЗУ меньше 2 кб), то используйте команду **BRANCH** вместо **BRANCHL**. Это минимизирует используемую память.

Пример:

```
BRANCHL B4,[dog,cat,fish]
```

Или то же самое:

```
If B4=0 Then dog       'если B4=0 то перейти на метку dog
If B4=1 Then cat 'если B4=1 то перейти на метку cat
If B4=2 Then fish 'если B4=2 то перейти на метку fish
```

Команда BUTTON.

Синтаксис:

```
BUTTON Pin, DownState, Delay, Rate, BVar, Action, Label
```

Работа этой команды, к сожалению, не совсем верно описана в файле помощи поставляемого вместе с компилятором.

По этой команде считывается значение на выводе - Pin. Этот вывод автоматически устанавливается в состояние входа. Параметр *Pin* может быть константой, значения которой лежат в пределах от 0 и до 15, переменной, значения которой могут меняться от 0 и до 15 (например, B0) или названием вывода (например, PORTA.0). Все кнопки можно разделить на два варианта подачи сигнала на вывод *Pin*. В первом случае при замыкании контактов кнопки на вход микроконтроллера подается логический 0. Во втором случае логическая единица. Это отображено на прилагаемом рисунке 45.

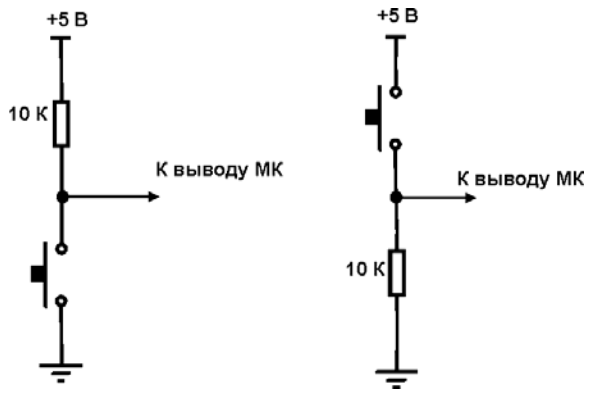


Рис. 32. Варианты подключения кнопок к микроконтроллеру.

Параметр `DownState` указывает на то какое логическое состояние на выводе к которому подсоединена кнопка должно быть в том случае когда кнопка нажата. Это может быть однобитовая переменная или константа. Именно от совпадения значения данного параметра со значением на выводе `Pin` будет определяться то, что клавиша была нажата.

Параметр `Delay` (задержка) может быть как переменная, так и константа, размер которой должен быть определен в один байт. Это значит, что значения `Delay` могут лежать в пределах от 0 до 255. Этот параметр по замыслу разработчика должен выполнять функцию противодребезговой защиты (debounce). Задержка измеряется в циклах обращения к команде **BUTTON**. Задержка имеет два специальных параметра настройки: 0 и 255. Если *Delay* = 0, то команда **BUTTON** не выполняет никакой противодребезговой защиты и автоповтора. Если же *Delay* = 255, команда исполняет противодребезговую защиту, но не осуществляет никакого автоповтора.

Параметр `Rate` также как и параметр `Delay` может быть переменной или константой с размером в один байт. Опять же по замыслу разработчика он должен был бы выполнять функцию автоповтора.

Вообще говоря, когда происходит замыкание некоторых механических контактов то в течение 1 – 100 мс совершаются короткие микроудары контактов друг о друга, то есть происходит серия замыканий - размыканий контактов. Это называется дребезгом контактов. Понятно, что это может интерпретироваться микроконтроллером как более чем одно срабатывание кнопки. Для того чтобы этого не происходило, в команде **BUTTON** присутствует так называемая противодребезговая защита, которая избавляет нас от этого электромеханического «шума». Физически она представляет собой некоторую задержку во времени после первого срабатывания контактов. Команда **BUTTON** также реагирует на то, как долго происходит нажатие на кнопку. Например, работая на компьютере, Вы замечали, что когда нажимаете на клавишу, символ нажатой клавиши немедленно появляется на экране. Если удерживать клавишу некоторое время, то на экране появляется повторяющейся поток этих символов. Это называется автоповтором. В команде **BUTTON** и присутствует такая функция автоповтора.

Параметр *BVar* является байтовой переменной. Эту переменную желательно установить вначале программы в 0.

Параметр *Action* – однобитовая переменная либо константа. Установкой этого параметра в 0 либо в 1 мы можем регулировать то, как будет происходить реакция на нажатие клавиши.

Параметр *Label* - метка, на которую переходит программа, при соблюдении ряда условий.

С помощью параметра *Action* можно устанавливать два варианта работы команды **BUTTON.**

В первом варианте, если параметр *Action* будет установлен в 0, команда **BUTTON** будет функционировать следующим образом. Когда программа доходит до команды **BUTTON,** производится анализ состояния вывода *Pin*. Если значение на выводе не совпадает со значением параметра *DownState* то программа переходит на метку *Label* и продолжается её дальнейшая работа. Если же значения этих двух параметров совпадут (кнопка нажата), то программа переходит на следующую строку, куда обычно записываются действия, которые программа должна выполнить в случае нажатия кнопки. Причем время выполнения команды **BUTTON** в этом случае будет занимать 50 мс. При этом в переменную *BVar* записывается значение *Delay*. Так как данная команда, по замыслу разработчика, должна работать в цикле то при каждом следующем обращении к команде **BUTTON** будет происходить декремент переменной *BVar*. Естественно, что это будет только тогда когда значение на выводе и значение *DownState* совпадают. При этом программа после команды **BUTTON** будет переходить на метку *Label*. Это будет продолжаться до тех пор, пока значение переменной *BVar* не станет равным 1. Если значение переменной *BVar* становится равной 1, то программа после команды **BUTTON** опять переходит на следующую строку на которой записано действие которое она должна выполнять когда кнопка нажата. Учитывая то что время выполнения одной команды **BUTTON** когда происходит декрементирование переменной *BVar* занимает 40 мкс на частоте 4 МГц (8 мкс на частоте 20 МГц), противодребезговая задержка на частоте 4 МГц будет равна 40*(*Delay*-1)[мкс] + 50[мс] (50 мс это то время когда команда **BUTTON** первый раз «столкнулась» с нажатой кнопкой). После этого при следующем обращении к команде **BUTTON** (при условии совпадения значений на выводе контакта *Pin* со значением параметра *DownState*) в переменную *BVar* будет записано значение параметра *Rate*. В следующем цикле при обращении к команде **BUTTON** будет происходить декремент переменной *BVar* (условие равенства значения на выводе *Pin* и значения *DownState* должно по-прежнему соблюдаться). При последующих обращениях к этой команде, пока значение переменной *BVar* будет > 1, программа будет продолжаться с метки *Label*. Когда же значение *BVar* станет вновь равным 1, программа продолжится со следующей строки, а в переменную *BVar* будет вновь записано значение *Rate*. Это и будет выполнением функции автоповтора. Из

сказанного ясно, что период времени между автоповторами будет равен 40*(*Rate* – 1)[мкс] на частоте 4 МГц .

Пример:-

```
'Данная программа демонстрирует работу команды BUTTON. Каждая из 3
‘ кнопок переключает светодиод.
' Если Вы держите кнопку в течение 1 секунды, то светодиод будет моргать
(автоповтор).
Device 16F84A
XTAL = 4

Dim buf1 As Byte                    ' Переменная для 1 кнопки
Dim buf2 As Byte                    ' Переменная для 2 кнопки
Dim buf3 As Byte                    ' Переменная для 3 кнопки
Symbol sw1 = PORTb.0
Symbol sw2 = PORTb.1
Symbol sw3 = PORTb.2
Symbol led1 = PORTb.7
Symbol led2 = PORTb.6
Symbol led3 = PORTb.5
delayms 500                         ' Ждем стабилизации PIC-контроллера
Clear                               ' Очищаем буферы
Low PORTB.5                         ' Выключаем все светодиоды
low PORTB.6
LOW PORTB.7

chk1:
delayus 25

Button sw1,0,40,5,buf1,0,chk2       ' Проверяем кнопку 1 (Переходим
‘ на 2 если не нажато)
Toggle led1                         ' Переключаем 1 светодиод

chk2:
Button sw2,0,40,5,buf2,0,chk3       ' Проверяем кнопку 2 (переходим
‘ на 3 если на нажата)
Toggle led2                         ' Переключаем 2 светодиод

chk3:
Button sw3,0,40,5,buf3,0,chk1       ' Проверяем кнопку 3 (переходим
‘ на 1 если не нажата)
Toggle led3                         ' Переключаем 3 светодиод

GoTo chk1                           ' Возвращаемся на начало и так до
```

‘ бесконечности

Во втором варианте работы команды **BUTTON** приравняем значение параметра *Action* к 1.

В этом случае при первом обращении к команде **BUTTON** когда переменная *BVar* равна 0, а значение на выводе *Pin* не совпадает со значением переменной *DownState* (кнопка не нажата), программа продолжает свою работу со следующей строки и переменная BVar не меняет своего значения. Если же мы нажимаем кнопку и программа «сталкивается» с командой **BUTTON** (значение на выводе *Pin* равно значению *DownState*), то в этом случае программа переходит на указанную метку и в переменную *BVar* записывается значение *Delay*. Если же продолжается удержание кнопки и программа циклически обращается к команде **BUTTON**, то, как и в первом варианте с каждым обращением к команде будет происходить декремент переменной *BVar* но программа будет переходить не на указанную метку, а на следующую строку. Это будет продолжаться так же, как и в варианте 1 до тех пор, пока переменная BVar не станет равной 1. После этого в неё будет записано значение *Rate*, а программа перейдет на метку. Дальнейшая работа команды **BUTTON** будет аналогична первому варианту, но только в тех случаях, где в ранее описанном варианте происходил переход на метку, в этом варианте программа будет работать со следующей строки. Когда же программа в предыдущем варианте переходила на следующую строку, здесь она будет переходить на указанную метку и наоборот, там, где программа переходила на метку, здесь она будет переходить на следующую строку.

Пример для случая, когда параметр *Action* равен 1:

```
'Данная программа демонстрирует работу команды BUTTON. Каждая из 3
‘ кнопок переключает светодиод.
' Если Вы держите кнопку в течение 1 секунды, то светодиод будет моргать
(автоповтор).
Device 16F84A
XTAL = 4

Dim buf1 As Byte                    ' Рабочий буфер для 1 кнопки команды
Dim buf2 As Byte                    ' Рабочий буфер для 2 кнопки команды
Dim buf3 As Byte                    ' Рабочий буфер для 3 кнопки команды
Symbol sw1 = PORTb.0
Symbol sw2 = PORTb.1
Symbol sw3 = PORTb.2
Symbol led1 = PORTb.5
Symbol led2 = PORTb.6
Symbol led3 = PORTb.7
delayms 500                         ' Ждем стабилизации PIC-контроллера
Clear                               ' Очищаем буферы
Low PORTb.5                         ' Выключаем все светодиоды
```

```
low PORTB.6
LOW PORTB.7

Main:
delayms 20
Button sw1,0,4,5,buf1,1,chk1  'Проверяем кнопку 1 (Переходим на
‘ метку 1 если нажато)
Button sw2,0,40,5,buf2,1,chk2 'Проверяем кнопку 2 (переходим
‘ наметку 2 если нажато)
Button sw3,0,40,5,buf3,1,chk3 'Проверяем кнопку 3 (переходим
‘ наметку 1 если нажато)

GoTo Main

chk1:   Toggle led1          'Зажигаем светодиод если нажато

GoTo Main

chk2:   Toggle led2          'Зажигаем светодиод если нажато

GoTo Main

chk3:   Toggle led3          'Зажигаем светодиод если нажато


GoTo Main                    'Возвращаемся на начало и так до
‘ бесконечности
```

В Интернете на странице разработчика компилятора рекомендуется использовать такую конструкцию программы для обслуживания кнопки:

```
MyButton var PortB.0
LED var PortB.7

TRISB = %00000001

Loop:
If MyButton=0 then
Toggle LED
While MyButton=0:Pause 100:Wend
endif
Goto Loop
End
```

Команда CALL.

Синтаксис:
CALL *Label*

По этой команде происходит переход на подпрограмму ассемблера, которая помечена меткой *Label*. Для того чтобы выполнить переход на подпрограмму PicBasicPro, обычно используется оператор **GOSUB**. Основное различие между операторами **GOSUB** и **CALL** состоит в том, что в операторе **CALL**, существование метки *Label* не проверено до времени ассемблирования. Использование команды **CALL** недоступно в PicBasic.

Пример:

```
CALL pass       'Выполните подпрограмму ассемблера, по метке _pass
```

Команда CLEAR.

Этот оператор обнуляет всю оперативную память. По команде **CLEAR** обнуляются все регистры оперативной памяти в каждом банке. При этом обнуляются все переменные, включая и внутренние системные переменные. Это не делается автоматически, когда начинается программа PicBasic Pro. Вообще говоря, переменные в начале программы должны быть установлены программой в соответствующее начальное состояние вместо того, чтобы использовать команду **CLEAR**.

Пример:

```
CLEAR 'установить все переменные в 0
```

Команда **CLEARWDT**.

Обнулить сторожевой таймер.

По этой команде происходит сброс сторожевого таймера, т.е. он устанавливается в 0. Понятно, что это препятствует ему выйти из режима сна.

Пример:

```
CLEARWDT 'сбросить сторожевой таймер
```

Команда **COUNT**.

Синтаксис:
COUNT *Pin,Period,Var*

По этой команде начинается счёт импульсов, которые приходят на вывод – *Pin* за время - *Period*, и результат сохраняется в переменной *Var*. Вывод *Pin* автоматически становится входом. Значение параметра *Pin* может быть константой от 0 до 15, или переменной, которая может содержать значения от 0 до 15 (например, B0) или имя контакта (например, PORTA.0).

Параметр *Period* измеряется в миллисекундах. И он зависит от частоты задающего генератора, которая определяется оператором **DEFINE** OSC.

Оператор **COUNT** проверяет состояние вывода *Pin* и считает переходы из низкого логического состояния в высокое. Для случая, когда частота задающего генератора равна 4 МГц, программа проверяет состояние контакта, каждые 20 мкс. При частоте в 20 МГц проверяет каждые 4 мкс. Отсюда ясно, что самая высокая частота, которая может быть измерена - это 25 кГц при кварце в 4 МГц и 125 кГц с кварцем на 20 МГц. При этом считается, что импульсы имеют форму меандра.

Пример:

```
' Сосчитать количество импульсов на выводе Pin1 за 100 миллисекунд
COUNT PORTB.1, 100, W1

' Измерить частоту на выводе - pin
COUNT PORTA.2, 1000, W1   ' Сосчитать за 1 секунду
SEROUT PORTB.0, N2400, [W1]
```

Команда **DATA**.

Синтаксис:

{label} **DATA** {@*Location*,}*Constant1*{,*Constant2*...}

Эта команда производит запись данных (Constant) в энергонезависимую память EEPROM, при программировании микроконтроллера. Если у микроконтроллера флэш-память, то такая запись производится при каждом акте программирования. Если параметр Location опущен, то первое значение данных записывается по адресу 0 т.д. Если же значение Location определено, то оно и обозначает адрес, по которому эти данные сохраняются. Дополнительная метка может быть назначена как стартовый адрес в EEPROM для того, чтобы позже можно было бы на него сослаться в программе.

В качестве значений Constant могут быть как числовые, так и строковые константы. Если модификатор WORD не используется, то будет сохранен только самый младший байт числового значения. Строковые переменные сохраняются как последовательность байтовых значений в кодировке ASCII. Никакого признака длины или конца автоматически не добавляется.

Оператор **DATA** работает только с теми микроконтроллерами, у которых на кристалле присутствует EEPROM, типа PIC16F84 и PIC16C84. Естественно, что

он не будет работать с микроконтроллерами, у которых нет своей внутренней EEPROM.

Пример 1:

```
DATA @5,10,20,30          'Сохранить 10, 20 и 30 с адреса 5
```

Пример 2:

```
dlabel DATA word $1234 'Записываются двухбайтовые значения: $34, $12
```

Пример 3:

```
DATA (4), 0(10)
```

В этом случае вначале будут пропущены 4 ячейки памяти, а затем подряд в 10 ячейках будет записано значение 0.

Пример 4:

```
DATA "ab"
```

В данном случае вначале, т.е. в 0 ячейке памяти будет сохранен код буквы a – 97. В следующей ячейке будет сохранен код буквы b – 98.

Команда **DEBUG.**

Синтаксис:
DEBUG *Item* {,*Item*...}

По этой команде микроконтроллер передает значения в стандартном асинхронном формате. При этом используется следующий формат передачи: 8 информационных разрядов, отсутствует проверка на четность и 1 стоповый бит (формат - 8N1). Скорость передачи измеряется в бодах (бит/сек).

Передача информации происходит через вывод, определенный в команде Define, который автоматически устанавливается в состояние выхода. Если посылаемому значению предшествует признак (#), то это значит, что посылается ASCII код. Команда **DEBUG** поддерживает те же самые модификаторы данных, что и команда **SEROUT2**.

Таблица модификаторов команды **DEBUG**

Таблица 11.

Модификатор	Операция
{I}{S}BIN{1..16}	Посылает двоичные значения
{I}{S}DEC{1..5}	Посылает десятичные значения

{I}{S}HEX{1..4}	Посылает шестнадцатеричные значения
REP c\n	Посылает знак c, повторив его n раз
STR ArrayVar{\n}	Посылает строковую переменную n знаков

DEBUG это одна из нескольких встроенных асинхронных последовательных функций. Она является наименьшей из сгенерированных последовательных подпрограмм программного обеспечения. Эта команда, в основном, предназначена для отладки программ. С ее помощью можно посылать информацию об изменении переменных, регистров маркеры позиции программы, и т.д. в окно MicroCode Studio (ICD) в режиме отладки. И использование этой команды вне этого режима вряд ли целесообразно. Тем более разработчик компилятора нигде не приводит описания и примеров применения данной команды. Поэтому Вам для связи по асинхронному каналу лучше пользоваться командой **SEROUT**.

Команда **DEBUGIN.**

Синтаксис:
DEBUGIN {*Timeout*,*Label*,}**[***Item*{,*Item*...}**]**

По этой команде осуществляется прием значений по определенному выводу в предопределенной скорости в бодах в стандартном асинхронном формате, с использованием 8 бит данных. При этом отсутствует проверка на четность и 1 стоповый бит (8N1). Вывод автоматически становится входом.

Дополнительно могут быть включены параметры: блокировка времени - *Timeout* и метка - *Label*, чтобы позволить программе продолжаться, если знак не был получен в течение определенного времени, которое измеряется в единицах миллисекунд. Если сигнал не был получен за это время, то программа выйдет из команды **DEBUGIN** и перейдет на указанную метку.

Команда **DEBUGIN** так же как и команда **DEBUG** были введены разработчиком для режима отладки (MicroCode Studio (ICD)). Поэтому использование этих команд вне этого режима, скорее всего, нецелесообразно. Тем более что разработчик нигде не предлагает примеров их использования. Для организации обмена информацией по асинхронному каналу связи лучше воспользуйтесь командами: **SERIN**, **SERIN2**, **SEROUT**, **SEROUT2**.

Команда **DISABLE**.

По команде **DISABLE** прекращается работа программы в режиме отладки а также прекращается обработка всех прерываний. И хотя прерывания всё ещё могут происходить, но программа обработки прерываний не будет на них реагировать, пока не столкнется с командой **ENABLE**. Команды **DISABLE** и **ENABLE** больше походят на псевдокоманды, в которых они только дают

указания компилятору, вместо того, чтобы фактически создавать некую программу.

Пример:

```
DISABLE   ‘ Отключить обработку прерываний
myint: led = 1        ' Включить светодиод когда происходит прерывание
RESUME    ' Вернутся в основную программу
ENABLE    ' Включить обработку прерывания
```

Команда **DISABLE DEBUG.**

По этой команде отключается программа обработки команд отладки. Отладчик не будет вызываться, пока не столкнется с командами **ENABLE** или **ENABLE DEBUG**.

Команда **DISABLE INTERRUPT.**

Команда **DISABLE INTERRUPT** отключает программу обработки прерываний. И, несмотря на то, что процессы вызывающие прерывания могут происходить, микроконтроллер не будет на них реагировать, пока не встретится с командами **ENABLE** или **ENABLE INTERRUPT**.

Команда **DTMFOUT**.

Синтаксис:
DTMFOUT *Pin*,{*Onms*,*Offms*,}[*Tone*{,*Tone*...}]

По этой команде происходит генерация стандартного двухтонального телефонного номера.

Параметр *Pin* указывает на вывод микроконтроллера, который по этой команде переходит в состояние выхода. Значения этого параметра могут быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0).

Параметр *Onms* – время генерации одним частотным набором в миллисекундах. По умолчанию *Onms* – 200 мс

Параметр *Offms* – пауза в миллисекундах между генерациями. По умолчанию *Offms* – 50 мс

Параметр *Tone* представляет собой номер двухчастотной комбинации стандартного набора номера. Наборы частот пронумерованы от 0 и до 15. Частоты с номерами 0 - 9 – соответствуют цифрам на телефонной клавиатуре. Набор 10 соответствует клавише – «*» на телефонной клавиатуре, а набор 11 соответствует

клавише – «#». Номера 12-15, соответствуют клавишам A-D расширенной телефонной клавиатуры.

Команда **DTMFOUT** при своей работе использует команду **FREQOUT**. С той лишь разницей, что на выходе микроконтроллера - Pin генерируется комбинация стандартного телефонного двухчастотного набора номера. Если к выводу *Pin* подключить фильтр, примерная схема которого представлена на рисунке 19, то на выходе этого фильтра получим комбинации гармоник стандартного телефонного набора номера.

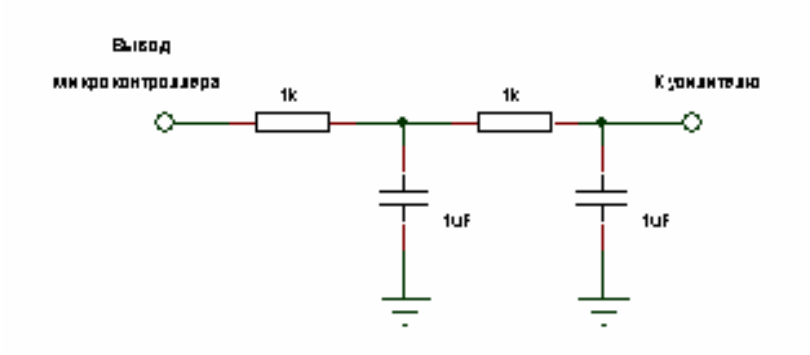


Рис. 33. Одна из схем фильтров по выделению гармонического сигнала.

Команда **DTMFOUT** как и команда **FREQOUT** по умолчанию рассчитана и лучше всего работает с задающим генератором на 20 МГц. Но она может также работать и с генератором на 10 МГц, и с генератором на 4 МГц. Информацию по поводу значения частот формируемых этой командой смотри раздел **FREQOUT**.

Пример:

```
' Сгенерируем двухтональный многочастотный набор 212 на выводе Pin1
DTMFOUT PORTB.1,[2,1,2]
```

Команда **EEPROM.**

Синтаксис:

EEPROM {*Location*,}[*Constant*{,*Constant*...}]

Это команда записи во внутреннюю память EEPROM. По этой команде происходит запись значений - *Constant*{,*Constant*...} во внутреннюю энергонезависимою память микроконтроллера EEPROM на чинная с адреса - *Location* и далее последовательно. Если дополнительное значение *Location* в команде опущено, то первый байт массива - [*Constant*{,*Constant*...}] в команде EEPROM, сохраняется с адреса 0. Сохраняемые значения могут быть, как числовыми, так и строковыми. Сохраняется только самый младший байт числового значения. Строковые переменные сохраняются как последовательные байты кода ASCII. Никакой информации о длине или признака конца автоматически не добавляется.

Команда **EEPROM** работает только с микроконтроллерами, у которых память EEPROM находится внутри самого микроконтроллера. Это микроконтроллеры типа PIC16F84 и PIC16C84. Эта команда не будет работать с микросхемами памяти EEPROM, подключенными к микроконтроллеру по стандарту I2C, как например 12CE67x и 16CE62x.

Так как EEPROM - энергонезависимая память, то данные, записанные в эту память, останутся нетронутыми, даже если питание будет отключено. Такая память допускает многократную перезапись содержимого.

Эта команда аналогична команде **DATA**. В программе для чтения значений записанных в EEPROM может использоваться команда **READ**.

Пример:

```
' Запишем в память EEPROM значения 10, 20 и 30 начиная с адреса 5
EEPROM 5,[10,20,30]
```

Команда **ENABLE**.

Команда **ENABLE** разрешает программе производить обработку прерываний и выполнять команды отладки, которые до этого были отключены командой **DISABLE**. Команды **DISABLE** и **ENABLE** больше походят на псевдокоманды, так как они только дают указания компилятору, вместо того, чтобы фактически создавать код программы.

Пример:

```
DISABLE  ' Отключить обработку прерываний
myint: led = 1       ' Включить светодиод когда происходит прерывание
RESUME   ' Вернутся в основную программу
ENABLE   ' Включить обработку прерываний
```

Команда **ENABLE DEBUG.**

Команда **ENABLE DEBUG** запускает процесс обработки команд отладки, которые до этого были предварительно заблокированы командой **DISABLE** либо командой **DISABLE DEBUG**.

Команда **ENABLE INTERRUPT**.

Команда **ENABLE INTERRUPT** разрешает программе производить процесс обработки прерываний, которые до этого были заблокированы командой **DISABLE INTERRUPT** или **DISABLE**.

Команда **END**.

Команда **END** останавливает выполнение программы и вводит микроконтроллер в режим малого энергопотребления (Sleep). При этом все выводы микроконтроллера (вход либо выход) остаются в их текущем состоянии. Команда **END** работает, выполняя команду **SLEEP** непрерывно в цикле.

Совет: в конце каждой программы должны быть помещены команды **END**, или **STOP**, или **GOTO** для того чтобы препятствовать засорению свободной памяти микроконтроллера.

Команда **ERASECODE.**

Синтаксис:
ERASECODE *Block*

Некоторые PIC-микроконтроллеры, обладающие *флэш*-памятью, такие, например, как ряд PIC18Fxxx требуют, чтобы часть пространства памяти программ была очищена прежде, чем туда будет записан новый массив по команде **WRITECODE**. В этих микроконтроллерах, стирание памяти выполняется блоками. Блок стирания может быть в 32 слова (64 байта) или иметь другой размер, в зависимости от микроконтроллера. Для уточнения этих данных смотрите справочные листы от фирмы Microchip.

Начальный адрес блока, который будет стерт, определяется значением *Block*. *Block* – адрес размером в байт. Остерегайтесь определять блок, в котором может находиться код основной программы. При попытке использования этой команды для устройств, которые не поддерживают стирание блоками, вызовет ошибку компиляции.

Пример:

```
ERASECODE $100 ' Очистить блок памяти, начинающийся с адреса $100
```

Команда **FOR...NEXT.**

Синтаксис:

```
FOR Count = Start TO End {STEP {-} Inc}
   {Body}
NEXT {Count}
```

Это так называемый оператор цикла, который позволяет выполнять определенную группу операторов заданное число раз. Работает этот оператор так: сначала переменная-счетчик получает значение, равное значению выражения - *Start*. Затем производится сравнение этого счетчика со значением *End* и если значение *Start* меньше значения выражения -*End*, то выполняется блок операторов до ключевого слова **NEXT**, а затем значение переменной меняется на

величину приращения, равного значению выражения - *Inc* (шаг цикла), вновь производится проверка счетчика и т. д.

Шаг приращения может быть как положительным, так и отрицательным числом - тогда значение начального выражения - *Start* должно быть меньше значения конечного выражения - *End* (то есть, значение счетчика не увеличивается, а уменьшается). Если ключевое слово **STEP** опущено, то величина приращения по умолчанию равно 1. Если конечное значение переменной цикла больше чем 255, то должна использоваться переменная цикла размером в WORD.

Пример 1:

```
FOR i = 1 TO 10                 ' Счет ведется от 1 до 10
  SEROUT 0,N2400,[#i," "]       ' Посылаем на вывод Pin0 последовательный ряд
‘ чисел
NEXT i                          'Вернутся к началу и так далее
SEROUT 0,N2400,[10]             'Посылаем команду перевода строки
```

Пример 2:

```
FOR B2 = 20 TO 10 STEP -2       ' Счет ведется от 20 до 10 с шагом - 2
SEROUT 0,N2400,[#B2," "]        ' Посылаем на вывод Pin0
‘ последовательный ряд чисел
NEXT B2                         ' Вернутся к началу и так далее
SEROUT 0,N2400,[10]             'Посылаем команду перевод строки
```

Команда **FREQOUT**.

Синтаксис:

```
FREQOUT Pin,Onms,Frequency1{,Frequency2}
```

По этой команде на выводе Pin в течении периода - `Onms` формируется ШИМ сигнал подобный представленному на рисунке 20.

Рис. 34. Примерный вид сигнала ШИМ формируемого на выходе микроконтроллера по команде **FREQOUT**.

Если к этому выводу микроконтроллера подключить RC-фильтр, то на выходе фильтра будет получен синусоидальный сигнал частоты - `Frequency1`, либо сигнал, состоящий из двух синусоид с частотами `Frequency1` и `Frequency2`. Генерируемые частоты могут быть выбраны из диапазона 0 – 32767 герц.

Параметр – *Onms* определяет время генерации в миллисекундах. Вывод Pin автоматически становится выходом. Он может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут

меняться от 0 и до 15 (например, B0) или названием вывода (например, PORTA.0). Так как команда **FREQOUT** генерирует сигнал с использованием широтно-импульсной модуляцией, то от качества используемого фильтра зависит качество формы получаемой синусоиды. На рисунке 19 показана схема такого фильтра.

Команда **FREQOUT** рассчитана и лучше всего работает с задающим генератором 20 МГц. Но она может также работать и с генератором 10 МГц, и с генератором 4 МГц.

При разработке генераторов синусоиды либо комбинаций синусоид на основе этой команды необходимо учитывать, что чем ниже частота генерации, тем ниже уровень синусоиды на выходе фильтра.

Значение частот получаемых на выходе фильтра будет равно отношению частоты 20 МГц к частоте фактически используемого генератора. Т.е.:

$(Tone)_N = (Tone)_{20} * 20/N$

Где

N – частота фактически используемого генератора (в МГц),

$(Tone)_{20}$ – частота, которую вы бы получили, используя генератор 20 МГц,

$(Tone)_N$ – частота, которую получим, используя генератор с частотой N.

Пример:

```
'Выдать на вывод PORTB.1 частоту 1KHz в течении 2 секунд
FREQOUT PORTB.1,2000,1000
```

Команда **GOSUB**.

Синтаксис:

GOSUB *Label*

Эта команда указывает программе на то, что необходимо перейдите к подпрограмме, находящейся после метки *Label*, сохраняя при этом адрес возврата в стеке. По команде **GOSUB** в отличие от команды **GOTO**, при достижении ключевого слова **RETURN** программа должна вернуться на следующую строку после команды **GOSUB**. В программе может использоваться неограниченное число подпрограмм. В свою очередь, подпрограммы могут также быть вложенными. Другими словами, Вы можете из одной подпрограммы вызвать другую подпрограмму. Такое вложение подпрограмм ограничено. Глубина вложения не должна превышать четырех уровней.

Пример:

```
GOSUB beep              ' Выполните подпрограмму, по метке beep
...
beep: HIGH 0            'Включите светодиод подключенный к Pin0
```

```
SOUND 1,[80,10]  ' Выдать гудок на динамик подключенный к Pin1
LOW 0                    'Выключите светодиод
RETURN           'Возвратится в основную подпрограмму
```

Команда **GOTO**.

Синтаксис:
GOTO Label

По этой команде выполнение программы продолжается с утверждения, находящегося после метки Label.

Пример:

```
GOTO send                        'Перейти на мтку send
...
send: SEROUT 0,N2400,["Hi"]      'Послать текст“Hi”с  вывод Pin0
‘ последовательного порта
```

Команда **HIGH.**

Синтаксис:
HIGH *Pin*

Переводит указанный вывод в высокое логическое состояние. Контакт автоматически становится выходом. Параметр *Pin* может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0) или названием вывода (например, PORTA.0)).

Пример:

```
HIGH 0           ' Сделать Pin0 выходом и установить его в высокое состояние
‘ (+5 в)
HIGH PORTA.0     ' Сделать pin 0 PORTA, выходом и установить его в высокое
‘ состояние (+5 в)
led VAR PORTB.0          ' Определим переменную LED как PORTB.0
HIGH led         ' Сделать LED выходом и установить его в высокое состояние
‘ (+5 в)
```

С другой стороны, если этот контакт - уже установлен как выход, то намного более быстрый и более короткий путь (с точки зрения сгенерированной программы), установить его в высокое состояние, это:

```
PORTB.0 = 1      'установить PORTB.0 в высокое логическое состояние
```

Команда **HPWM**.

Синтаксис:
HPWM *Channel*,*Dutycycle*,*Frequency*

Данная команда генерирует на выводе микроконтроллера прямоугольные импульсы с частотой –*Frequency*. Параметр - *Channel* определяет вывод микроконтроллера, который будет использован для этого. Некоторые МК имеют 1, 2 или 3 канала PWM. Параметр - *Dutycycle* определяет скважность сигнала. Она может меняться от 0 и до 255, где 0 – соответствует нулевой скважности (генерация отсутствует, на выходе - 0), а 255 полное заполнение (генерация отсутствует, на выходе – 1). Значение 127 дает 50 %- скважность (меандр). Если разделить значение *Dutycycle* на 2.55, то мы получим скважность в %. Параметр *Frequency* - желательная частота сигнала PWM. На микроконтроллрах с 2 каналами, частоты в обоих каналах должны быть одинаковыми. Однако не все частоты доступны для использования в этой команде. Это зависит от задающего генератора. Самая низкая частота при кварце в 4 МГц – 245 Гц. Самая высокая частота на любой частоте задающего генератора – 32767 Гц.

Самые низкие возможные для использования частоты в команде HPWM в зависимости от используемого генератора показаны в таблице:

Таблица зависимости минимальной частоты в команде **HPWM** от частоты задающего генератора

Таблица 12.

Частота генератора, МГц	**Для 14-разрядных МК и 18Cxxx, Гц**	**Для 17Cxxx, Гц**
4	245	3907
8	489	7813
10	611	9766
12	733	11719
16	977	15625
20	1221	19531
24	1465	23438
33	2015	32227
40	2442	na

Некоторые устройства, типа PIC18C452, имеют дополнительные выводы, которые могут использоваться в качестве выхода для команды HPWM. Следующие **DEFINE** разрешают использовать эти выводы:

```
DEFINE CCP1_REG PORTC      ' Определяем порт для первого канала Hpwm
DEFINE CCP1_BIT 2      ' Определяем вывод для первого канала Hpwm
DEFINE CCP2_REG PORTC      ' Определяем порт для второго канала Hpwm
DEFINE CCP2_BIT 1      ' Определяем вывод для второго канала Hpwm
```

Следующие **DEFINE** определяют, какой таймер, 1 или 2, использовать с каналом PWM 2 и каналом PWM 3 для устройств типа PIC17C7xx. Если никакого **DEFINE** не будет предложено, то по умолчанию - таймер 1,.

```
DEFINE HPWM2_TIMER 1  'Для второго канала Hpwm использовать таймер 1
DEFINE HPWM3_TIMER 1  'Для третьего канала Hpwm использовать таймер 1
```

Пример:

```
HPWM 1,127,1000   ' Установить 50 % скважность для сигнала в 1 кГц
HPWM 1,64,200     ' Установить 25 % скважность для сигнала в 2 Гц
```

Команда **HSERIN**.

Синтаксис:

HSERIN {*ParityLabel*,}{*Timeout*,*Label*,}[*Item*{,...}]

Эта команда предназначена для приема информации по последовательному асинхронному каналу связи.

Её можно использовать только с микроконтроллерами, которые имеют в своем составе модуль USART. Для уточнения этой информации смотрите справочные листы на эти микроконтроллеры. Там же Вы сможете найти информацию о номерах контактов используемых для такой связи. Основные параметры, в том числе и скорость канала в бодах, должны быть определены командами **DEFINE**.

```
' Установить в регистре приемнике разрешение на прием
DEFINE HSER_RCSTA 90h

' Установить регистр передатчика разрешение на передачу
DEFINE HSER_TXSTA 20h

' Установить скорость в бодах
DEFINE HSER_BAUD 2400
```

```
' Установить непосредственно регистр SPBRG (обычно установлено
HSER_BAUD)
DEFINE HSER_SPBRG 25
```

Для оперирования скоростью в бодах команда **HSERIN** по умолчанию предполагает генератор на 4 МГц. Для того чтобы правильно рассчитать скорость в бодах с другими значениями генератора, воспользуйтесь соответствующей командой **DEFINE,** с помощью которой можно установить нужное значение частоты. Дополнительные параметры, такие как время ожидания -*Timeout* и метка -*Label* могут быть включены в команду для того, чтобы разрешить программе продолжиться, если данные не будут получены в определенное время, с указанием того, куда должна перейти программа для продолжения. Время ожидания определяется в единицах миллисекунд.

По умолчанию формат принимаемых данных - 8N1, т.е. 8 информационных разрядов, нет разряда четности и 1 стоповый бит. Но можно использовать и другие форматы, например 7E1 (7 информационных разрядов, проверка на четность, 1 стоповый бит) или 7O1 (7 информационных разрядов, проверка на нечетность, 1 стоповый бит). Для этого надо использовать один из следующих **DEFINE**:

```
' Если желаете используйте проверку на четность
DEFINE HSER_EVEN 1

' Если желаете, используйте проверку на нечетность
DEFINE HSER_ODD 1
```

Установка HSER_EVEN и HSER_ODD в 0 - отключит режим проверки на четность либо нечетность.

Установка проверки на четность, наряду со всеми другими **DEFINE** также будет касаться и команды **HSEROUT**. В команду может быть включен дополнительный параметр – *ParityLabel (* метка результата проверки на четность). Это значит, что программа продолжится с этого места, если в результате проверки на четность будет получена ошибка. Понятно, что это должно использоваться, если разрешена проверка на четность, которая задается в одном из предыдущих **DEFINE**. Так как последовательный прием данных осуществляется на основе аппаратных средств, то естественно не возможно перевести его в режим работы с инвертированными уровнями не заменив драйвер RS-232. Поэтому с командой **HSERIN** должен использоваться подходящий драйвер. Команда **HSERIN** поддерживает те же самые модификаторы данных, которые работают с командой **SERIN2**.

Таблица модификаторов используемых с командой **HSERIN**

Таблица 13

Модификатор	Операция

BIN{1..16}	Прием двоичных значений
DEC{1..5}	Прием десятичных значений
HEX{1..4}	Прием шестнадцатеричных значений
SKIP n	Пропустить прием n символов
STR ArrayVar\n{\c}	Принять строку в n символов, произвольно законченных на символе c
WAIT ( )	Ждать последовательность символов
WAITSTR ArrayVar{\n}	Ждать строку символов

В микроконтроллерах с двумя последовательными портами (типа 17Cxxx), команда **HSERIN** получает доступ только к первому порту. Второй порт должен быть установлен и прочитан, получая доступ к регистраторам непосредственно.

Определения - HSER_RCSTA, HSER_TXSTA и HSER_SPBRG устанавливают в PIC-микроконтроллере соответствующий регистр RCSTA, TXSTA и SPBRG в необходимое состояние. Для получения дополнительной информации о каждом из этих регистров, смотрите справочные данные фирмы Microchip для выбранного Вами устройства.

Бит BRGH (бит 2) регистра TXSTA управляет высокоскоростным режимом генератора управляющего скоростью передачи информации. Определенные скорости в бодах при определенных частотах генератора требуют, чтобы этот бит был установлен должным образом. Чтобы сделать это, установите HSER_TXSTA в 24-ый вместо 20-ого. Смотрите справочные листы от фирмы Microchip в таблице скорость в бодах последовательного порта микроконтроллера.

Поскольку последовательный порт микроконтроллера имеет только 2-байтовый входной буфер, он может легко переполниться, если символы не считываются из него достаточно быстро. Когда это произойдет, USART прекращает принимать новые символы и должен быть сброшен. Эта ошибка переполнения может быть сброшена, переключением бита CREN в регистре RCSTA. Для того чтобы автоматически очищать эту ошибку может использоваться следующий **DEFINE**:

```
DEFINE HSER_CLROERR
```

Однако, в этом случае Вы не будете знать, о том что ошибка произошла, и символы, возможно, будут потеряны. Для того чтобы вручную очищать ошибку переполнения следует записать:

```
RCSTA.4 = 0
```

```
RCSTA.4 = 1
```

Пример:

```
B0 var byte
W1 var word

Main:
    HSERIN [B0, dec W1]   ‘Принять десятичные значения по последовательному каналу
    Goto Main
```

Необходимо обратить Ваше внимание на то, что команда **HSERIN** выполняет ту же задачу что и команда **SERIN**. Разница их только в том, что первая команда работает только с микроконтроллерами оснащенными модулем USART, а вторая с любым микроконтроллером.

Команда **HSEROUT**.

Синтаксис:
HSEROUT [*Item* {,*Item...*}]

Эта команда предназначена для передачи информации по последовательному асинхронному каналу связи.

Её можно использовать только с микроконтроллерами, которые имеют в своем составе модуль USART. Для уточнения этой информации смотрите справочные листы на эти микроконтроллеры. Там же Вы сможете найти информацию о номерах контактов используемых для такой связи. Основные параметры, в том числе и скорость канала в бодах, должны быть определены командами **DEFINE**.

```
' Установить регистр приемника разрешение на прием
DEFINE HSER_RCSTA 90h

' Установить регистр передатчика разрешение на передачу
DEFINE HSER_TXSTA 20h

' Установить скорость обмена информацией в бодах
DEFINE HSER_BAUD 2400

' Установить непосредственно регистр SPBRG (обычно установлено HSER_BAUD)
DEFINE HSER_SPBRG 25
```

Для вычисления скорости передачи/приема информации в бодах, команда **HSEROUT** по умолчанию предполагает генератор на 4 МГц. Для того чтобы правильно рассчитать выбор времени скорости в бодах с другими значениями

генератора, убедитесь, что соответствующий **DEFINE** определил нужное значение частоты. По умолчанию формат передаваемых данных - 8N1, т.е. 8 информационных разрядов, нет разряда четности и 1 стоповый бит. Но можно использовать и другие форматы, например 7E1 (7 информационных разрядов, проверка на четность, 1 стоповый бит) или 7O1 (7 информационных разрядов, проверка на нечетность, 1 стоповый бит). Для этого надо использовать один из следующих **DEFINE**:

```
' Использовать проверку на четность
DEFINE HSER_EVEN 1

' Использовать проверку на нечетность
DEFINE HSER_ODD 1
```

Установка проверки на четность, наряду со всеми другими **DEFINE** также будет касаться и команды **HSERIN**. Команда **HSEROUT** поддерживает те же самые модификаторы данных, которые используются и с командой **SEROUT2** (см. таблицу 11).

В микроконтроллерах типа 17Cxxx с 2 последовательными портами, команда **HSEROUT** получает доступ только к первому порту. Второй порт должен быть установлен и прочитан, используя непосредственный доступ к регистрам.

```
'Послать десятичное значение B0, код операции  перевод строки с помощью
встроенного модуля USART
HSEROUT [dec B0,10]
```

Команда **HSERIN2.**

Синтаксис:
HSERIN2 {ParityLabel,}{Timeout,Label,}[Item{,...}]

Эта команда предназначена для микроконтроллеров имеющих в своем составе два модуля USART. По этой команде микроконтроллер получает значения по второму последовательному порту микроконтроллера. Для уточнения этой информации смотрите справочные листы на микроконтроллеры относительно последовательного асинхронного порта и контактов используемых для этой цели. **HSERIN2** - одна из нескольких встроенных функций предназначенных для работы с асинхронными последовательными портами. Основные параметры последовательной передачи данных и скорость в бодах определяются использованием соответствующих **DEFINE**:

```
' Установить регистр приемника на получение информации
DEFINE HSER2_RCSTA 90h
```

```
' Установить регистр передатчика на выдачу информации
DEFINE HSER2_TXSTA 20h

' Установить скорость в бодах при приеме и передачи информации
DEFINE HSER2_BAUD 2400

' Установить SPBRG непосредственно (обычно устанавливается HSER_BAUD)
DEFINE HSER2_SPBRG 25
```

Для расчета скорости в бодах команда **HSERIN2** по умолчанию предполагает генератор на 4 МГц. Для того чтобы правильно пересчитать скорость в бодах для других значений генератора, введите соответствующую команду-определение **DEFINE**, устанавливающую новое значение генератора. Дополнительное время ожидания -*Timeout* и метка *Label* могут быть включены в команду, для того чтобы программа могла продолжиться, даже если информация не будет получена в этом временном интервале. Время ожидания определяется в единицах миллисекунд. По умолчанию последовательные данные передаются в формате 8N1, 8 информационных разрядов, нет разряда проверки на четность и 1 стоповый бит. Другие форматы, такие как 7E1 (7 информационных разрядов, проверка на четность, 1 стоповый бит) или 7O1 (7 информационных разрядов, проверка на нечетность, 1 стоповый бит) можно установить, используя один из следующих **DEFINE**:

```
' Установить проверку на четность
DEFINE HSER2_EVEN 1

' Установить проверку на нечетность
DEFINE HSER2_ODD 1
```

Все эти установки служат для управления командами как **HSERIN2** так и **HSEROUT2**. Еще один дополнительный параметр - - *ParityLabel* может быть также включен в команду. Он предназначен для того, чтобы указать программе с какого места она должна будет продолжиться, если будет получен байт с ошибкой четности либо нечетности в зависимости от того, что проверяется. Команда **HSERIN2** поддерживает те же самые модификаторы данных, что и **SERIN2** (см. таблицу 13).

Команда **HSEROUT2.**

Синтаксис:

```
HSEROUT2 [Item {,Item...}]
```

Эта команда предназначена для передачи одного или несколько байтов со второго последовательного порта микроконтроллера, который поддерживает асинхронную передачу данных. Эта команда может использоваться только с теми

микроконтроллерами, которые имеют в своем составе второй модуль USART. Для получения полной информации смотрите справочные листы данных на используемый микроконтроллер. Параметры последовательного порта и скорость в бодах определяются, соответствующими **DEFINE**:

```
' Установить регистр 2 приемника на получение информации
DEFINE HSER2_RCSTA 90h

' Установить регистр 2 передатчика на выдачу информации
DEFINE HSER2_TXSTA 20h

' Установить скорость приема-передачи в бодах для 2 приемника и передатчика
DEFINE HSER2_BAUD 2400

' Установить SPBRG непосредственно (обычно устанавливается HSER_BAUD)
DEFINE HSER2_SPBRG 25
```

В команде **HSEROUT2**, для расчета скорости передачи в бодах, по умолчанию предполагается генератор на 4 МГц. Для того чтобы команда правильно пересчитывала скорость для других значений генератора, введите соответствующий **DEFINE**, устанавливающий новое значение генератора. По умолчанию последовательные данные передаются в формате 8N1, 8 информационных разрядов, нет разряда проверки на четность и 1 стоповый бит. Другие форматы, такие как 7E1 (7 информационных разрядов, проверка на четность, 1 стоповый бит) или 7O1 (7 информационных разрядов, проверка на нечетность, 1 стоповый бит) можно установить, используя один из следующих DEFINE:

```
' Установить проверку на четность для 2 канала приема-передачи
DEFINE HSER2_EVEN 1

' Установить проверку на нечетность для 2 канала приема-передачи
DEFINE HSER2_ODD 1
```

Установка проверки на четность, наряду с другими (**DEFINE** HSER2_) служат для управления командами как **HSERIN2** так и **HSEROUT2**. Команда **HSEROUT2** поддерживает те же самые модификаторы данных, что и команда **SEROUT2** (см. таблицу 11).

```
'Послать десятичное значение B0, код перевода строки с помощью второго

‘ встроенного USART передатчика
HSEROUT2 [dec B0,10]
```

Команда **I2CREAD**.

Синтаксис:

I2CREAD
DataPin,ClockPin,Control,{*Address,*}[*Var*{,*Var...*}]{,*Label*}

По этой команде происходит чтение информации от внешних устройств, работающих по протоколу I^2C. Через контакты - *ClockPin* (синхронизация) и - *DataPin* (данные) посылается байт управления - *Control* и байты адреса – *Addres*. Данные, полученные в ответ на это, сохраняются последовательно в переменных – *Var*. Параметры *ClockPin* и *DataPin* могут быть константами, значения котор*ых* лежат в пределах от 0 и до 15, или переменн*ыми*, значения котор*ых* могут меняться от 0 и до 15 (например, B0) или названием вывода (например, PORTA.0).

Команды **I2CREAD** и **I2CWRITE** могут использоваться, для чтения и записи данных во внешнюю энергонезависимую память EEPROM по 2-х проводному интерфейсу I^2C. Это микросхемы типа 24LC01B и ей подобные. Использование этих микросхем позволяет данным быть сохраненными во внешней энергонезависимой памяти и по необходимости быть востребованными так, что даже при выключении питания они будут сохранены. Эти команды работают в режиме master шины I^2C и могут использоваться для обмена данными с другими устройствами по интерфейсу I^2C. Это, например, температурные датчики и внешние АЦП.

Только для 12-разрядных PIC-микроконтроллеров при работе по протоколу I2C, вначале основного текста программы необходимо указать выводы данных и синхронизации с помощью соответствующих DEFINE. И хотя указания на эти выводы должны присутствовать в команде **I2CREAD**, но компилятор эту информацию игнорирует.

```
DEFINE I2C_SCL PORTA,1  ' Только для 12-разрядных микроконтроллеров
DEFINE I2C_SDA PORTA,0  ' Только для 12-разрядных микроконтроллеров
```

Старшие 7 разрядов служебного байта содержат управляющий код выбора чипа или дополнительной информацией об адресе, в зависимости от выбранного устройства. Младший разряд - внутренний флаг, указывающий на текущий режим – чтение или запись. Этот формат служебного байта отличен от формата первоначального компилятора PicBasic. Убедитесь, что использовали этот формат при работе с шиной I^2C. Например, общаясь с 24LC01B, управляющий код - %1010, разряды выбора микросхемы в этом случае не используются, таким образом, служебный байт будет %10100000 или $A0. Форматы служебных байтов для некоторых других микросхем приведены в таблице:

Таблица 14.

Микросхема	Емкость	Управление	Размер адрса
24LC01B	128 байт	%1010xxx0	1 байт

24LC02B	256 байт	%1010xxx0	1 байт
24LC04B	512 байт	%1010xxb0	1 байт
24LC08B	1K байт	%1010xbb0	1 байт
24LC16B	2K байт	%1010bbb0	1 байт
24LC32B	4K байт	%1010ddd0	2 байта
24LC65	8K байт	%1010ddd0	2 байта

bbb = выбор разрядов заблокирован(адрес старшего разряда)

ddd = разряды выбора устройства

xxx = не заботится

Посылаемый размер адреса (байт или слово) определен размером используемой переменной. Если для адреса используется переменная размером в байт, то посылается 8-разряный адрес. Если используется переменная размером в слово, то посылается 16-разрядный адрес. Убедитесь, что Вы используете переменную надлежащего размера для выбранного устройства. В качестве адреса не должны использоваться константы, поскольку размер может измениться в зависимости от размера константы. Если переменная определена размером в слово, то будут прочитаны 2 байта и будет сохранен вначале старший байт переменной, а затем младший байт. Этот порядок отличен от того, каким обычно сохраняются переменные, т.е. вначале сохраняется младший байт, а затем старший.

Перед именем переменной может быть включен модификатор STR. В этом случае можно загрузить все множество строковых переменных сразу. Если модификатор STR присутствует, то следующим элементом должно быть имя переменной размером в байт или слово, сопровождаемо*й* левой наклонной чертой (\) и количеством посылаемых знаков:

```
a VAR byte[8]
I2CREAD PORTC.4,PORTC.3,$a0,0,[STR a\8]
```

Если определено множество переменных размером в слово, то при чтении элементов этого множества, - вначале читается младший байт. Если будет включена дополнительная метка -*Label*, то программа перейдет на эту метку, как только будет получено подтверждение от устройства I^2C.

Для 12CE67x устройств, вывод данных - GPIO.6, а вывод синхроимпульсов - GPIO.7. Для 16CE62x устройств, вывод данных - EEINTF.1, а синхроимпульсов - EEINTF.2. Для уточнения этой информации смотрите справочные листы данных фирмы Microchip на эти устройства.

В командах I2C стандартные установки скорости (100 кГц) будут доступны при тактовых частотах до 8 МГц. Для быстрого режима (400 кГц) должны использоваться генераторы до 20 МГц. Если Вы хотите использовать стандартные

установки скорости на частоте 8 МГц и выше то добавьте в начале текста программы следующее определение:

```
DEFINE I2C_SLOW 1
```

В связи с тем, что размер памяти и стека у 12-разрядных микроконтроллеров ограничен, это DEFINE с этими микроконтроллерами ничего не изменяет. Поэтому для стандартных скоростей должны использоваться генераторы до 4 MHz. А для скоростей – 400 кГц должны использоваться генераторы выше 4 МГц.

Передача информации по шине I^2C может быть приостановлена приемным устройством. Для этого надо установить вывод синхросигнала в низкое логическое состояние. Чтобы это разрешить добавьте в начале программы следующее определение:

```
DEFINE I2C_HOLD 1
```

Цепь синхронизации I^2C и вывод данных должны быть подключены к выводу питания Vcc через подтягивающий резистор в 4.7 кОм, поскольку к этим выводам подключены устройства, которые работают двунаправленным способом с открытым коллектором (см. рисунок 40). Для того чтобы сделать вывод синхронизации I^2C биполярным вместо вывода с открытым коллектором необходимо добавить в начале программы следующее определение:

```
DEFINE I2C_SCLOUT 1
addr VAR byte
cont CON %10100000
addr = 17 ' Установить адрес 17

' Считать данные по адресу 17 и записать их в переменную B2
I2CREAD PORTA.0,PORTA.1,cont,addr,[B2]
```

Для получения дополнительной информации об этих и других запоминающих устройствах, которые могут использоваться с командами **I2CREAD** и **I2CWRITE**, смотрите справочные материалы в “Non-Volatile Memory Products Data Book”.

Команда **I2CWRITE.**

Синтаксис:

I2CWRITE
DataPin,ClockPin,Control,{*Address,*}[*Value*{,*Value...*}]{,*Label*}

Команда **I2CWRITE** служит для записи информации во внешние устройства. Микроконтроллер по этой команде посылает через выводы *DataPin и ClockPin*

сигналы управления и выбора адреса. Контакты *ClockPin* и *DataPin* могут быть константами, значения которых лежат в пределах от 0 и до 15, или переменными, значения которых могут меняться от 0 и до 15 (например, B0) или названием вывода (например, PORTA.0). Размер адреса (байт или слово) определяется размером переменной, которая используется для этого. Вы должны контролировать, что правильно определили размер переменной в зависимости от устройства, с которым Вы желаете общаться. Для этих целей нельзя использовать константы.

Только для 12-разрядных PIC-микроконтроллеров при работе по протоколу I2C, вначале основного текста программы необходимо указать выводы данных и синхронизации с помощью соответствующих DEFINE. И хотя указания на эти выводы должны присутствовать в команде **I2CWRITE**, но компилятор эту информацию игнорирует.

```
DEFINE I2C_SCL PORTA,1   ' Только для 12-разрядных микроконтроллеров
DEFINE I2C_SDA PORTA,0   ' Только для 12-разрядных микроконтроллеров
```

При записи в последовательную EEPROM необходимо подождать 10 мс прежде, чем вновь обращаться к немй. Если Вы попытаетесь использовать команды **I2CREAD** или **I2CWRITE** прежде, чем пройдет указанный промежуток времени (запись еще полностью не закончена), то доступ будет игнорироваться. Обычно для записи 1 байта информации используется одна команда **I2CWRITE**. После чего идет время ожидания конца записи. При попытке записать несколько байтов информации может быть нарушено вышеупомянутое требование выбора времени для последовательного СППЗУ. Некоторые же последовательные EEPROM разрешают записывать множество байтов в отдельную страницу но затем все равно сделать необходимую паузу. Для уточнения этих деталей смотрите справочные листы данных для выбранных устройств. Возможность *м*ногократной побайтовой записи может быть полезной с устройствами, работающими с шиной I^2C, кроме последовательных СППЗУ, которые не допускают пауз между циклами записи. Если информационное значение определено размером в слово, то посылаются 2 байта. Вначале старший байт, а затем младший. Этот порядок отличен от того, как обычно сохраняются переменные, то есть, сначала сохраняется младший байт а затем стпрший.

Перед именем переменной может быть включен модификатор STR. Это может потребоваться для того, чтобы записать массив строковых переменных в последовательную EEPROM в постраничном режиме. Данные должны вписаться в отдельную страницу SEEPROM. Размер страницы зависит от специфического устройства SEEPROM. Если присутствует модификатор **STR**, то следующим должно быть имя переменной (размером в слово или байт), а затем наклонная черта влево (\) и количество записываемых значений:

```
a VAR byte[8]
I2CWRITE PORTC.4,PORTC.3,$a0,0,[STR a\8]
```

Если множество переменных определено размером в слово, то сначала записывается младший байт. Это - противоположно тому, как обычно записываются простые слова, и совместимо со способом, которым компилятор обычно сохраняет переменные размером в слово. Если в команде будет включена дополнительная метка -*Label*, то программа перейдет на эту метка, если не будет получено подтверждение от устройства I^2C.

Выбор времени для команд I^2C установлен так, что стандартные установки скорости (100 КГц) будут доступны для тактовых частот до 8 МГц. В режиме быстрых устройств (400 кГц) могут использоваться до 20 МГц. Если Вы желаете получить доступ к стандартному устройству скорости для вышеупомянутой 8 МГц, то в начале программы должно быть добавлено следующее определение:

```
DEFINE I2C_SLOW 1
```

Передача по шине I2C может быть приостановлена ведомым устройством. Для этого необходимо установить низкий логический уровень в цепи синхронизации (это не поддерживается 12-разрядными микроконтроллерами). Чтобы это разрешить надо добавить в начале программы следующую строку:

```
DEFINE I2C_HOLD 1
```

Чтобы сделать цепь синхронизации шины I^2C биполярной, вместо цепи с открытым коллектором, введите в начале Вашей программы следующее определение:

```
DEFINE I2C_SCLOUT 1
```

Пример:

```
addr VAR byte
cont CON %10100000
addr = 17     ' Установить адрес 17
' Послать 6 байтов начиная с адреса 17
I2CWRITE PORTA.0,PORTA.1,cont,addr,[6]
PAUSE 10  ' Ждем 10 мс пока запишется
addr = 1 'Устанавливаем адрес 1

' Отправим байт в B2 по адресу 1
I2CWRITE PORTA.0,PORTA.1,cont,addr,[B2]
PAUSE 10 'Ждем 10 мс пока запишется
```

Команда **IF...THEN.**

Синтаксис:

```
IF Comp {AND/OR Comp...} THEN
   Statements...
ELSE
   Statements...
ENDIF
```

Этот оператор выполняет одно или несколько сравнений. Каждый элемент набора может связывать переменную с константой или другой переменной и включает один из операторов сравнения. Выражение **If..Then** производит сравнение и оценивает истинно оно или ложно. Если оно оценено как истинное, то выполняется операция, стоящая после слова **Then**. Если же оно оценено как ложное, то операция, стоящая после **Then** не выполняется. Сравнения, в результате которых получается 0, считаются ложными. Любое другое значение считается истинным. Все аргументы сравнения должны быть без знака. Используйте круглые скобки для того чтобы определить порядок, в котором должны выполняться операции сравнения.

Оператор **IF..THEN** может условно выполнить группу утверждений после **Then**. Утверждения должны сопровождаться словом **ELSE** или **ENDIF**, чтобы верно закончить структуру.

```
IF B0 <> 10 THEN
  B0 = B0 + 1
  B1 = B1 - 1
ENDIF

IF B0 = 20 THEN
  led = 1
ELSE
  led = 0
ENDIF
```

Команда **INPUT.**

Синтаксис:

```
INPUT Pin
```

Эта команда делает указанный контакт входом. Контакт может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0) или названием вывода (например, PORTA.0).

Пример:

```
INPUT 0 'Сделать Pin0 входом
INPUT PORTA.0 'Сделать pin 0 PORTA, входом
```

Однако, любой вывод можно сделать входом намного более быстрым и более коротким способом (с точки зрения сгенерированной программы) это:

```
TRISB.0 = 1 'Установить pin 0  PORTB, в состояние входа
```

Все выводы порта можно сделать входами, если записать 1 сразу в весь регистр TRIS:

```
TRISB = %11111111 ' Установить все выводы PORTB в состояние входа
```

Команда **LET.**

Синтаксис:

{**LET**} *Var* **=** *Value*

Команда LET помещает значение выражения в переменную. Выражение может быть константой, переменной или значением некоторого другого выражения. Обычно, дополнительное управляющее слово **LET** исключается.

Пример:

```
LET B0 = B1 * B2 + B3
B0 = SQR W1
```

Во втором выражении управляющее слово **LET** – опущено.

Команда **LCDIN**.

Синтаксис:

LCDIN {*Address*,}**[***Var*{**,***Var...*}**]**

Эта команда считывает данные из ОЗУ ЖК-монитора по указанному адресу – *Address* и считанные значения сохраняет в переменной - *Var*.

ЖК мониторы имеют встроенное ОЗУ, которое используется для хранения выводимых знаков. Большинство ЖК мониторов имеет большое доступное ОЗУ, которое является необходимым для визуализируемой области. Данные в это ОЗУ могут быть записаны, используя команду **LCDOUT**. Команда же **LCDIN** позволяет считывать эту информацию из ОЗУ.

Знакогенератор ОЗУ работает с адресами от $40 до $7f. Стартовый адрес данных ОЗУ дисплея начинается с адреса $80. Смотрите справочные листы

данных для определенного типа ЖК монитора относительно его функций и адресов.

Для работы этой команды необходимо соединить цепь чтение/запись (R/W) ЖК монитора с соответствующим выводом PIC контроллера. Состояние ее, 0 или 1, определяет, то какая команда в данный момент используется – чтения (**LCDIN**) или записи (**LCDOUT**). А в начале программы записать два **DEFINE** указывающие на адрес вывода:

```
DEFINE LCD_RWREG PORTE        'Подключить цепь R/W ЖК монитора к PORTE
DEFINE LCD_RWBIT 2     'Подключить цепь R/W ЖК монитора к выводу 2
```

Команда **LCDOUT**.

Синтаксис:

```
LCDOUT Item {,Item...}
```

Эта команда выводит информацию на экран алфавитно-цифрового ЖК монитора. Компилятор PicBasicPro поддерживает модули ЖК мониторов с контроллерами типа Hitachi 44780 или эквивалентными ему. Эти мониторы обычно имеют 14 или 16 контактов - или двухрядный разъем.

Если элементу предшествует признак (**#**), то это значит, что на ЖК монитор посылается ASCII код (это утверждение справедливо для всех микроконтроллеров кроме 12-разрядных). С командой **LCDOUT** может также использоваться любой из модификаторов, используемых с командой **SEROUT2** (см. таблицу 11).

Может потребоваться относительно длительное время для запуска ЖК монитора после включения питания. Поэтому программа должна выждать, по крайней мере, 0.5 секунды перед посылкой первой команды на ЖК монитор. В таблице 15 перечислены некоторые полезные команды, в которых присутствует код $FE.

Таблица 15

Команда	Операция
$FE, 1	Очистить дисплей
$FE, 2	Вернутся домой (идти на первую строку и первое знакоместо)
$FE, $0C	Выключить курсор
$FE, $0E	Курсор в форме подчеркивания
$FE, $0F	Включить мигающий курсор
$FE, $10	Сдвинуть курсор влево на одну позицию
$FE, $14	Сдвинуть курсор вправо на одну позицию
$FE, $C0	Переместить курсор на начало второй строки

$FE, $94	Переместить курсор на начало третьей строки
$FE, $D4	Переместить курсор на начало четвертой строки

Замечание. Для большинства ЖК мониторов отображаемые знакоместа и строки размещены не последовательно в памяти дисплея. Поэтому может быть промежуток между знакоместами. Большинство мониторов имеют формат экрана 16x2, и первая строка начинается со значения $0, а вторая строка со значения $40. Поэтому команда:

```
LCDOUT $FE, $C0
```

Сообщает монитору, что показ знаков надо начинать с начала второй строки. В мониторах с форматом экрана 16x1 знакомест обычно форматируются как 8x2 знакомест*а* с промежутком в памяти между первыми и вторыми 8 знаками.

Экраны с 4 строками также имеют перепутанную карту памяти. Поэтому смотрите справочные листы данных для каждого конкретного ЖК монитора относительно расположения в памяти знаков и дополнительных команд.

Пример:

```
LCDOUT $FE, 1, "Hello"     ' Очистить экран и вывести “Hello”
LCDOUT B0, #B1
```

ЖК монитор может быть связан с PIC микроконтроллером с использованием 4 или 8-разрядной шины. Если используется 8-разрядная шина, то все 8 цепей должны быть подключены к одному порту. Если же используется 4-разряднаяя шина, то она должна быть связана или со старшими или с младшими 4 разрядами одного порта. Цепи Enable (E) и Select Register (SR) могут быть связаны с контактами любого порта. Цепь R/W должна быть подключена к цепи “земля”, поскольку команда **LCDOUT** работает только на запись в ЖК монитор. Вообще в PicBasicPro, если не указано иначе, ЖК монитор подключается к микроконтроллеру к определенным контактам. Это означает, что, если ЖК монитор будет соединен 4-разрядной шиной, то цепи данных DB4 - DB7, подключаются к контактам PORTA.0 - PORTA.3, Select Register к PORTA.4, а Enable к PORTB.3. Чтобы изменять эти установки, необходимо поместить в начале программы следующие переопределения:

```
' Определим порт данных
DEFINE LCD_DREG PORTB
' Установим, стартовый информационный разряд (0 или 4), если 4-разрядная шина
DEFINE LCD_DBIT 4
' Установим порт к которому подключается цепь Register Select
DEFINE LCD_RSREG PORTB
```

```
' Установим разряд порта к которому подключается цепь Register Select
DEFINE LCD_RSBIT 1
' Установим порт к которому подключается цепь Enable
DEFINE LCD_EREG PORTB
' Установим разряд порта к которому подключается цепь Enable
DEFINE LCD_EBIT 0
' Установим тип шины (4 или 8 разрядов)
DEFINE LCD_BITS 4
‘Установим тип LCD (количество строк на экране)
DEFINE LCD_LINES 2
' Установить команду задержки в мкс
DEFINE LCD_COMMANDUS 2000
' Установить время задержки данных в мкс
DEFINE LCD_DATAUS 50
```

Эти установки сообщат PicBasicPro, что используется ЖК монитор с 2 строками, связанный 4-разрядной шиной данных подключенной к старшим 4 разрядам PORTB, цепь Select Register подключена к PORTB.1, а цепь Enable к PORTB.0 (см. рисунок 35).

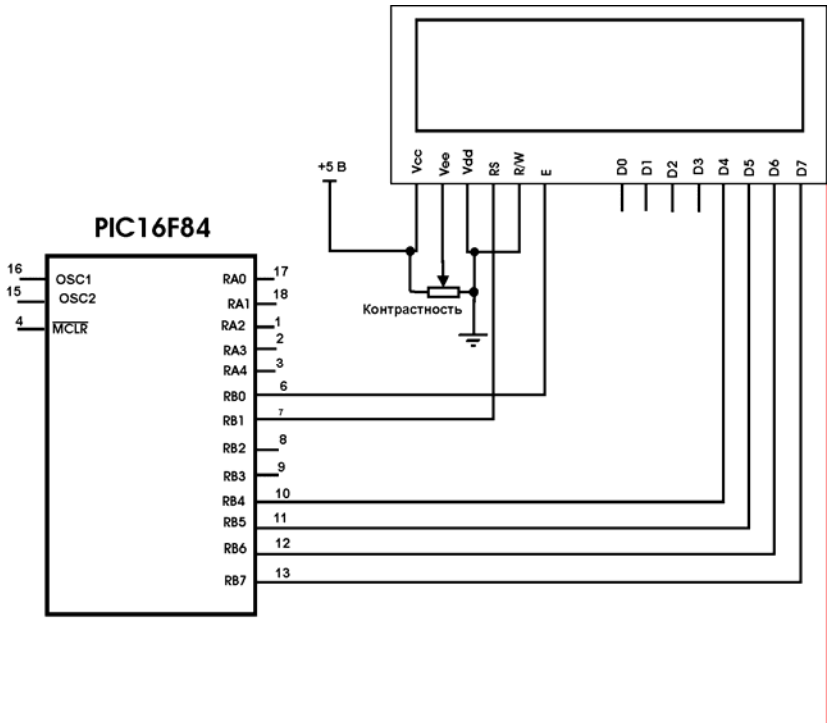


Рис. 35. Вариант схемы подключения алфавитно-цифрового ЖКИ к PIC16F84 с использованием 4-проводной шины.

Команда **LOOKDOWN**.

Синтаксис:

```
LOOKDOWN Search,[Constant{,Constant...}],Var
```

Команда **LOOKDOWN** ищет в списке 8-битовых констант значение - *Search*. Как только это значение будет найдено, его порядковый номер будет сохранен в переменной *-Var*. Если значение находится вначале списка, то переменной присваивается значение ноль. Если она вторая в списке, переменная получает значение 1. И так далее. Если в приведенном списке не было найдено искомого значения, то переменная остается неизменной. Список констант может быть смесью как числовых, так и строковых констант. Каждый знак в строковой переменной обрабатывается как отдельная константа с ASCII значением.

Массивы, составленные из переменных с переменным индексом не могут использоваться в команде **LOOKDOWN**, хотя массивы, составленные из переменных с постоянными индексами разрешены. До 255 констант разрешено использовать в списке (256 для 18Cxxx).

Пример:

SERIN 1,N2400,B0 *' Получить шестнадцатеричное значение по последовательному каналу связи на выводе Pin1 и присвоить его переменной B0*
LOOKDOWN B0,["0123456789ABCDEF"],B1 *' Преобразуйте шестнадцатеричный значение переменной B0 в десятичное значение B1*

SEROUT 0,N2400,[#B1]ë *'Пошлите значение переменной B1 по последовательному каналу связи через вывод Pin0*

Команда **LOOKDOWN2**.

Синтаксис:
LOOKDOWN2 *Search*,{*Test*}[*Value*{,*Value...*}],*Var*

Команда **LOOKDOWN**2 ищет в списке 8-битовых констант значение - *Search*. Как только это значение будет найдено, его порядковый номер будет сохранен в переменной -*Var*. Если значение находится вначале списка, переменной присваивается значение ноль. Если она вторая в списке, переменная получает значение 1, и так далее. Если в приведенном списке не было найдено искомого значения, то переменная остается неизменной.

В этой команде используется дополнительный параметр *Test*, с помощью которого можно осуществить выборку индекса значения большего или меньшего, из предложенного. Для проверки используются знаки « < » или « > ». Например, при использовании знака « >» может быть отыскано первое значение в списке, которое больше, чем параметр *Search*. Если знака проверки нет, то принимается, что использован знак «=». Список значений может быть смесью 16-разрядных числовых и строковых констант и переменных. Каждый знак в строковой переменной обрабатывается как отдельная константа, значение которой равно ASCII коду этого знака. Выражения не могут использоваться в списке значений, хотя они могут использоваться как значения *Search*. Массивы, составленные из переменных с переменными индексами, не могут использоваться в команде **LOOKDOWN2**, хотя массивы, составленные из переменных с

постоянными индексами, разрешены. Разрешено использовать в списке до 85 (256 для 18Cxxx) значений.

LOOKDOWN2 генерирует программу, которая приблизительно в 3 раза больше, чем программа, сгенерированная командой **LOOKDOWN**. Если список для поиска составлен только из 8-разрядных констант или строковых переменных, то, понятно, что предпочтительней использовать команду **LOOKDOWN**.

Пример:

```
LOOKDOWN2 W0,[512,W1,1024],B0
LOOKDOWN2 W0,<[10,100,1000],B0
```

Команда **LOOKUP**.

Синтаксис:
LOOKUP *Index*,[*Constant*{,*Constant...*}],*Var*

Команда **LOOKUP** используется для того, чтобы выбрать значение из таблицы 8-разрядных констант в соответствии с параметром *Index* и присвоить это значение переменной - *Var*. Если значение *Index* равно нулю, то выбирается значение первой константы. Если *Index* равен единице, то выбирается значение второй константы. И так далее. Если *Index* больше или равен числу констант в списке значений, то никакое действие не предпринимается, и переменная остается неизменной. Список констант может быть смесью числовых и строковых констант. Каждый знак в строковой переменной обрабатывается как отдельная константа, равная по значению ее ASCII коду. Массивы, составленные из индексированных переменных не могут использоваться в команде **LOOKUP**, хотя массивы составленные из индексированных констант разрешены для использования. Списки констант могут содержать до 255 (256 для 18Cxxx) значений.

Пример:

```
FOR B0 = 0 TO 5                 ' Считаем от 0 до 5
LOOKUP B0,["Hello!"],B1    ' Выбрать из символьных значений Hello!
Значение под номером B0 и присвоить это значение переменной B1
SEROUT 0,N2400,[B1]             ' Отправить по последовательному каналу
значение B1
NEXT B0                         'Перейти к следующему символу
```

Команда **LOOKUP2**.

Синтаксис:
LOOKUP2 *Index*,[*Value*{,*Value...*}],*Var*

Команда **LOOKUP2** используется для того, чтобы выбрать значение из таблицы 8-разрядных констант в соответствии с параметром *Index* и присвоить это значение переменной - *Var*. Если значение *Index* равно нулю, то выбирается значение первой константы. Если *Index* равен единице, то выбирается значение второй константы. И так далее. Если *Index* больше или равен числу констант в списке значений, то никакое действие не предпринимается, и переменная остается неизменной. Список значений может быть смесью 16-разрядных числовых и строковых констант и переменных. Каждый знак в строковой переменной обрабатывается как отдельная константа равная значению ASCII кода. Выражения не могут использоваться в списке значений, хотя они могут использоваться как значение *Index*. Массивы, составленные из переменных не могут использоваться с командой **LOOKUP2**. В команде **LOOKUP2** можно использовать список до 85 (256 для 18Cxxx) значений.

Команда **LOOKUP2** генерирует программу, которая приблизительно в 3 раза больше, чем программа, сгенерированная командой **LOOKUP**. Если список значений составлен только из 8-разрядных констант и строковых переменных, то лучше использовать команду **LOOKUP**.

Пример:

```
LOOKUP2 B0,[256,512,1024],W1
```

Команда **LOW**.

Синтаксис:

LOW *Pin*

Команда **LOW** устанавливает на указанном контакте низкий логический уровень. Этот контакт автоматически становится выходом. Номер контакта может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0) или названием вывода (например, PORTA.0).

Пример:

LOW 0 ' *Сделать Pin0 выходом и установить его в низкий уровень* (0 вольт)
LOW PORTA.0 *' Сделать разряд 0 PORTA выходом и установить на низкий уровень (0 вольт)*

```
led VAR PORTB.0 'Определим переменную LED как pin0 PORTB
LOW led 'Сделать LED выходом и установить ее в низкий уровень (0 вольт)
```

Однако, если контакт уже является выходом, то намного более быстрый и более короткий путь (с точки зрения сгенерированной программы), чтобы установить его в низкое логическое состояние это:

```
PORTB.0 = 0 'Установим pin 0 PORTB на 0
```

Команда **NAP.**

Синтаксис:

NAP *Period*

Эта команда переводит микроконтроллер в режим малого энергопотребления на короткий промежуток времени, равный значению - *Period*. В течение действия команды **NAP**, потребляемая мощность уменьшается до минимума. Значение *Period* можно определить только приблизительно, поскольку выбор времени зависит от сторожевого таймера, который управляется R/C генератором и может изменяться от температуры и от кристалла к кристаллу. Так как команда **NAP** использует сторожевой таймер, то выбор времени не зависит от частоты генератора.

Зависимость времени задержки от значения параметра *Period*

Таблица 16

Период	Задержка (прибл.)
0	18 миллисекунд
1	36 миллисекунд
2	72 миллисекунд
3	144 миллисекунд
4	288 миллисекунд
5	576 миллисекунд
6	1.152 секунд
7	2.304 секунд

Пример:

```
NAP 7 ‘ переводим в спящий режим на 2.3 секунды
```

Команда **ON INTERRUPT.**

Синтаксис:

ON INTERRUPT GOTO *Label*

Команда **ON INTERRUPT** позволяет производить обработку прерываний микроконтроллером. Есть 2 способа обработки прерываний в компиляторе PicBasicPro. В первом случае должна быть написана подпрограмма на ассемблере, которая будет обрабатывать возникшее прерывание. Этот способ работы с прерываниями можно считать самым быстрым и занимающим меньше места в памяти программ микроконтроллера. Второй способ это, когда прерывание обрабатывается средствами языка PicBasicPro. При втором способе подпрограмма обработки прерывания пишется на языке PicBasicPro и заканчивается командой **RESUME**.

Когда происходит прерывание, микроконтроллер отмечает его у себя в памяти и заканчивает выполнение текущей команды. После окончания выполнения этой команды программа переходит на метку *Label* и здесь уже начинает выполнять подпрограмму, которая для этого предназначена. Как только подпрограмма обработки прерывания будет выполнена, команда **RESUME** возвращает программу назад туда, откуда она была вызвана. В PicBasicPro имеются две команды, это **DISABLE** и **ENABLE**, которые позволяют некоторым секциям программы выполняться без возможности быть прерванными. Используя команду **DISABLE**, Вы тем самым запрещаете программе реагировать на прерывания. Команда же **ENABLE**, наоборот, разрешает программе (после того, как была использована команда **DISABLE**) обрабатывать возникающие прерывания. Самое лучшее место, где правильно поместить команду **DISABLE**, так это непосредственно перед подпрограммой обработки прерывания.

Необходимо отметить, что программа не перейдет на подпрограмму обработки прерывания, пока не будет завершена исполняемая в настоящий момент команда. Например, если исполняемой в настоящий момент является команда **PAUSE** или **SERIN**, то это может занять довольно значительное время прежде, чем начнется обработка прерывания. Поэтому в таких случаях долго исполняемые команды рекомендуется делить на части. Например, вместо команды **PAUSE** 300, можно записать – **PAUSE** 100: **PAUSE** 100: **PAUSE** 100.

Если же Ваша задача очень критична ко времени реакции обработки прерывания, то для этих целей должна быть написана подпрограмма на ассемблере.

Пример:

```
ON INTERRUPT GOTO myint 'При возникновении прерывания перейти на
метку - myint
INTCON = %10010000 'Включить прерывание по выводу RB0
. . .
DISABLE          ' Отключить обработку прерываний
```

```
myint: led = 1 ' Включить LED при возникновении прерывания
RESUME       ' Вернуться в основную программу
ENABLE       ' Включить обработку прерываний
```

Чтобы выключать обработку прерываний постоянно (или пока обработка не понадобится), установите регистр INTCON равным $80:

```
INTCON = $80
```

Команда **OUTPUT.**

Синтаксис:

OUTPUT *Pin*

Эта команда установит указанный контакт на выход. Контакт может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0).

Пример:

```
OUTPUT 0          ' Сделать Pin0 выходом
OUTPUT PORTA.0    ' Сделать pin 0 PORTA выходом
```

По-другому, сделать контакт выходом намного более быстрым и более коротким способом (с точки зрения сгенерированной программы) можно так:

```
TRISB.0 = 0 ' Установить pin 0  PORTB как выход
```

Если необходимо сразу все контакты порта установить как выходы, то достаточно записать в регистр TRIS:

```
TRISB = %00000000
```

Команда **OWIN**.

Синтаксис:

OWIN *Pin*, *Mode*,*[Item {,Item}]*

По этой команде на выводе *Pin* микроконтроллер осуществляет прием одного или нескольких разрядов или байтов данных от устройства, работающего по однопроводному интерфейсу. Вывод может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Режим (Mode) определяет, когда посылается импульс сброса - до и / или после приема данных, и размер элементов данных, то есть, бит или байт.

Таблица режимов в команде **OWIN**

Таблица 17

Номер режима	Эффект
0	1 = пошлите импульс сброса перед данными
1	1 = пошлите импульс сброса после данных
2	0 = размер данных - байт, 1 = размер данных - бит

Некоторые примеры режимов:

Режим 0. Нет никакого сброса и данные, размером в байт. Режим 1 – сигнал сброса перед данными и данные, размером в байт. Режим 4 - нет никакого сброса и данные, размером в бит. Элемент - *Item* является одной или несколь*кими* переменными или модификаторами, отделенными запятыми. Допустимыми модификаторами являются **- STR**, который сообщает, что надо считывать данные типа массива переменных, типа байт, и – **SKIP**, который говорит о том, что надо пропустить некоторое количество данных на входе. Модификаторы **SKIP** и **STR** не поддерживаются 12-разрядными PIC-микроконтроллерами из-за недостатка оперативной памяти.

Пример:

```
OWIN PORTC. 0,0,[temperature\2, SKIP 4, count_remain,
count_per_c]
```

Это команда для получения байтов от устройства по одному проводу на выводе PORTC 0 без посылаемого импульса сброса. Полученные 2 байта помещает в переменн*ую* ‘температура’, затем пропускают следующие 4 байта и в заключении помещают последние 2 байта в отдельные переменные.

Команда **OWOUT.**

Синтаксис:

OWOUT *Pin, Mode, [Item{, Item...}]*

По этой команде через вывод Pin посылается один или несколько разрядов или байтов приемному устройству, работающему по однопроводному интерфейсу. Вывод может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Режим определяет, посылают ли сброс прежде и/или после операции передачи данных, а также размер элементов данных (см. таблицу 17).

Некоторые примеры режимов:

Режим 0 Нет никакого сброса и данные размером в байт. Режим 1 – сигнал сброса перед данными, и данные размером в байт. Режим 4 - нет никакого сброса,

и данные размером в бит. Элемент - *Item* является одной или несколькими переменными или модификаторами, отделенными запятыми. Допустимыми модификаторами являются: **STR**, который служит для отправки байтового массива переменных и **REP** для того, чтобы отправить множество одинаковых значений. Модификаторы **REP** и **STR** не поддерживаются 12-разрядными микроконтроллерами из-за недостатка оперативной памяти.

Пример:

```
OWOUT PORTC. 0,1,[$cc,$be]
```

По этой команде посылается импульс сброса приемному устройству через вывод PORTC.0, а затем два байта $cc и $be.

Команда **PAUSE**.

Синтаксис:

PAUSE *Period*

Эта команда приостанавливает работу программы на время – *Period*. Параметр *Period* задается в миллисекундах. Значение периода – 16-разрядное число; таким образом задержки могут быть до 65 535 миллисекунд (немногим более, чем минута). В отличие от других функций задержки (**NAP** и **SLEEP**), **PAUSE** не переводит микроконтроллер в режим малого энергопотребления. Таким образом, в режиме **PAUSE** микроконтроллер потребляет больше мощности, но этот режим также намного более точен. Он работает с той же точностью, что и задающий генератор системы. По умолчанию команда **PAUSE** предполагает частоту задающего генератора 4 МГц. Если используется другой генератор, то необходимо компилятору PicBasicPro сообщить об этом в определении **DEFINE**.

Пример:

```
PAUSE 1000 'Задержка на 1 сек
```

Команда **PAUSEUS.**

Синтаксис:

PAUSEUS *Period*

Эта команда аналогична команде **PAUSE**. Только параметр -*Period* задается в микросекундах. Значение периода – 16-разрядное число; таким образом задержки могут быть до 65 535 микросекунд. Поскольку команда **PAUSEUS** рассчитывается от времени цикла работы микроконтроллера, а время цикла, в свою очередь, зависит от частоты задающего генератора, то задержки меньшие, чем длительность такта, невозможны с оператором **PAUSEUS**. Чтобы получить более короткие задержки, используйте подпрограмму на ассемблере конструкции **ASM..ENDASM**.

Таблица зависимости минимальных задержек от частоты задающего генератора

Таблица 18

OSC МГц	Минимальная задержка мкс
3 (3.58)	20
4	24
8	12
10	8
12	7
16	5
20	3
25*	2
32*	2
33*	2
40**	2

* PIC17Cxxx and PIC18Cxxx only.
** PIC18Cxxx only.

По умолчанию оператор **PAUSEUS** предполагает частоту генератора 4 МГц. Если будет использован другой генератор, то необходимо сообщить об этом компилятору в переопределении **DEFINE**.

Пример:

```
PAUSEUS 1000 ' Задержка в 1 миллисекунду
```

Команда **PEEK.**

Синтаксис:

PEEK *Address*,*Var*

Команда **PEEK** считывает регистр микроконтроллера по указанному адресу - *Address* и сохраняет результат в переменной - *Var*. Специальные особенности PIC микроконтроллеров, типа АЦП и дополнительных портов ввода - вывода могут быть прочитаны команд*ой* **PEEK**. Однако, ко всем регистрам PIC-микроконтроллеров можно и нужно получать доступ, не пользуясь командами **PEEK** и **POKE**. Все регистраторы PIC-микроконтроллеров можно считать как 8-разрядные переменные и использовать их в программах PicBasic Pro, как и люб*ую* переменн*ую*, размером в байт.

Команда **PEEKCODE**.

Синтаксис:

PEEKCODE *Address,Var*

Эта команда считывает значение из памяти программ по указанному адресу – *Address* и результат сохраняет в переменной - *Var* .

Команда **PEEKCODE** может использоваться, в тех случаях, когда есть необходимость считать данные, хранящиеся в памяти программ микроконтроллера.

Пример:

```
PEEKCODE $3ff, OSCCAL     'Установить значение OSCCAL для PIC12C671/12CE673
PEEKCODE $7ff, OSCCAL     'Установить значение OSCCAL для PIC12C672/12CE674
```

Команда **POKE.**

Синтаксис:

POKE Address,Value

Эта команда записывает значение в регистр микроконтроллера по указанному адресу.

```
PEEK PORTB, B0
```

Однако рекомендуется команды **PEEK** и **POKE** не использовать. Намного проще просто записать:

```
B0 = PORTB
```

Команда **POKECODE.**

Синтаксис:
POKECODE *Value{,Value...}*

Эта команда записывает значения в память программ по текущему адресу, когда микроконтроллер запрограммирован.

Команда **POKECODE** может использоваться, для того чтобы генерировать таблицы в памяти программ микроконтроллера.

Команда ассемблера - **Org** может использоваться, для того чтобы установить начальный адрес хранения данных. Если адрес не установлен, данные будут сохранены после последней выполненной команды программы.

Чтобы избежать процесса прерывания выполнения программы, команда **POKECODE** должна находиться в последней строке Вашей программы. В этом месте обычно располагаются команды **STOP** или **END**.

Пример:

```
  POKECODE 10, 20, 30        ‘Сохранить значения 10, 20, и 30 в памяти
‘ программ
Generates:
retlw 10
retlw 20
retlw 30
@     org   7ffh  ‘ Установить стартовый адрес программы $7ff
      POKECODE $94       ‘Установить значения OSCCAL для
PIC12C672/12CE674
Generates:
org   7ffh
retlw       94h
```

Команда **POT.**

Синтаксис:
POT *Pin,Scale,Var*

По этой команде встроенный конденсатор, заряженный до напряжения питания, подключается к указанному выводу - *Pin* и разряжается через измеряемый потенциометр (см. рис. 36). Вывод этот может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Величина *с*опротивления определяет время разряда конденсатора, которое микроконтроллер измеряет и которое является мерой измеряемого сопротивления (обычно от 5 кОм до 50 кОм). Параметр – *Scale* (масштаб) используется для корректировки RC-констант. Для больших RC-констант масштаб должен быть установлен в минимальное значение (1). Для малых RC констант масштаб должен быть установлен в его максимальное значение (255). Если масштаб установлен правильно, то для минимального сопротивления величина переменной *Var* должна быть около нуля и близко к 255 при максимальных сопротивлениях.

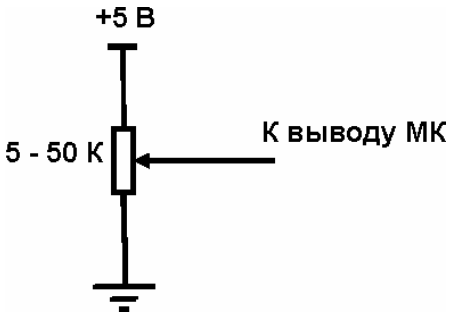


Рис. 36. Схема подключения потенциометра к микроконтроллру.

К сожалению, параметр *Scale* должен быть определен экспериментально. Чтобы сделать это, установите используемый в Вашей схеме потенциометр на максимальное значение сопротивления. Установити в тексте программы значение масштаба – 127 и прочитайте значение потенциала. Затем, корректируя каждый раз, значение масштаба добейтесь того, чтобы командой **POT** было возвращено значение - 254. Если будет равно 255, уменьшите масштаб. Если 253 или ниже, увеличьте масштаб.

Можно использовать следующий текст программы, чтобы автоматизировать этот процесс.

```
B0 Var Byte
scale Var Byte
For scale = 1 To 255
POT 0, scale, B0
If (B0 > 253) Then calibrated
Next scale

Serout 2,0,[" Increase R or C.", 10,13]
Stop
```

```
calibrated:
Serout 2,0,[" Scale= ",# scale, 10,13]
```

Команда **PULSIN**.

Синтаксис:
PULSIN *Pin*,*State*,*Var*

Эта команда измеряет длительность импульса на выводе - *Pin*. Если значение – *State* равно нулю, то измеряется длительность импульса, представляющего собой переход из 1 в 0 и обратно. Если же значение - *State* равно единице, то измеряется импульс перехода из состояния 0 в 1 и обратно. Измеренная величина помещается в переменную - *Var*. Если импульс, начавшись, не кончается, или длительность импульса слишком большая, значение переменной устанавливается на нуль. Если используется 8-битовая переменная, то сохраняется только младший байт 16-разрядного измерения. Вывод может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Разрешающая способность команды **PULSIN** зависит на частоты задающего генератора. Если используется генератор на 4 МГц, то минимальная длительность импульса, которая может быть измерена, это 10 мкс. Если же используется генератор на 20 МГц, то минимальная ширина импульса может быть 2 мкс. Определение значения OSC не имеет никакого эффекта для команды **PULSIN**. Разрешающая способность всегда определяется фактической скоростью генератора.

Пример:

'Измерить длительность положительного импульса на выводе Pin4 и сохранить результат в переменной W3
```
PULSIN PORTB.4,1,W3
```

Команда **PULSOUT**.

Синтаксис:

PULSOUT *Pin*,*Period*

Эта команда генерирует импульс на выводе - *Pin* указанной длительности - *Period*. Импульс создается при двухкратном изменением логического уровня вывода; начальное состояние контакта определяет полярность импульса. Контакт автоматически становится выходом. Вывод может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Разрешающая способность команды **PULSOUT** зависит на частоты задающего генератора. Если будет использоваться генератор на 4 МГц, то минимальная длительность сгенерированного импульса будет 10 мкс. При частоте 20 МГц, минимальная длительность – 2 мкс. Определение значения OSC не

оказывает влияния на **PULSOUT**. Разрешающая способность всегда изменяется с фактической частотой генератора.

Пример:

```
' Пошлите импульс длительностью 1мсек ( 4 МГц) через вывод Pin5
PULSOUT PORTB.5,100
```

Команда **PWM.**

Синтаксис:

```
PWM Pin,Duty,Cycle
```

По этой команде генерируются модулированные по ширине импульсы на выводе - `Pin`. Генерируемый сигнал состоит из циклов. Каждый цикл PWM состоит из 256 тактов. Параметр -`Duty` определяет процент заполнения импульсами одного цикла, то есть 0 это 0 % а 255 это 100 %. Параметр - `Cycle` характеризует количество циклов в одной команде. Вывод - `Pin` может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Время одного цикла команды **PWM** зависит от частоты задающего генератора. Если используется генератор на 4 МГц, то продолжительность цикла - 5 мс. Если же используется генератор на 20 МГц, то - 1 мс. Определение значения OSC не оказывает влияния на **PWM**. Время цикла всегда изменяется в зависимости от фактической частоты задающего генератора. Вывод, который определен в команде, автоматически становится выходом. После завершения команды его состояние возвращается в исходное. Для улучшения формы выходного PWM сигнала желательно использовать фильтр, составленный из R и C элементов. На рисунке 37 представлена возможная схема такого фильтра.

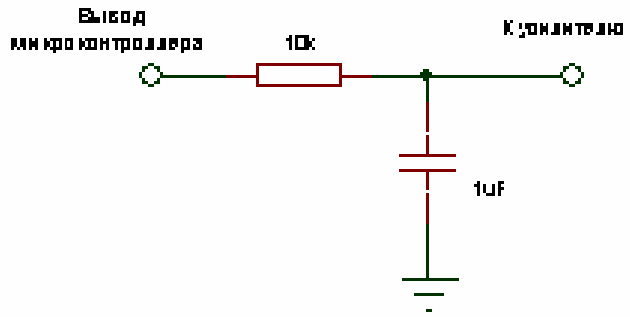


Рис. 37. Схема фильтра предназначенного для выделения гармонической составляющей сигнала создаваемого командой **PWM**.

Пример:

```
PWM PORTB.7,127,100 ' Сгенерировать на выводе Pin7 режим 50%-ного заполнения сигнала PWM в 100 циклов
```

Команда **RANDOM**.

Синтаксис:
RANDOM Var

Эта команда генерирует псевдослучайное число и присваивает его переменной - Var.

Переменная величина должна быть 16-разрядной. Переменные типа массива с переменным индексом не могут использоваться в команде **RANDOM**, хотя переменные типа массива с индексом типа константа разрешены.

Команда **RANDOM** использует алгоритм, в результате работы которого генерируется различные значения от минимальных до максимальных. Максимальное значение может быть 65535. И только ноль не генерируется.

Пример:

```
RANDOM W4  'Сгенерировать случайное число и записать его в переменную W4
```

Команда **RCTIME**.

Синтаксис:
RCTIME Pin,State,Var

Команда **RCTIME** измеряет время, пока вывод - Pin остается в указанном состоянии (State) и измеренное значение присваивает переменной - Var . Это похоже на половину функции - **PULSIN**. Вывод может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Команда **RCTIME** может использоваться для того, чтобы считывать с измеряемого потенциометра (или другого устройства, имеющего сопротивление) время спада напряжения на подключенной емкости. Разрешающая способность **RCTIME** зависит на частоты генератора. Если используется генератор на 4 МГц, время нахождения в указанном состоянии измеряется с точностью до 10 мкс. Если же будет использоваться генератор на 20 МГц, то дискретность измерения станет равной 2 мкс. Определение значения OSC командой-переопределением **DEFINE** не оказывает влияния на команду **RCTIME**. Разрешающая способность измерения всегда изменяется в зависимости от фактической частоты генератора. Если на выводе *Pin* логическое состояние остается неизменным, значение переменной возвращается в 0.

Пример:

```
LOW PORTB.3        ' Для начала разрядим потенциал на контакте PORTB.3
PAUSE 10           ' Выдержим паузу в 10мсек
RCTIME PORTB.3,0,W0  ' считаем потенциометр на Pin3
```

Команда **READ**.

Синтаксис:
READ *Address*,*Var*

Эта команда считывает значения из памяти EEPROM микроконтроллера по указанному адресу - *Address* и сохраняет результат в переменной - *Var*. Эта команда может использоваться только с PIC-микроконтроллерами, которые имеют в своем составе область данных EEPROM, типа PIC16F84, PIC16C84 и ряда PIC16F87x. И эта команда не будет работать при попытке считать данные из внешней EEPROM, подключенной к микроконтроллеру через последовательный порт I^2C как 12CE67x и 16CE62x. Для этого используйте команду **I2CREAD**.

Пример:

```
READ 5,B2 ' Считать значение из EEPROM по адресу 5 и результат поместить в B2
```

Команда **READCODE**.

Синтаксис:
READCODE *Address*,*Var*

Эта команда считывает значения из памяти программ микроконтроллера по указанному адресу - *Address* и результат сохраняет в переменной - *Var*.

Некоторые микроконтроллеры PIC16Fxxx и PIC18Xxxx разрешают программе считывать код программы во время ее выполнения. Это может быть полезным для дополнительного сохранения данных или осуществлять проверку корректности выполняемой программы.

Пример:

```
READCODE 100,W  ' Считать значения кода программы по адресу 100 и результат сохранить в W
```

Команда **REPEAT...UNTIL**.

Синтаксис:

```
REPEAT
  Statements...
UNTIL Condition
```

Оператор **REPEAT...UNTIL** позволяет группе команд выполняться до тех пор, пока указанное условие неверно. Само условие помещается после слова **UNTIL**. Например,

```
i = 0
REPEAT
   PORTB.0[i] = 0
   i = i + 1
UNTIL i > 7
```

Команда **RESUME**.

Синтаксис:
RESUME {*Label*}

Вернутся в то место в программе, откуда она была выведена для обработки прерывания. Команда **RESUME** подобна команде **RETURN**, но записывается она в конце подпрограммы обработки прерывания PicBasicPro. Если в операторе будет использоваться дополнительный указатель - *Label*, то выполнение программы продолжится с этой метки вместо того, чтобы возвращаться в то место, где было совершено прерывание. В этом случае, любые другие адреса возврата в стеке больше не будут доступны.

Команда **RETURN**.

Команда осуществляет возврат из подпрограммы. По этой команде возобновляется выполнение утверждений, стоящих после команды **GOSUB**, которая вызвала переход на подпрограмму.

Пример:

```
GOSUB sub1                  ' Перейти на подпрограмму по метке sub1
...
sub1:
 SEROUT 0,N2400,["Lunch"]   ' Пошлите слово "Lunch" через последовательный порт Pin0

 RETURN                     ' Возвратитесь к основной программе после Gosub
```

Команда **REVERSE**.

Синтаксис:
REVERSE *Pin*

Эта команда, как бы, переворачивает функцию вывода микроконтроллера. Если контакт являлся до этого входом, то после применения команды он становится выходом. И наоборот, если контакт являлся выходом, то он становится входом. Параметр *Pin* может быть константой, значения которой лежат в пределах от 0 и

до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0).

Пример:

```
OUTPUT  4       ' Сделаем Pin4 выходом
REVERSE  4      ' Изменим функцию Pin4 на вход
```

Команда **SERIN**.

Синтаксис:

SERIN *Pin*,*Mode*,{*Timeout*,*Label*,}{[*Qual...*],}{*Item...*}

По этой команде осуществляется прием одного или нескольких значений данных на выводе *Pin* в стандартном асинхронном формате или как говорят по протоколу I2C; при этом (по умолчанию) используется формат: 8 информационных разрядов без проверки на четность и один стоповый бит (8N1). Команда **SERIN** подобна команде **SERIN** от BS1 (BasicStamp 1) с добавлением параметра времени ожидания - *Timeout*. Вывод - *Pin* автоматически становится входом. Вывод может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Названия режимов (например, T2400) определяются в файле MODEDEFS.BAS. Чтобы использовать его, надо в начале программы PicBasicPro добавить строку:

INCLUDE "modedefs.bas"

Номера режима могут использоваться без включения этого файла. Для этого воспользуйтесь таблицей:

Таблица 19

Режим	No. режима	Скорость в бодах	Состояние
T2400	0	2400	Прямое
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	Инверсное
N1200	5	1200	
N9600	6	9600	
N300	7	300	

В команду могут быть включены дополнительные параметры, такие как *Timeout* и *Label*. Параметр *Timeout* или время ожидания разрешает

программе продолжаться с метки *Label*, если значение не было получено в интервале времени – *Timeout*. Время ожидания измеряется в миллисекундах. Списку значений, которые будут получены, может предшествовать один или несколько спецификаторов, заключенных в скобки. По команде **SERIN** эти байты должны быть получены в указанном порядке перед получением элементов данных. Если какой-нибудь полученный байт не соответствует следующему байту в последовательности спецификатора, процесс квалификации начинается сначала. Спецификатором может быть константа, числовая или строковая, или переменная. Каждый знак строковой переменной обрабатывается как индивидуальный спецификатор. Как только спецификатор получен, команда **SERIN** начинает сохранять данные в переменные, связанные с каждым элементом. Если перед переменной есть символ #, команда **SERIN** преобразовывает десятичное значение в ASCII код и сохраняет результат в этой переменной.

Команда **SERIN** по умолчанию рассчитана на работу с генератором в 4 МГц. Чтобы достичь другой скорости передачи в бодах с другими генераторами, необходимо использовать соответствующую директиву **DEFINE**.

Любые нецифровые значения полученные до первой цифры десятичного значения игнорируются и не запоминаются. Нецифровой символ, который следует за десятичн*ым* значением, также будет отвергнут.

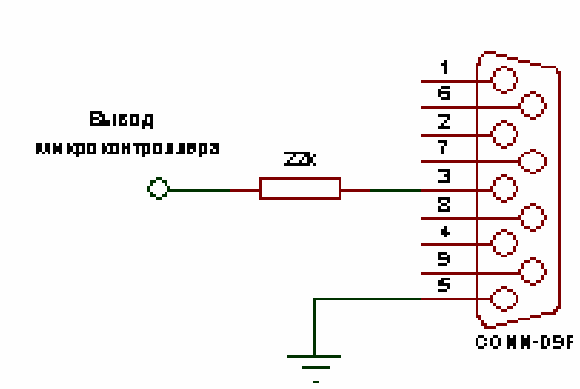


Рис. 38. Схема подключения микроконтроллера к COM порту для команды **SERIN** без использования буферных элементов.

Несмотря на то, что сейчас многие преобразователи уровня интерфейса RS-232 недефицитны и недороги, превосходные параметры портов ввода - вывода PIC-микроконтроллеров позволяют большинству приложений работать непосредственно без буферных элементов.

Пример:

```
' Ждем до знака, полученного по последовательной шине, на Pin1 и поместим следующий знак в B0
SERIN 1,N2400,["A"],B0
```

Команда **SELECT...CASE**.

Синтаксис:

```
SELECT CASE var
   CASE expr1 {, expr...}
      statements
   CASE expr2 {, expr...}
      statements
   {CASE ELSE statements}
END SELECT
```

Этот оператор выбора, в некоторых случаях, являются более легкими в использовании, чем многократный **IF.. THEN**. Он используется для того, чтобы сравнить переменную с различными значениями или диапазоном значений и принять меры на основании этих сравнений.

Переменная, используемая во всех сравнениях, определяется в утверждении - **SELECT CASE**. Каждое **CASE** сопровождается утверждениями, которые выполняются, если **CASE** верно. Вспомогательное **IS** может использоваться во всех случаях сравнения, кроме равенства. Если ни один из **CASE** не верен, то выполняется дополнительное утверждение **CASE ELSE**. Выражение **SELECT END** закрывает выполнение команды **SELECT CASE**.

Пример:

```
SELECT CASE x
 CASE 1
  y = 10
 CASE 2, 3
  y = 20
 CASE IS > 5
  y = 100
 CASE ELSE
  y = 0
END SELECT
```

Команда **SERIN2**.

Синтаксис:

```
SERIN2 DataPin{\FlowPin},Mode,{ParityLabel,}
{Timeout,Label,}[Item...]
```

По этой команде принимается один или несколько элементов данных в стандартном асинхронном формате.

Команда **SERIN2** подобна команде **Serin** в BS2 (BasicStamp 2). Вывод - *DataPin* автоматически устанавливается в состояние входа, а дополнительный вывод - *FlowPin* автоматически становится выходом.

Выводы *DataPin* и *FlowPin* могут быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0).

Дополнительный вывод управления потоком данных - *FlowPin*, может быть включен для того, чтобы препятствовать переполнению приемника данными. Если он используется, то вывод *FlowPin* автоматически устанавливается в состояние, разрешающее передачу символ*ов*. Это состояние определено полярностью данных, указанных в параметре *Mode* (Режим). Параметр *Mode* используется для того, чтобы определить скорость в бодах и операционные параметры последовательной передачи данных. Младшие 13 разрядов осуществляют выбор скорости в бодах. Разряд 13 определяет, есть проверка на четность или ее нет. Бит 14 определяет формат данных, то есть, если 14 бит равен 0, то данные передаются в прямом форме, а если 14 разряд равен 1 – в инверсном; 15 разряд не используется.

Этот режим может использоваться, для того чтобы не устанавливать специальные драйверы RS-232.

Разрядное значение скорости в бодах определяет время передачи 1 бита в микросекундах + 20. Чтобы найти значение для данной скорости в бодах, используйте уравнение:

(1000000 / baud) – 20

Например:

при скорости 600 бод получаем 1000000/600 = 1666,6 - 20 = 1646

Некоторые из стандартных скоростей в бодах перечислены в следующей таблице.

Таблица 20

Скорость в бодах	**Разряды 0 - 12**
300	3313
600	1646
1200	813
2400	396
4800	188
9600	84
19200	32

Приведем примеры некоторых режимов: Режим = 84 (9600 бод, нет проверки на четность, формат данных - прямой), Режим = 16780 (2400 бод, нет проверки на четность, формат данных - инверсный), Режим = 27889 (300 бод, проверка на

четность, инверсный). Если параметр *ParityLabel* будет включен в команду, то программа перейдет на эту метку, как только будет получено нарушение четности. Естественно, что этот параметр можно использовать только тогда, когда выбрана проверка на четность (разряд 13 равен 1). Дополнительное время ожидания -*Timeout* и метка -*Label* могут быть включены в команду для того, чтобы программа продолжилась с этой метки, если в течении указанного промежутка времени не будет получен ни один знак. Время ожидания измеряется в миллисекундах. Команды переопределения **DEFINE** позволяют организовать обмен данными с различным количеством информационных разрядов. Количество информационных разрядов может варьироваться от 4 до 8 (по умолчанию -8, если не было использовано соответствующего **DEFINE**). Если заблокирована проверка на четность (по умолчанию), то можно использовать следующие **DEFINE**:

```
DEFINE SER2_BITS 4 'режим из 4 информационных разрядов
DEFINE SER2_BITS 5 'режим из 5 информационных разрядов
DEFINE SER2_BITS 6 'режим из 6 информационных разрядов
DEFINE SER2_BITS 7 'режим из 7 информационных разрядов
DEFINE SER2_BITS 8 'режим из 4 информационных разрядов (по умолчанию)
```

Если же проверка на четность разрешена то:

```
DEFINE SER2_BITS 5 'режим из 4 информационных разрядов
DEFINE SER2_BITS 6 'режим из 5 информационных разрядов
DEFINE SER2_BITS 7 'режим из 6 информационных разрядов
DEFINE SER2_BITS 8 'режим из 7 информационных разрядов (по умолчанию)
DEFINE SER2_BITS 9 'режим из 8 информационных разрядов
```

Команда **SERIN2** поддерживает различные модификаторы данных и в различных сочетаниях**,** в пределах одной команды **SERIN2** для того, чтобы обеспечить разнообразные способы ввода информации.

Таблица 21

Модификатор	Операция
BIN{1..16}	Принимать двоичные значения
DEC{1..5}	Принимать десятичные значения
HEX{1..4}	Принимать шестнадцатеричные значения
SKIP n	Пропустить получение n знаков
STR ArrayVar\n{\c}	Получить строку n символов, заканчивающуюся символом c

WAIT ( )	Ждите последовательность символов
WAITSTR ArrayVar{\n}	Ждите символьной строки

1) Значения, которым предшествует модификатор **BIN**, будут сохранены в переменную как ASCII код двоичного значения. Например, если записано **BIN,** B0 и будет получено значение A1000, то в переменную B0 будет записано - 8.

2) Значения, которым предшествует модификатор **DEC,** будут сохранены в переменную как ASCII код десятичного значения. Например, если записано **DEC,** B0 и будет получено значение A123, то в переменную B0 будет записано - 123.

3) Значения, которым предшествует модификатор **HEX,** будут сохранены в переменную как ASCII код шестнадцатеричного значения. Например, если записано **HEX,** B0 и будет получено значение как AFE, то в переменную B0 будет записано - 254.

4) Если использован модификатор **SKIP**, то во время приема знаков будут пропущены указанное количество. Например, **SKIP** 4 пропустит 4 бита.

5) Если будет использован модификатор **STR**, который сопровождает массив переменных типа байтовых, то будет получено некоторое количество строковых переменных, заканчивающихся определенным символом. Количество принимаемых знаков определяется значением – n и знаком, стоящим в конце последовательности – c.

6) Получаемым значениям может предшествовать список элементов данных, которые будут получены. Этот список должен быть заключен между круглыми скобками после модификатора - **WAIT**. С этим модификатором по команде **SERIN2** должны быть получены значения в точном порядке, как указано в скобках. Если какой-нибудь полученный байт не соответствует следующему байту в последовательности спецификатора, процесс квалификации начнется сначала (то есть следующий полученный байт - по сравнению с первым элементом в списке спецификатора). Спецификатор может быть константой, переменной или строковой константой. Каждый знак строковой переменной обрабатывается как индивидуальный спецификатор.

7)Модификатор **WAITSTR** может использоваться, как и приведенный выше модификатор **WAIT**, для того, чтобы вынудить команду **SERIN2** дождаться определенного количества знаков строковых переменных перед продолжением.

Как только любой из модификаторов **WAIT** или **WAITSTR** будут выполнены, команда **SERIN2** начинает сохранять данные в переменные, связанные с каждым элементом. Если используется одно имя переменной, то ASCII значение полученного байта сохраняется в этой переменной. Если переменной предшествует **BIN**, **DEC** или **HEX**, то команда **SERIN2** преобразовывает двоичное, десятичное или шестнадцатеричное значение в ASCII код и сохраняет результат в этой переменной. Все нецифровые значения, полученные до первой цифры десятичного значения игнорируются и теряются. Нецифровое значение, которым заканчиваются посылаемые значения, также игнорируется.

Модификаторы **BIN**, **DEC** и **HEX** могут сопровождаться номером. Обычно, эти модификаторы получают столько цифр, сколько их попадает на вход. Однако если номер будет следовать за модификатором, то команда **SERIN2** будет всегда получать это количество цифр, пропуская дополнительные цифры по мере необходимости. Команда **SERIN2** предполагает по умолчанию генератор в 4 МГц. Чтобы поддерживать надлежащий выбор скорости в бодах с другими значениями генератора, убедитесь в том, что соответствующий **DEFINE** присутствует и устанавливает новое значение частоты генератора. В то время, как однокристальные преобразователи уровня RS-232 не дефицитны и недороги, но благодаря превосходным характеристикам портов ввода - вывода PIC-микроконтроллера, большинство приложений не требует преобразователей уровня.

Команда **SERIN2** не поддерживается 12-разрядными PIC-микроконтроллерами из-за недостатка оперативной памяти и ограничения стека.

Примеры:

```
'Ждите до получения знака  “A” на выводе Pin1 и поместите следующий знак в переменную B0
SERIN2 1,16780,[WAIT(“A”),B0]
'Пропустите 2 символа и сохраните затем 4 десятичных цифры
SERIN2 PORTA.1,84,[SKIP 2,DEC4 B0]
```

Команда **SEROUT**.

Синтаксис:

SEROUT *Pin*,*Mode*,[*Item* {,*Item...*}]

По этой команде посылается один или более элементов, в стандартном асинхронном формате, используя 8 информационных разрядов, без проверки на четность и один стоповый бит (8N1). Команда **SEROUT** подобна команде Serout в BS1. Вывод *Pin* автоматически устанавливается как выход. Контакт может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Названия режимов (например, T2400), определяются в файле MODEDEFS.BAS. Чтобы использовать их, добавьте следующую строку в начале программы:

INCLUDE "modedefs.bas"

Файлы BS1DEFS.BAS и BS2DEFS.BAS уже включают в себя MODEDEFS.BAS. Не включайте их снова, если один из этих файлов уже подключен. Номера режимов могут использоваться и без включения этого файла.

Таблица 22.

Режим	Номер режима	Скорость в бодах	Состояние

T2400	0	2400	Прямое
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	Инверсное
N1200	5	1200	
N9600	6	9600	
N300	7	300	
OT2400	8	2400	Открытое прямое*
OT1200	9	1200	
OT9600	10	9600	
OT300	11	300	
ON2400	12	2400	Открытое инверсное*
ON1200	13	1200	
ON9600	14	9600	
ON300	15	300	

* Открытые режимы не поддерживаются 12 разрядными микроконтроллерами

Команда **SEROUT** поддерживает три различных типа данных в любом сочетании в пределах одной команды **SEROUT**.

1) Строковые константы, которые выводятся как строка символов.

2) Числов*ые* значени*я* (переменная или константа), которые посылают соответствующий ASCII код. Чаще всего это 13 - перевод каретки, и 10 - перевод строки.

3) Числовое значение, которому предшествует знак (#), посылает ASCII код его десятичного значения. Например, если W0 = 123, то # W0 (или # 123) пошлет '1', '2', '3'.

Команда **SEROUT** по умолчанию рассчитана на работу с генератором 4 МГц. Чтобы достичь другой скорости передачи в бодах с другими генераторами, необходимо использовать соответствующую директиву **DEFINE**, устанавливающую другое значение частоты генератора. В некоторых случаях передача команд или знаков происходит слишком быстро для приемного устройства. Соответствующая директива **DEFINE** добавляет несколько тактов в последовательную передачу данных. Это увеличивает промежуток времени между передаваемыми знаками. Этот **DEFINE** позволяет осуществлять задержку от 1 - 65 535 микросекунд (0.001 к 65.535 миллисекунд) между каждым

переданным знаком. Например, чтобы вставить задержку в 1 миллисекунду между передачей каждого знака, надо в начале программы записать строку:

```
DEFINE CHAR_PACING 1000
```

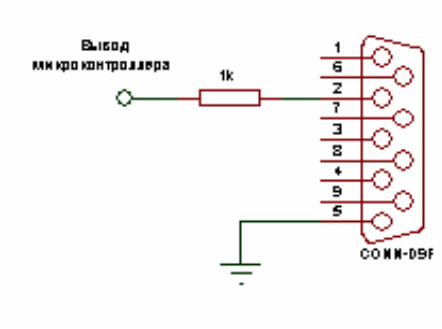


Рис. 39. Схема подключения микроконтроллера к COM порту для команды **SEROUT** без использования буферных элементов.

Несмотря на то, что сейчас многие преобразователи уровня интерфейса RS-232 не дефицитны и недороги, но за счет превосходных показателей параметров портов ввода - вывода PIC микроконтроллеров позволяют большинству приложений работать непосредственно без конвертеров уровня.

Пример:

```
' Посылаем ASCII значение переменной B0 через вывод Pin0 и заканчим знаком перевода строки

SEROUT 0,N2400,[#B0,10]
```

Команда **SEROUT2**.

Синтаксис:
SEROUT2 *DataPin*{*FlowPin*},*Mode*,{*Pace*,}
{*Timeout*,*Label*,}[*Item...*]

По этой команде посылается один или несколько элементов данных в стандартном асинхронном формате. Команда **SEROUT2** подобна команде Serout в BS2. Вывод *DataPin* автоматически становится выходом, а дополнительный вывод *FlowPin* автоматически делается входом. Контакты *DataPin* и *FlowPin* могут быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Дополнительное управление потоком данных может осуществляться с помощью вывода *FlowPin*. С помощью сигналов на этом выводе можно препятствовать данным переполнять приемник. Если вы его будете использовать, то последовательные данные не будут посылаться, пока вывод *FlowPin* не будет находиться в нужном состоянии. Это

состояние определяется полярностью данных, которые зависят от выбранного режима. Дополнительное время ожидания *-Timeout* и метка -*Label* могут быть включены в команду для того чтобы разрешить программе продолжиться с этой метки, если состояние вывода *FlowPin* не изменяется на разрешенное состояние в пределах этого определенного времени. Время ожидания измеряется в миллисекундах.

В некоторых случаях, передача команд или знаков происходит слишком быстро для приемного устройства. Соответствующая директива **DEFINE** добавляет несколько тактов в последовательную передачу данных. Это добавляет дополнительное время между знаками, когда они передаются. Этот **DEFINE** позволяет осуществлять задержку от 1 - 65 535 микросекунд (0.001 к 65.535 миллисекунд) между каждым переданным знаком. Параметр -*Mode* используется для того, чтобы определить скорость в бодах и операционные параметры последовательной передачи данных. Младшие 13 разрядов осуществляют выбор скорости в бодах. Разряд 13 определяет, есть проверка на четность или нет никакой проверки. Бит 14 определяет формат данных то есть если 14 бит равен 0 то данные передаются в прямом формате, а если 14 разряд равен 1 – в инверсном. Разряд 15 определяет, есть ли подключение в настоящий момент или нет.
Разряды значения скорости в бодах определяют время передачи 1 бита в микросекундах - 20. Чтобы найти значение для данной скорости в бодах, используйте уравнение:

(1000000 / baud) – 20

Например:

при скорости 600 бод получаем 1000000/600 = 1666,6 - 20 = 1646

Немного стандартных скоростей в бодах перечисляются{вносятся в список} в следующей таблице.

Таблица 23

Скорость в бодах	**Разряды 0 - 12**
300	3313
600	1646
1200	813
2400	396
4800	188
9600	84
19200	32

Разряд 13 говорит о том, осуществляется ли проверка на четность (разряд 13 = 1) или нет проверки на четность (разряд 13 = 0). Обычно, формат

последовательной передачи данных это - 8N1 (8 информационных разрядов, нет проверки на четность и 1 стоповый бит). Если проверка на четность присутствует, то этот формат можно записать как 7E1 (7 информационных разрядов, проверка на четность и 1 стоповый бит). Разряд 14 говорит о уровне управления потоком данных. Если разряд 14 = 0, то данные посылаются в прямой форме для использования с микросхемами драйверов RS-232. Если разряд 14 = 1, то данные посылаются инвертированными. Этот режим может использоваться, в тех случаях, когда не требуется установка специальных микросхем-драйверов RS-232.

Приведем примеры некоторых режимов: Режим = 84 (9600 бод, нет проверки на четность, формат данных – прямой, всегда управляемый). Режим = 16780 (2400 бод, нет проверки на четность, формат данных – инверсный, управляемый). Режим = 60657 (300 бод, проверка на четность, инверсный, открытый).

Команды переопределения **DEFINE** позволяют организовать обмен данными с различным количеством информационных разрядов. Количество информационных разрядов может варьироваться от 4 до 8 (по умолчанию 8, если не было использовано никакого **DEFINE**). Если заблокирована проверка на четность (по умолчанию) то можно использовать следующие **DEFINE**:

```
DEFINE SER2_BITS 4 'режим из 4 информационных разрядов
DEFINE SER2_BITS 5 'режим из 5 информационных разрядов
DEFINE SER2_BITS 6 'режим из 6 информационных разрядов
DEFINE SER2_BITS 7 'режим из 7 информационных разрядов
DEFINE SER2_BITS 8 'режим из 4 информационных разрядов (по умолчанию)
```

Если же проверка на четность разрешена то:

```
DEFINE SER2_BITS 5 'режим из 4 информационных разрядов
DEFINE SER2_BITS 6 'режим из 5 информационных разрядов
DEFINE SER2_BITS 7 'режим из 6 информационных разрядов
DEFINE SER2_BITS 8 'режим из 7 информационных разрядов (по умолчанию)
DEFINE SER2_BITS 9 'режим из 8 информационных разрядов
```

Команда **SEROUT2** поддерживает много различных модификаторов данных, которые могут быть смешаны и подобраны свободно в пределах одной команды **SERIN2**, для того чтобы обеспечить различный ввод информации.

Таблица 24

Модификаторы	Операция
{I}{S}BIN{1..16}	Посылаются двоичные числа
{I}{S}DEC{1..5}	Посылаются десятичные числа
{I}{S}HEX{1..4}	Посылаются шестнадцатеричные числа
REP c\n	Послать символ с n раз
STR ArrayVar{\n}	Послать n символов,

	заканчивающихся символом с

1) Строковая константа посылается буквально как знак.

2) Числовое значение (переменная или константа) посылает соответствующий ASCII код. Чаще всего это, 13 - перевод каретки, и 10 - перевод строки.

3) Значения, которым предшествует модификатор **BIN**, будут посланы как ASCII представление этого двоичного значения. Например, если B0 = 8, то **BIN** B0 (или **BIN** 8) пошлет "1000".

4) Значения, которым предшествует модификатор **DEC**, будут посланы как ASCII представление этого десятичного значения. Например, если B0 = 123, то **DEC** B0 (или **DEC** 123) пошлет "123".

5) Значения, которым предшествует модификатор **HEX**, будут посланы как ASCII представление этого шестнадцатеричного значения. Например, если B0 = 254, то **HEX** B0 (или **HEX** 254) пошлет "знак спецификации формата".

6) Модификатор **REP**, сопровождаемый знаком и числом повторит этот знак указанное количество раз. Например, **REP** A0" \4 пошлет "0000".

7) Модификатор **STR**, сопровождаемый переменной типа байтового массива и дополнительного числа пошлет строковую переменную. Длина строковой переменной определяется указанным числом.

Модификаторам **BIN**, **DEC** и **HEX** могут предшествовать или сопровождать их несколько дополнительных параметров. Если любому из них будет предшествовать параметр I (для обозначенного), то выходным значениям должны будут предшествовать знаки - "%", "#" или "$", чтобы указать, что следующие затем значения являются двоичными, десятичными или шестнадцатеричными. Если же будет предшествовать параметр **S** (знак «-»), то это значит, что принимаемое значение нужно воспринимать как отрицательное **значение**. Это ухищрение позволяет осуществлять передачу отрицательных чисел. Имейте в виду, что вся математика в PicBasicPro беззнаковая. Однако, такая математика без знака может привести к неверным результатам. Например, возьмите случай B0 = 9 - 10. Результат **DEC** B0 был бы "255". Посылка **SDEC** B0 дала бы "-1".

Модификаторы **BIN**, **DEC** и **HEX** могут также сопровождаться некоторым числом. Обычно, оно сообщает модификатору, какое точно количество цифр необходим послать. Обычно впереди стоящие и ничего не значащие нули не посылаются. Однако, если за модификатором будет стоять число, то команда **SEROUT2** будет посылать это количество цифр, добавляя начальные нули по мере необходимости. Например, **BIN**6 8 послал бы "001000", а **BIN**2 8 пошлет "00". Любые комбинации из модификаторов могут использоваться все вместе. Например, **ISDEC**4 B0. Команда **SEROUT2** по умолчанию предполагает генератор на 4 МГц. Чтобы использовать необходимую скорость в бодах с другими значениями частоты генератора не забудьте своевременно записать соответствующий **DEFINE**, устанавливающий новое значение частоты генератора.

Пример:

' Пошлите со скоростью в 2400 бод через вывод Pin0 ASCII значение переменной B0, сопровождая кодом перевода строки

```
SEROUT2 0,16780,[dec B0,10]
```

' Пошлите со скоростью 9600 бод через вывод PORTA.1 "B0 =" двоичное значение переменной B0

```
SEROUT2 PORTA.1,84,["B0=", ihex4 B0]
```

Команда **SHIFTIN**.

Синтаксис:

SHIFTIN *DataPin*,*ClockPin*,*Mode*,[*Var*{*Bits*}...]

Эта команда используется для связи микроконтроллера с внешними устройствами по интерфейсу SPI или протокола синхронной последовательной передачи данных.

По этой команде данные, получаемые на выводе -*DataPin*, синхронизируются импульсами на выводе – *ClockPin* и сохраняются в переменную *Var*. Выводы *ClockPin* и *DataPin* могут быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Параметр – *\Bits* определяет количество разрядов, которые будут приняты. Если параметр специально не определен в соответствующем **DEFINE**, по умолчанию принимается 8 разрядов в независимости от типа переменной. Названия режимов (например, **MSBPRE**), определяются в файле MODEDEFS.BAS. Чтобы использовать их, добавьте в программе строку:

INCLUDE "modedefs.bas"

Номер режима можно использовать и без включения этого файла. Некоторые режимы не имеют названия. Для режимов 0-3 синхроимпульсы не выдаются, пока не появятся данные. Логическое состояние на этом выводе низкое. Перед появлением данных состояние на выводе *ClocPin* меняется на высокое и затем опять возвращается в низкое. Для режимов 4-7, логический уровень на выводе постоянно высокий, а перед появлением данных он падает до 0 и возвращается в высокое логическое состояние.

Таблица 25

Режим	***№ режима.***	**Операция**
MSBPRE	**0**	Данные передаются, начиная с самого старшего разряда. Данные считываются перед посылкой синхроимпульса. Основное логическое состояние цепи ClocPin - низкое.
LSBPRE	**1**	Данные передаются, начиная с самого

		младшего разряда. Данные считываются перед посылкой синхроимпульса. Основное логическое состояние цепи ClocPin - низкое.
MSBPOST	**2**	Данные передаются, начиная с самого старшего разряда. Данные считываются после появления синхроимпульсов. Основное логическое состояние цепи ClocPin – низкое.
LSBPOST	**3**	Данные передаются, начиная с самого младшего разряда. Данные считываются после появления синхроимпульсов. Основное логическое состояние цепи ClocPin - низкое.
	4	Данные передаются, начиная с самого старшего разряда. Данные считываются перед приходом синхроимпульса. Основное логическое состояние цепи ClocPin - высокое.
	5	Данные передаются, начиная с самого младшего разряда. Данные считываются перед приходом синхроимпульса. Основное логическое состояние цепи ClocPin - высокое.
	6	Данные передаются, начиная с самого старшего разряда. Данные считываются после прихода синхроимпульса. Основное логическое состояние цепи ClocPin - высокое.
	7	Данные передаются, начиная с самого младшего разряда. Данные считываются после прихода синхроимпульса. Основное логическое состояние цепи ClocPin - высокое.

Частота синхроимпульсов приблизительно 50 кГц и зависит от генератора. Минимальная длительность синхроимпульса 2 микросекунды. Определение **DEFINE** позволяет увеличить длительность синхроимпульса до 65.535 миллисекунд, чтобы замедлить тактовую частоту. Минимальная дополнительная

задержка определяется выбором времени **PAUSEUS**. Например, чтобы установить длительность синхроимпульса равной 100 микросекунд:

DEFINE SHIFT_PAUSEUS 100

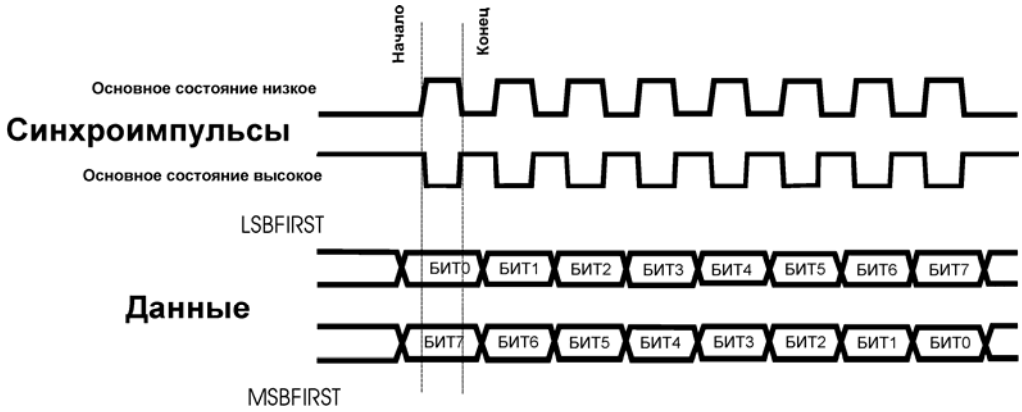


Рис. 40. Временные диаграммы передачи данных по интерфейсу SPI.

Пример:

```
SHIFTIN 0,1,MSBPRE,[B0,B1\4]
```

Команда **SHIFTOUT.**

Синтаксис:
SHIFTOUT *DataPin*,*ClockPin*,*Mode*,[*Var*{*Bits*}...]

Эта команда также как и команда **SHIFTIN** используется для связи микроконтроллера с внешними устройствами по интерфейсу SPI или протокола синхронной последовательной передачи данных.

По этой команде данные, определяемые значением *Var*, выдаются с вывода *DataPin* и синхронизируются импульсами на выводе *ClockPin*. Выводы *ClockPin* и *DataPin* могут быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Параметр – *\Bits* определяет количество разрядов, которые будут переданы. Если это специально не определено в соответствующем **DEFINE**, по умолчанию принимается 8 разрядов в независимости от типа переменной. Названия режимов (например. **MSBPRE**), определяются в файле MODEDEFS.BAS. Чтобы использовать их, добавьте в начале Вашей программы PicBasic Pro строку:

INCLUDE "modedefs.bas".

Номера режимов могут использоваться без включения этого файла. Некоторые режимы не имеют названия. Для Режимов 0-1 пассивное логическое состояние цепи *ClocPin* низкое. При появлении синхроимпульса логическое состояние меняется на высокое и затем вновь возвращается в низкое состояние. Для Режимов 4-5, наоборот, пассивное логическое состояние высокое, появление

синхроимпульса переводит его в низкое и затем возвращается к высокому уровню.

Таблица 26

Режим	Номер режима	Операция
LSBFIRST	**0**	Данные выдаются, начиная с самого младшего разряда. Неактивное состояние цепи Clc - низкое.
MSBFIRST	**1**	Данные выдаются, начиная с самого старшего разряда. Неактивное состояние цепи Clc - низкое.
	4	Данные выдаются, начиная с самого младшего разряда. Неактивное состояние цепи Clc - высокое.
	5	Данные выдаются, начиная с самого старшего разряда. Неактивное состояние цепи Clc - высокое.

Частота синхронизации приблизительно 50 КГц и зависит от частоты задающего генератора. Минимальная длительность импульса синхронизации 2 микросекунды. Команда **DEFINE** позволяет изменить длительность синхроимпульса до 65 535 микросекунд или 65.535 миллисекунд. Например, чтобы сделать длительность синхроимпульса 100 микросекунд введите следующую строку:

```
DEFINE SHIFT_PAUSEUS 100
```

Пример:

```
SHIFTOUT 0,1,MSBFIRST,[B0,B1]
SHIFTOUT PORTA.1, PORTA.2, 1,[wordvar\4]
```

Команда **SLEEP**.

Синтаксис:
SLEEP *Period*

По этой команде микроконтроллер переходит в режим малого энергопотребления на время - *Period* (в секундах). Период - 16 разрядное число. Таким образом, задержки могут быть до 65 535 секунд (т.е. более, чем 18 часов). Команда **SLEEP** использует сторожевой таймер, поэтому время задержки этой команды не зависит от фактической частоты генератора. Минимальная дискретность составляет приблизительно 2.3 секунды и может изменяться в зависимости от типа микроконтроллера и температуры.

Пример:

```
SLEEP 60 'Перейти в режим малого энергопотребления на 1 час
```

Команда **SOUND**.

Синтаксис:
SOUND *Pin*,[*Note*,*Duration*{,*Note*,*Duration*...}]

По этой команде генерируется частота или белый шум на выводе - *Pin*. Указанный контакт автоматически становится выходом. Контакт может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0) или названием вывода (например, PORTA.0). Параметр *Note* задает частоту генерируемого сигнала. При *Note* = 0 – отсутствует генерация. При *Note* от 1 до 127 – определенные частоты, а от 128 до 255 - белый шум. Частоты и белые шумы следуют в порядке возрастания (то есть 1, и 128 - самые низкие частоты, 127 и 255 являются самыми высокими). Если *Note* = 1 генерируется частота в 78.74 Гц, а если *Note* = 127 - частота 10,000 Гц. Продолжительность генерации - *Duration* может меняться от 0 до 255 и измеряется приблизительно в 12 миллисекундах. *Note* и *Duration* не должны быть константами.

Команда **SOUND** выводит на указанном контакте периодический сигнал прямоугольной формы с ТТЛ-уровнями. Благодаря превосходным характеристикам ввода - вывода PIC микроконтроллера, можно непосредственно подключать динамик к контакту через проходной конденсатор (см. рис. 55). Значение емкости конденсатора должно быть определено на основании тех частот, на которых будет работать динамик. Если Вы имеете Piezo-излучатель, то его можно подключать и непосредственно.

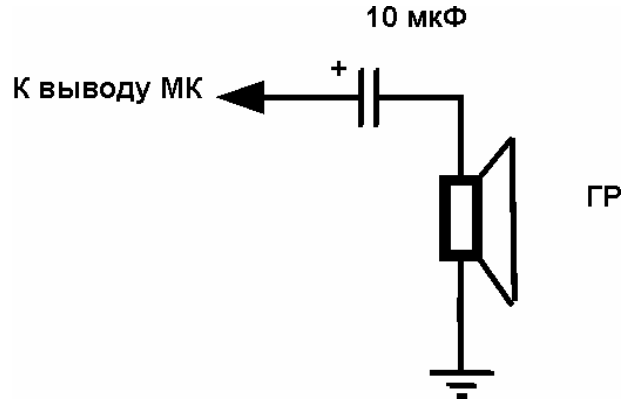


Рис.41. Схема подключения громкоговорителя к микроконтроллеру.

Пример:

```
SOUND PORTB.7,[100,10,50,10] 'Сгенерировать на выводе Pin7 последовательно 2 частоты
```

Команда **STOP**.

Эта команда останавливает выполнение программы. Однако эта команда не переводит микроконтроллер в режим малого энергопотребления.

Пример:

```
STOP 'Остановить выполнение программы
```

Команда **SWAP**.

Синтаксис:

```
SWAP Variable,Variable
```

Обменивает значения между двумя переменными. Обычно, это - утомительный процесс - поменять значения двух переменных. Команда **SWAP**, делает процесс обмена переменными одном заявлением, при этом, не используя никаких промежуточных звеньев. Команда **SWAP** рабртает с переменными типа bit, byte и word. Переменные типа массива с переменным индексом не могут использоваться в команде **SWAP**, хотя переменные типа массива с постоянными индексами допустимы.

Пример:

```
temp = B0          'Старый способ
B0 = B1
B1 = temp
SWAP B0, B1        'Новый способ
```

Команда **TOGGLE**.

Синтаксис:

```
TOGGLE Pin
```

Эта команда инвертирует состояние указанного контакта. Контакт автоматически становится выходом. Контакт может быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0).

Пример:

```
LOW 0          ' Установить Pin0 в низкое состояние
TOGGLE 0       ' Изменить состояние Pin0 на высокое
```

Команда **WHILE...WEND.**

Синтаксис:

```
WHILE Condition
  Statements...
WEND
```

Этот оператор многократно выполняет условие **WHILE**, если оно верно. Когда утверждение больше не верно, выполнение программы продолжается с утверждения, стоящего после **WEND**. Условие может быть любым выражением сравнения.

Пример:

```
i = 1
WHILE i <= 10
  SEROUT 0,N2400,["No:",#i,13,10]
  i = i + 1
WEND
```

Команда **USBINIT**.

Это утверждение должно быть одним из первых в тексте программы, в которой подразумевается использование USB канала связи. Эта инструкция предназначена для микроконтроллеров, которые аппаратно поддерживают USB связь, то есть имеют USB порт. Например, PIC16C745 и 765. Эта поддержка принимает форму трех новых команд PicBasicPro и некоторых модификаций библиотек Microchip для USB. Эта команда инициализирует USB-порт PIC микроконтроллера и будет ждать, пока шина USB не будет сформирована и к ней не будет разрешен доступ. Программы для работы по каналу USB требуют, чтобы было добавлено несколько файлов (которые находятся в подкаталоге USB). Связь по шине USB намного более сложна, чем синхронная (**SHIFTIN** и **SHIFTOUT**) или асинхронная (**SERIN**, **SEROUT**) связи. Для того, чтобы больше узнать о связи по каналу USB, смотрите информацию в Интернет на странице фирмы Microchip.

Команда **USBIN**.

Синтаксис:

```
USBIN Endpoint,Buffer,CountVar,Label
```

По команде **USBIN** микроконтроллер получает любые доступные данные по каналу USB и помещает их в буфер. Буфер должен быть байтовым множеством подходящей длины, чтобы иметь возможность сохранять данные. Параметр *Countvar* содержит число байтов, помещаемых в буфер. Если по каким-то причинам данные не будут доступны, программа перейдет на метку – *Label*. Подкаталог USB содержит измененные библиотеки USB от фирмы Microchip, а так же примеры программ. Программы для работы по каналу USB требуют, чтобы

было добавлено несколько файлов (они находятся в подкаталоге USB). Связь по шине USB намного более сложна, чем синхронная (**SHIFTIN** и **SHIFTOUT**) или асинхронная (**SERIN**, **SEROUT**) связи. Для того чтобы больше узнать о связи по каналу USB, смотрите информацию в Интернет на странице фирмы Microchip.

Команда **USBOUT**.

Синтаксис:

USBOUT *Endpoint,Buffer,Count,Label*

Команда **USBOut** выбирает из буфера – *Buffer* некоторое количество -*Count* байтов и посылает их конечной точке – *Endpoint* USB канала связи. Если по каким то причинам данные не могут быть переданы то выполнение программы продолжится с указанной метки - *Label* . Подкаталог USB содержит измененные библиотеки USB от фирмы Microchip, а так же примеры программ. Программы для работы по каналу USB требуют, чтобы было добавлено несколько файлов (они находятся в подкаталоге USB). Связь по шине USB намного более сложна чем синхронная (**SHIFTIN** и **SHIFTOUT**) и асинхронная (**SERIN**, **SEROUT**) связи. Для того чтобы больше узнать о связи по каналу USB, смотрите информацию в Интернет на странице фирмы Microchip.

Команда **WRITE**.

Синтаксис:

WRITE *Address,Value*

По этой команде записывается новое значение в EEPROM по указанному адресу. Эта команда может использоваться только с PIC микроконтроллерами, которые имеют внутреннюю EEPROM, типа PIC16F84, PIC16C84 из ряда PIC16F87x. Чтобы записать новое значение в EEPROM при программировании, используйте для этого команды EEPROM или DATA. Каждая команда **WRITE** затрачивает на своё выполнение приблизительно 10 миллисекунд.

Если в программе используются прерывания, они должны быть отключены (замаскированы) перед выполнением команды **WRITE**, а затем возвращены (если нужно) после того, как запись произведена. Прерывание, которое может возникнуть во время выполнения команды **WRITE**, может привести к некорректному выполнению этой команды. Понятно, что команда **WRITE** не будет работать с EEPROM, находящейся вне кристалла и работающей по последовательному каналу связи типа I^2C, как 12CE67x и 16CE62x части. Для этого используйте команду **I2CWRITE**.

Пример:

```
WRITE 5,B0 ' Записать значение переменной B0 в EEPROM по адресу 5
```

Команда **WRITECODE**

Синтаксис:

WRITECODE *Address*,*Value*

По этой команде записывается значение –*Value* размером в слово в память программ по адресу - *Address*. Микроконтроллеры типа PIC16F87x позволяют программе читать и записывать в память в процессе выполнения программы. Эта команда позволяет осуществлять самомодифицирующуюся программу. Но делать это надо очень осторожно. Если в программе используются прерывания, то они должны быть временно отключены (замаскированы) командой **DISABLE** перед выполнением **WRITECODE**, и возвращены (если это необходимо) после того, как команда записи будет выполнена. Прерывание, произошедшее во время выполнения **WRITECODE**, может привести к плачевным результатам.

Пример:

```
WRITECODE 100,W ' записать значение W в память по адресу 100
```

Команда **XIN.**

Синтаксис:

XIN *DataPin*,*ZeroPin*,{*Timeout*,*Label*,}[*Var*{,...}]

Эта команда работает по интерфейсу X-10. Она получает данные и сохраняет в памяти переменные этого интерфейса - House Code и Key Code. Устройства, работающие по этому интерфейсу, выпускаются многими производителями и под разными торговыми марками. По этому интерфейсу микроконтроллер должен быть подключен к питающей сети переменного напряжения. Для организации двухсторонней связи (протокол TW-523) интерфейса X-10 требуется команда **XIN**. Это устройство должно содержать схему для подключения к сетевому переменному напряжению, так как вся информация передается по цепям питания. Интерфейс X-10 запатентован, поэтому за его использование необходимо платить.

Для того, чтобы получать данные по интерфейсу X-10, вывод `DataPin` автоматически становится входом. Вывод -`ZeroPin` также автоматически устанавливается в состояние “вход”. Он служит для получения сигнала перехода синусоиды питающего напряжения через нулевую точку. По этому сигналу начинается выборка информации согласно правилам интерфейса X-10. При этом оба вывода должны быть подключены к цепи +5 В через резистор сопротивлением 4.7 кОм. Контакты `ZeroPin` и `DataPin` могут быть константой, значения которой лежат в пределах от 0 и до 15, или переменной, значения которой могут меняться от 0 и до 15 (например, B0), или названием вывода (например, PORTA.0). Параметры - *Timeout* (дополнительное время ожидания) и - *Label* (метка) могут быть включены в команду для того, чтобы программа могла продолжиться, если данные X-10 не будут получены в пределах указанного времени. Время ожидания определено в полупериодах сетевого

питания (при 60 Гц, приблизительно 8.33 миллисекунды). Команда **XIN** обрабатывает данные только тогда, когда будет получен 0 по цепи ZeroPin, то есть, при пересечении нуля синусоидой сетевого питания. Если этого сигнала не будет, команда XIN будет ждать его бесконечно.

Key Code может быть номером определенного модуля X-10 или функцией, которая должна быть выполнена модулем. В нормальной практике сначала посылают команду, определяющую номер модуля X-10. Номер сопровождается командой, определяющей желаемую функцию. Некоторые функции работают на всех модулях сразу, и, следовательно, номер модуля не нужен. Хотелось бы надеяться, что примеры, показанные ниже, разъяснят эти вещи лучше. Номер Key Code (от 0 до 15) соответствует номерам модулей (1-16).

Пример:

```
housekey VAR WORD
'Получить данные по каналу X-10
loop: XIN PORTA.2,PORTA.0,[housekey]

'Вывести данные, полученные по X-10 на ЖК монитор
LCDOUT $fe,1,"House=",#housekey.byte1,
"Key=",#housekey.byte0
GOTO loop ' Вернуться назад
'Примите данные по каналу X-1. Если данных не будет в течении 1 полупериода,
‘ перейдите на метку nodata
XIN PORTA.2,PORTA.0,1,nodata,[housekey]
```

Команда **XOUT.**

Синтаксис:

```
XOUT DataPin,ZeroPin, [HouseCode\KeyCode{\Repeat}{,...}]
```

Команда **XOUT** посылает в формате X-10 значения HouseCode и KeyCode, повторив их несколько раз. Если дополнительное значение -*Repeat* (повторить) опущено в команде, эти значения (по умолчанию) повторяются минимум 2 раза. Повторение обычно резервируется для использования с командами - Bright (Ярче) и – Dim (Темнее). Команда **XOUT** используется для того, чтобы послать информацию управления модулям X-10. Этих модулей выпускается довольно много и под несколькими торговыми марками. Согласно интерфейсу микроконтроллер обязан быть подключенным к сети питания переменного напряжения. Он может работать по протоколу – PL-513, в котором идет только передача информации, или по протоколу - TW-523, по которому осуществляется двухсторонняя связь. Так как интерфейс X-10 запатентован, то за его использование необходимо платить.

По команде **XOUT** вывод DataPin автоматически устанавливается в состояние выхода для того, чтобы отправлять данные по интерфейсу X-10. Вывод ZeroPin автоматически устанавливается в состояние входа для получения

значения 0, которое формируется при переходе синусоиды питающего напряжения через нулевое значение. Эти цепи должны быть подключены к цепи питания +5 В через подтягивающие резисторы сопротивлением 4.7 кОм. Контакты `ZeroPin` и `DataPin` могут быть константами, значения которых лежат в пределах от 0 и до 15, или переменными, значения которых могут меняться от 0 и до 15 (например, B0), или названием выводов (например, PORTA.0)

Команда **XOUT** начинает отправлять данные при поступлении 0 по цепи `ZeroPin`. Если переходов синусоиды питающего напряжения не будет, команда XOUT будет фактически ждать их бесконечно долго.

KeyCode может быть номером определенного модуля X-10 или функцией, которая должна быть выполнена модулем. В нормальной практике, сначала посылают команду, определяющую номер модуля X-10, затем команду, которая определяет желаемую функцию. Некоторые функции работают на всех модулях сразу, и поэтому номер модуля не нужен. Примеры, показанные ниже, надеюсь, разъяснят это лучше. Номера KeyCode (от 0 и до 15) соответствуют номерам модулей 1-16. Имена функций KeyCode (например, unitOn) определены в файле MODEDEFS.BAS. Чтобы использовать их, добавьте в начале программы PicBasic Pro следующую строчку:

ВКЛЮЧИТЕ "modedefs.bas"

в начале программы PicBasic Pro. Файлы BS1DEFS.BAS и BS2DEFS.BAS уже включают файл MODEDEFS.BAS. Поэтому не включайте его снова, если один из этих файлов уже подключен. Значения KeyCode могут использоваться и без включения этого файла.

Таблица 27

KeyCode	KeyCode No.	**Операция**
unitOn	%10010	Включить модуль
unitOff	%11010	Выключить модуль
unitsOff	%11100	Выключить все модули
lightsOn	%10100	Включите все световые модули
lightsOff	%10000	Выключите все световые модули
bright	%10110	Ярче световой модуль
dim	%11110	Темнее световой модуль

Соединение по интерфейсу X-10 требует 4 линий. Выводы микроконтроллера используемые в интерфейсе X-10 – должны быть выводами с открытым коллектором, к которым подключены подтягивающие резисторы приблизительно 4,7 кОм. Ниже приведены таблицы соединений для каждого интерфейса:

Соединение PL 513

Таблица 28

№ провода	Цвет провода	Подключение
1	**Черный**	Выход цепи Zero
2	**Красный**	Общий для цепи Zero
3	**Зеленый**	Общий X-10
4	**Желтый**	Выход X-10

Соединение TW-523

Таблица 29

№ провода	Цвет провода	Подключение
1	**Черный**	Выход цепи Zero
2	**Красный**	Общий
3	**Зеленый**	Выход X-10
4	**Желтый**	Вход X-10

Пример:

```
house VAR BYTE
unit VAR BYTE
INCLUDE "modedefs.bas"
house = 0 ' Установим переменную house в 0
unit = 8 ' Установим переменную unit равную 8

'Включить модуль (unit) 8 в доме (house) 0
XOUT PORTA.1,PORTA.0,[house\unit,house\unitOn]

'Отключить все освещение в доме (house) 0
XOUT PORTA.1,PORTA.0,[house\lightsOff]

'Будем выключать свет на 10 секунд
XOUT PORTA.1,PORTA.0,[house\0]
loop: XOUT PORTA.1,PORTA.0,[house\unitOn]
PAUSE 10000                    'Ждем 10 секунд
XOUT PORTA.1,PORTA.0,[house\unitOff]
PAUSE 10000                    'Ждем 10 секунд
GOTO loop
```

Оператор **DEFINE.**

Некоторые параметры PicBasicPro, такие, как частота задающего генератора, выводы микроконтроллера для подключения ЖК монитора и тому подобные,

являются предопределенными в PicBasicPro. Команда переопределения DEFINE позволяет компилятору PicBasicPRO изменять эти настройки по Вашему усмотрению. Эти определения должны быть все в верхнем регистре.
Примеры переопределений в PicBasicPro:

```
DEFINE ADC_BITS 8 'Количество разрядов в результате АЦП
DEFINE ADC_CLOCK 3 ' Источник синхронизации АЦП (rc = 3)
DEFINE ADC_SAMPLEUS 50 ' Время выборки АЦП преобразования в мкс
DEFINE BUTTON_PAUSE 10 'Противодребезговая задержка срабатывания
‘ кнопки в мс
DEFINE CCP1_REG PORTC ' Определяем порт для первого канала Hpwm
DEFINE CCP1_BIT 2 ' Определяем вывод для первого канала Hpwm
DEFINE CCP2_REG PORTC' Определяем порт для второго канала Hpwmt
DEFINE CCP2_BIT 1 ' Определяем вывод для второго канала Hpwmt
DEFINE CHAR_PACING 1000 ' Задержка между передачей знаков в мкс
DEFINE DEBUG_REG PORTB 'Установить порт для работы с функцией Debug
DEFINE DEBUG_BIT 0 'Установить вывод для работы с функцией Debug
DEFINE DEBUG_BAUD 2400 'Установить скорость передачи в бодах для
‘ Debug
DEFINE DEBUG_MODE 1 'Режим работы функции Debug: 0 = True, 1 = Inverted
DEFINE DEBUG_PACING 1000 ' Задержка между передачей знаков в мкс для
‘ функции Debug
DEFINE DEBUGIN_REG PORTB ' Устанавливаем порт для работы с командой
‘ Debugin
DEFINE DEBUGIN_BIT 0 ' Устанавливаем вывод порта для работы с
‘ командой Debugin
DEFINE DEBUGIN_MODE 1 ' Установка режима работы с командой Debugin:
‘ 0 = прямой, 1 = инверсный
DEFINE HPWM2_TMR 1 ' В команде Hpwm для 2 канала использовать таймер 1
DEFINE HPWM3_TMR 1 ' В команде Hpwm для 3 канала использовать таймер 1
DEFINE HSER_BAUD 2400 ' Установить скорость в бодах для команды Hser
DEFINE HSER_CLROERR 1 ' Автоматически очищать ошибку переполнения
Hser
DEFINE HSER_SPBRG 25 ' Установить непосредственно регистр SPBRG
(обычно установлено HSER_BAUD)
DEFINE HSER_RCSTA 90h ' Установить в регистре приемнике разрешение на
‘ прием
DEFINE HSER_TXSTA 20h ' Установить для регистра передатчика
‘ разрешение на передачу
DEFINE HSER_EVEN 1 ' Использовать проверку на четность
DEFINE HSER_ODD 1 ' Использовать проверку на нечетность
DEFINE HSER2_BAUD 2400 ' Установить скорость в бодах для 2 порта
DEFINE HSER2_CLROERR 1 ' Автоматически очищать ошибку переполнения
```

```
‘ для 2 порта
DEFINE HSER2_SPBRG 25 ' Установить SPBRG для 2 порта непосредственно
‘ (обычно устанавливается HSER_BAUD)
DEFINE HSER2_RCSTA 90h ' Установить регистр 2 приемника на получение
‘ информации
DEFINE HSER2_TXSTA 20h ' Установить регистр 2 передатчика на выдачу
‘ нформации
DEFINE HSER2_EVEN 1 ' Использовать проверку на четность для 2 порта
DEFINE HSER2_ODD 1 ' Использовать проверку на нечетность для 2 порта
DEFINE I2C_HOLD 1 ' пауза в передаче информации по шине I2C когда цепь
‘ синхронизации переведена в низкое логическое состояние
DEFINE I2C_INTERNAL 1 ' Работать с внутренней EEPROM
‘ микроконтроллеров 16CExxx и 12CExxx
DEFINE I2C_SCLOUT 1 ' Установить цепь синхроимпульсов биполярной
‘ вместо открытого коллектора
DEFINE I2C_SLOW 1 ' Для частот >8МГц установить доступ к
‘ стандартным скоростям
DEFINE I2C_SDA PORTA,0 ' Установить вывод данных (касается только12-
‘ разрядных МК)
DEFINE I2C_SCL PORTA,1 'Установить вывод синхронизации (касается
‘ только12-разрядных МК)
DEFINE LCD_DREG PORTA 'Определяем порт данных для связи с ЖК
‘ монитором
DEFINE LCD_DBIT 0 'Порядковый адрес порта для подключения цепи данных
‘ ЖК монитора( 0 или 4)
DEFINE LCD_RSREG PORTA 'Определяем порт к которому подключается цепь
‘ RS ЖК монитора
DEFINE LCD_RSBIT 4 'Определяем вывод к которому подключается цепь RS
‘ ЖК монитора
DEFINE LCD_EREG PORTB 'Определяем порт к которому подключается цепь
‘ ENABLE ЖК монитора
DEFINE LCD_EBIT 3 'Определяем вывод к которому подключается цепь
‘ ENABLE ЖК монитора
DEFINE LCD_RWREG PORTE 'Определяем порт к которому подключается цепь
‘ read/write ЖК монитора
DEFINE LCD_RWBIT 2 'Определяем вывод к которому подключается цепь
‘ read/write ЖК монитора
DEFINE LCD_BITS 4 ' Подключение ЖК монитора осуществляется по 4 или 8
‘ разрядной шине
DEFINE LCD_LINES 2 ' Количество строк используемого ЖК монитора
DEFINE LCD_COMMANDUS 2000 'Время задержки команд в мкс
DEFINE LCD_DATAUS 50 'Время задержки данных в мкс
DEFINE LOADER_USED 1 'Использовать встроенный Bootloader
```

```
DEFINE NO_CLRWDT 1 ' Не добавлять команды CLRWDT
DEFINE PULSIN_MAX 1000 ' Установить максимальное значение счета для
' измерения pulsin
DEFINE OSC 4 ' Установить частоту генератора в МГц: 3(3.58), 4, 8, 10, 12,
' 16, 20, 24, 25, 32, 33, 40
DEFINE OSCCAL_1K 1 ' Установить значение частоты генератора для МК
' PIC12C671/CE673
DEFINE OSCCAL_2K 1 ' Установить значение частоты генератора для МК
' PIC12C672/CE674
DEFINE SER2_BITS 8 ' Установить разрядность данных для команд Serin2 и
' Serout2
DEFINE SER2_ODD 1 ' Установить проверку на нечетность
DEFINE SHIFT_PAUSEUS 50 ' Изменить длительность синхроимпульсов для
' команд Shiftin и Shiftout
```

Некоторые замечания по поводу обозначения выводов микроконтроллеров.

В тексте программы PicBasic Pro к выводам микроконтроллера можно обратиться множеством различных способов. Лучший способ для указания вывода микроконтроллера это просто использовать его название:

```
PORTB.1 = 1  ' Установить первый разряд PORTB в 1
```

В тех случаях, когда необходимо помнить для каких целей используется данный вывод микроконтроллера принято определять этот вывод как переменную с использованием команды VAR и затем уже использовать это название в любых операциях:

```
led VAR PORTA.0   ' Изменим название вывода PORTA.0 на имя led
HIGH led          ' Установим на выводе led (PORTA.0) высокий логический
уровень (+5 вольт)
```

Для совместимости с программами BASIC Stamp, выводы, используемые в текстах программ PicBasicPro можно также обозначать, используя относительную нумерацию, т.е. 0 - 15. Эти выводы физически могут относиться к различным портам микроконтроллера в зависимости от того, сколько контактов имеет микроконтроллер.

Таблица 31

Количество выводов в PIC-микроконтроллере	0 - 7	8 - 15
8-выводов	GPIO*	GPIO*

18-выводов	PORTB	PORTA*
28-выводов (кроме 14C000)	PORTB	PORTC
28-выводов (14C000)	PORTC	PORTD
40-выводов	PORTB	PORTC

*GPIO и PORTA не имеют 8 выводов типа вход - выход.

Если количество выводов порта меньше 8 (например, PORTA), то при обозначении вывода этого порта надо указывать номер только реально существующего вывода, то есть 8 - 12. Поэтому если обозначить вывод этого порта номером 13, 14 или 15 то при ассемблировании это не даст никакого результата. В зависимости от конкретного PIC-микроконтроллера, номер вывода - 0 может быть физическим выводом 6, 21 или 33, но в каждом случае это относится к PORTB.0 (или GPIO.0 для микроконтроллеров с 8 выводами, или PORTC.0 для PIC14C000). Отсюда следует, что в программе PicBasicPro можно ссылаться на выводы как на номера (например. Pin0 или Pin10). Но это возможно, только если один из файлов bsdefs.bas включен в Вашу программу или Вы определили их непосредственно. И естественно, что обозначать выводы можно указанием полного названия вывода (например. PORTA.1). Наилучшим методом обращения к любому выводу микроконтроллера является последний метод. В некоторых случаях, обозначать выводы, удобнее присвоив им определенное имя. Для этого используется команда VAR:

```
led VAR PORTB.3
```

Для того чтобы увеличить совместимость программ PicBasicPro с программами BASIC Stamp, в комплект компилятора включены два файла определений. Это файлы "bs1defs.bas" и "bs2defs.bas". Они могут быть включены в программу PicBasic Pro, для того чтобы названия разрядов и выводов были такими же, как и в BASIC Stamp. В файле BS1DEFS.BAS выводы обозначаются как B0-B13, и W0-W6 и т.д. А в файле BS2DEFS.BAS выводы обозначаются как Ins, Outs, B0-B25, W0-W12 и т.п.

Когда на PIC-микроконтроллер подается питание, то все его выводы устанавливаются в состояние входа. Для того чтобы установить вывод порта в состояние выхода (или входа), установите соответствующим образом регистр TRIS. Запись 0 в один из разрядов регистра TRIS устанавливает соответствующий разряд соответствующего порта в состояние выхода. И наоборот, запись 1 в один из разрядов регистра TRIS устанавливает соответствующий разряд порта в состояние входа. Например:

```
‘ установить все разряды PORTA в состояние выхода.
TRISA = %00000000          ' Или TRISA = 0

‘ установить все разряды PORTB в состояние выхода.
```

```
TRISB = %11111111 ' Или  TRISB = 255

' установить все четные разряды PORTC в состояние выхода, а нечетные в
состояние входа.
TRISC = %10101010 ' Или  TRISC = 170
```

Заключение.

Замечания по поводу использования 12-разрядных микроконтроллеров

В связи с тем, что 12-разрядные микроконтроллеры имеют ограниченные возможности, не все команды компилятора PicBasic Pro будут работать с этими контроллерами.

Ниже приведена таблица команд, которые не поддержаны для этих микроконтроллеров:

Таблица 32

Команда	Причина отсутствия
ADCIN	Отсутствие в составе контроллера АЦП
DATA	Отсутствие в составе контроллера ЕЕПРОМ
DTMFOUT	Недостаточно ОЗУ или стека
EEPROM	Отсутствие в составе контроллера ЕЕПРОМ
FREQOUT	Недостаточно ОЗУ или стека
HPWM	Отсутствии встроенного ШИМ
HSERIN	Отсутствие встроенного порта
HSEROUT	Отсутствие встроенного порта
ON INTERRUPT	Не поддержаны прерывания
READ	Отсутствие в составе контроллера ЕЕПРОМ
READCODE	Отсутствие флаш памяти
RESUME	Не поддержаны прерывания
SERIN2	Недостаточно ОЗУ или стека
SEROUT2	Недостаточно ОЗУ или стека
USBIN	Отсутствие USB порта
USBINIT	Отсутствие USB порта
USBOUT	Отсутствие USB порта
WRITE	Отсутствие в составе контроллера ЕЕПРОМ
WRITECODE	Отсутствие флаш памяти
XIN	Недостаточно ОЗУ или стека
XOUT	Недостаточно ОЗУ или стека

Некоторые команды при использовании 12-разрядных микроконтроллеров модифицированы.

Приложение 1.

Однопроводный интерфейс или интерфейс “Micro-LAN”.

Семейство приборов, известное в настоящее время под общим названием Button, было анонсировано компанией Dallas Semiconductor в 1991 г. под товарным знаком “Touch Memory”, что переводится примерно, как “прикоснись—память”. Это название достаточно полно соответствовало сути нового класса приборов, содержащих, как минимум, постоянную память (а во многих приборах еще и оперативную память и другие узлы), считывание которой возможно при кратковременном прикосновении прибора к специальному контактирующему устройству. Интерфейс, поддерживаемый этими приборами, первоначально получил название “1-Wire”, т. е. однопроводный, а позднее стал называться интерфейсом “Micro-LAN” (Miniature Local Area Network).

Можно назвать следующие привлекательные черты этой системы: 1) наличие большой номенклатуры взаимозаменяемых аппаратных средств (датчиков, аппаратных ключей, модулей памяти и т.д.); 2) наличие простого (дешевого) канала связи (однопроводный интерфейс); 3) возможность работы без внутреннего источника питания, только за счет энергии получаемой вместе с информационным сигналом; 4) гибкая топология сетей сбора информации; 5) возможность организации беспроводной сети на основе автономных энергонезависимых датчиков накопителей информации.

В основу работы сети Micro-LAN положена организация связи главного или ведущего микроконтроллера или микро ЭВМ с периферийными устройствами по однопроводной шине. Однопроводная шина представляет собой систему, состоящую из одного мастера шины и одного или нескольких ведомых (адресуемых ключей, датчиков и т.п.). Сеть Micro-LAN в общем случае строится по древовидной структуре (см. рис. 42). В корне такого дерева располагается ведущий, а на ветвях его находятся ведомые устройства. В узлах сочленения ветвей находятся адресуемые ключи. Для того чтобы получить информацию с определенного устройства ведущий вначале должен подключиться к соответствующему ключу, который после получения определенной команды подключит ведущего к ветви, на которой находится нужное устройство.

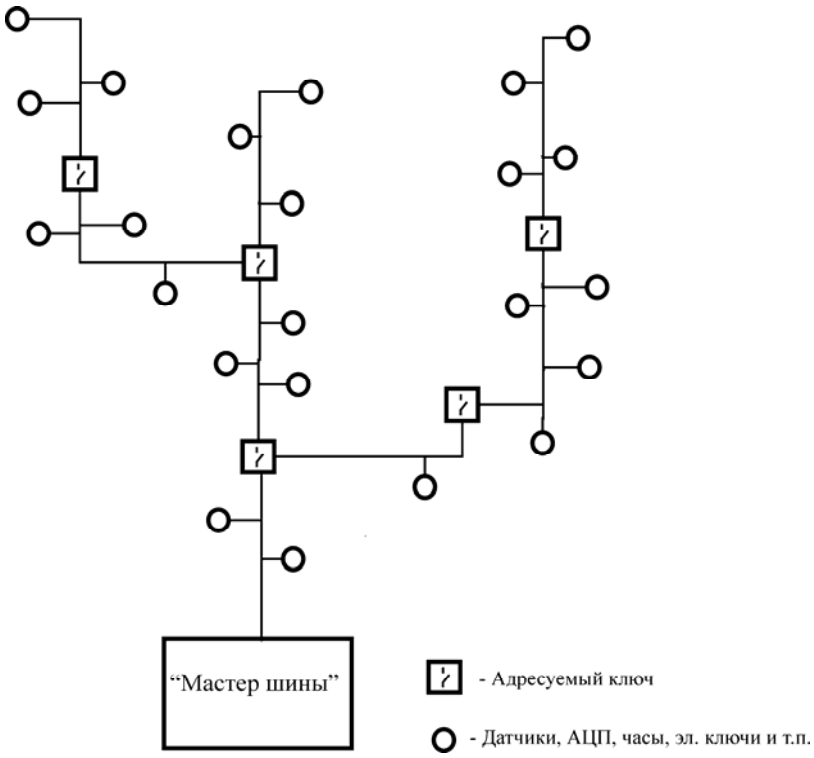


Рисунок 42. Древовидная структура сети Micro-LAN.

Однопроводная шина имеет по определению только один сигнальный провод и провод общей земли (см. рис. 43). Важно, чтобы любое подключенное к ней устройство имело возможность захватить эту шину в некоторый момент времени. Для этого каждое из подсоединенных устройств должно иметь либо выход с открытым коллектором либо выход с тремя состояниями.

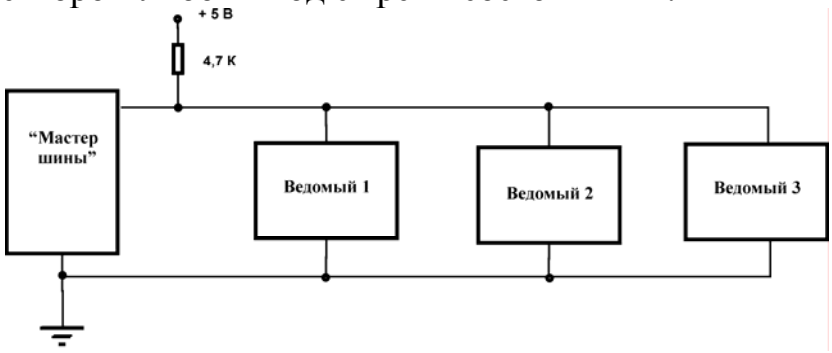


Рисунок 43. Схема подключения абонентов к однопроводной сети.

Подобный однопроводный интерфейс уже реализован в большом числе микросхем. Все эти приборы многофункциональны и весьма полезны при разработке различных приложений. В частности, они обеспечивают возможность совместного использования однопроводной шины вместе с другими подобными приборами. В каждом таком приборе при его изготовлении лазером записывается индивидуальный идентификационный код. Почти все эти приборы (микросхемы) могут получать питание от центрального устройства по однопроводной шине. Наиболее известными у нас являются цифровые термометры DS1820 и DS18S20. Получить доступ к какому-либо устройству можно в такой последовательности:

- Произвести инициализацию
- Выполнить одну из команд функций ПЗУ
- Выполнить команду управления или чтения памяти.

Передача данных по однопроводной шине инициируется главным или ведущим устройством (обычно эта роль отводится основному микроконтроллеру). Перед первым обращением к периферийному устройству сначала происходит обмен сигналами Reset (Сброс) и Presence (Наличие или присутствие). Сигнал Reset формируется ведущим устройством путем перевода линии в низкий уровень на 480-960 мкс. В ответ на этот сигнал периферийное устройство генерирует ответный, устанавливая низкий уровень линии приблизительно на 100 мкс (от 60 до 240 мкс). Импульс присутствия указывает мастеру на то, что хотя бы одно из ведомых устройств присутствует в сети. После того как мастером был получен импульс присутствия, он может выдать в линию команду первой группы, позволяющей исследовать состав группы или если состав известен, обратиться к конкретному устройству. Команды эти однобайтовые и передаются побитно. Причем первым передается младший бит.

Посылки ведутся фиксированными временными интервалами – слотами. Если передается логическая единица то передающее устройство переводит шину на короткий промежуток времени в низкий уровень и возвращает шину вновь в высокое состояние на время слота. Если же необходимо передать логический ноль то передающее устройство переводит шину в низкое состояние и удерживает ее в таком состоянии в течении всего времени слота. Затем шина вновь возвращается в высокое состояние.

Сигналы однопроводной шины.

Для надежного функционирования однопроводной сети требуется строгое выполнение протокола обмена.

Протокол состоит из нескольких типов сигналов. Это импульс сброса, импульс присутствия и импульсы передаваемых данных – логический ноль и логическая единица. На рис. 44 приведена инициализирующая последовательность необходимая для начала любых обменов данными по шине.

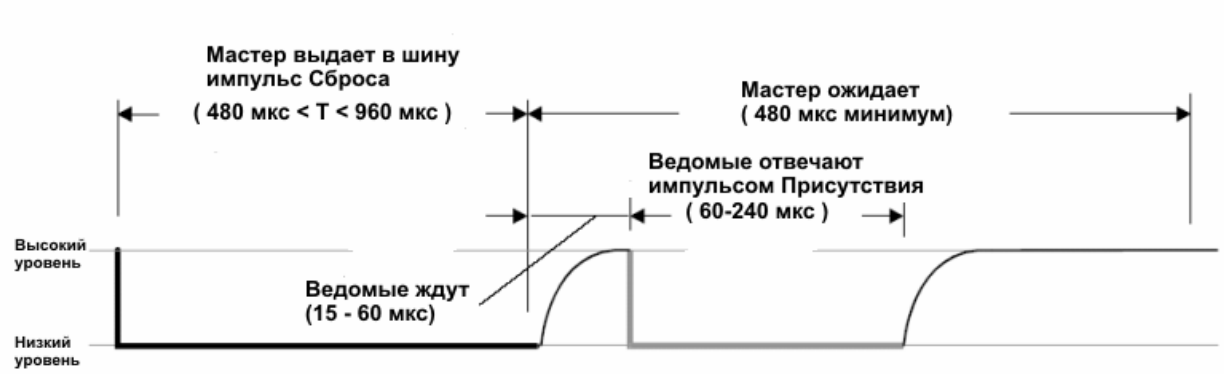


Рисунок 44. Инициализирующая последовательность сигналов однопроводной сети.

Мастер шины выдает в шину импульс сброса (сигнал низкого уровня длительностью не менее 450 мкс и не более 960 мкс). Ведомое устройство,

обнаружив импульс сброса и выждав интервал времени (минимум 15 и максимум 60 мкс) выдает в линию импульс присутствия (сигнал низкого уровня длительностью не менее 60 мкс но не более 240 мкс). Затем начинается непосредственно обмен данными. Мастер выдает команды и в зависимости от команды начинает принимать информацию от периферийных устройств. Передача команд и прием информации происходит с использованием временных интервалов – слотов разделяющих соседние биты и слова. Каждый слот должен иметь длительность не менее 60 мкс. Одним временным слотом передается только один бит данных. Между соседними слотами должна присутствовать восстановительная пауза длительностью не менее 1 мкс. Ведомым устройствам допускается иметь существенные отличия от номинальных выдержек времени. Однако это требует более точного отсчета времени Мастером, чтобы гарантировать корректность связи с подчиненными, у которых различаются временные базисы. Таким образом, следует в точности выдерживать временные границы, рассматриваемые в следующих разделах.

Все сеансы передачи данных инициируются Мастером.

Основные сигналы шины.

Ведущий инициирует каждую связь на битном уровне. Это означает, что передача каждого бита, независимо от направления, должна быть инициирована ведущим. Это достигается установкой низкого уровня на шине, который синхронизирует логику всех остальных устройств. Существует 5 основных сигналов для связи по однопроводной шине: “Запись лог. 1”, “Запись лог. 0”, “Чтение” и рассмотренные нами “Сброс” и “Присутствие”.

Сигнал “Запись лог. 0”.

Сигнал “Запись лог. 0” показан на рисунке 50. Ведущий формирует низкий уровень в течение не менее 60 мкс, но не дольше 120 мкс.

Сигнал “Запись лог. 1”.

Сигнал “Запись лог. 1” показан на рисунке 45. Мастер устанавливает низкий уровень в течение 1…15 мкс. После этого, в течение оставшейся части временного слота он освобождает шину.

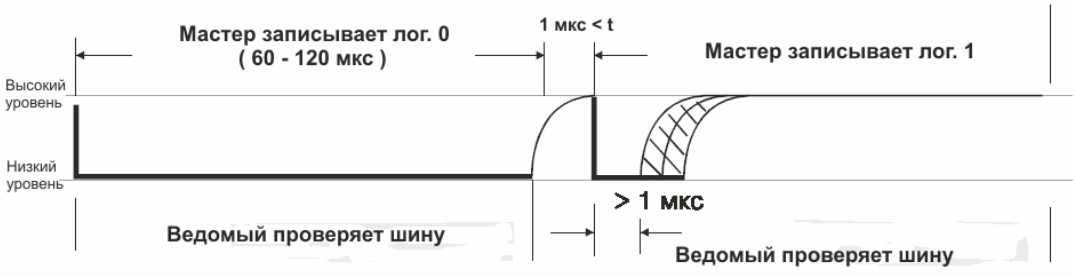


Рисунок 45. Диаграммы записи сигналов лог. 0 и лог. 1 в ведомое устройство.

Сигнал “Чтение”.

Если необходимо считать данные из ведомого устройства Мастер генерирует слоты чтения. Слот чтения инициируется сразу же, как только Мастер переводит шину из высокого состояния в низкое. Шина данных должна оставаться в низком состоянии как минимум 1 мкс. Данные от ведомого устройства будут истинными в течении 15 мкс после снижения уровня Мастером. Это значит, что Мастер после формирования слота чтения в течении 1 мкс должен освободить шину чтобы дать возможность ведомому устройству выставить на шине свои данные. Процесс “Чтение” показан на рисунке 46. Ведущий устанавливает низкий уровень в течение 1…15 мкс. После этого подчиненный удерживает шину в низком состоянии, если желает передать лог. 0. Если необходимо передать лог. 1, то он просто освобождает линию. Сканирование шины необходимо выполнять по истечении 15 мкс после установки низкого уровня на шине. Если смотреть со стороны ведущего, сигнал “Чтение” является, в сущности, сигналом «Запись лог. 1». Собственно внутреннее состояние подчиненного будет определять это сигнал «Запись лог. 1» или «Чтение».

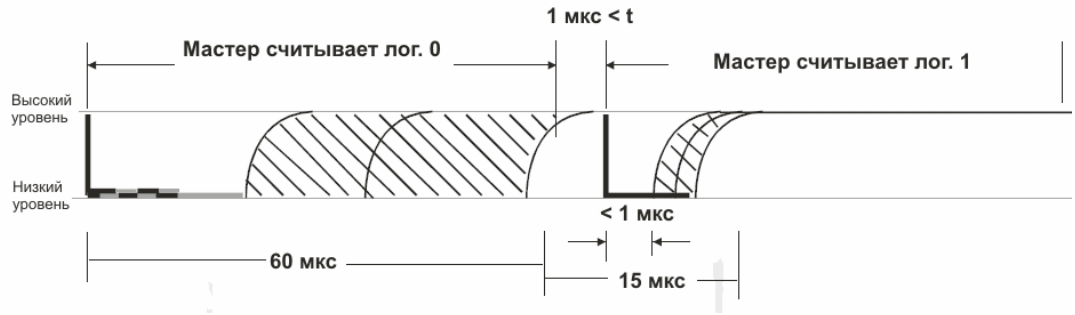


Рисунок 46. Диаграммы считывания Мастером лог. 0 и лог. 1.

Команды функций ПЗУ.

Все команды функции ПЗУ имеют длину в 8 бит. Причем младший бит команды передается первым. Каждая ИС из семейства 1-Wire содержит ПЗУ, в котором хранится уникальный 64-разрядный идентификационный код (ИК). Данный код может использоваться для адресации или идентификации конкретной ИС на шине. Идентификатор состоит из трех частей: 8 бит кода семейства, 48 бит серийного номера и 8 бит CRC-кода, вычисленного от первых 56 бит. Имеется небольшой набор команд, который работает совместно с 64-разр. ИК. Эти команды называются команды функций ПЗУ. В таблице 2 приведен перечень четырех команду ПЗУ.

Таблица 33 – Команды ПЗУ

Команда	Код	Назначение
READ ROM (ЧТЕНИЕ ПЗУ)	33H	Чтение ПЗУ (идентификация)
SKIP ROM	CCH	Игнорирование ПЗУ (пропуск

(ПРОПУСК ПЗУ)		адресации)
MATCH ROM (СОВПАДЕНИЕ ПЗУ)	55H	Выбор ПЗУ (выбор ведомого)
SEARCH ROM (ПОИСК ПЗУ)	F0H	Исследование ПЗУ (получение идентификационных данных о всех устройствах на шине)

Команда «Чтение ПЗУ».

Эта команда позволяет Мастеру шины прочитать 64 разрядный ИК ведомого устройства. Этот код состоит из 8 бит кода принадлежности к семейству однопроводных устройств, 48 битовый серийный номер и 8 битового циклического избыточного кода (ЦИК). Эту команду можно использовать если к ведущему подключено только одно ведомое устройство. Если же к шине подключено больше ведомых устройств, то в этом случае весьма вероятен конфликт данных.

Команда «Игнорирование ПЗУ».

Эта команда экономит время, когда к шине подключено только одно ведомое устройство и Мастер это знает. Она позволяет Мастеру обратиться к функциям памяти без предварительного генерирования 64 разрядного кода ведомого устройства. Если же на шине присутствует несколько ведомых устройств то этой командой можно передать общую для всех них команду.

Команда «Выбор ПЗУ»

Команда “Выбор ПЗУ” используется для адресации конкретного подчиненного устройства на шине. После выполнения команды “Совпадение ПЗУ” передается 64-разрядный ИК. По завершении, только тому подчиненному устройству, который принял свой ИК, разрешается отвечать после приема следующего импульса сброса.

Команда «Исследование ПЗУ».

Команда “Исследование ПЗУ” может использоваться, если идентификаторы подчиненных устройств неизвестны заранее. Это делает возможным обнаружить идентификаторы всех подчиненных устройств, подключенных к шине. Для этого, вначале передается команда “Пропуск ПЗУ”. Каждый ведомый размещает первый бит своего идентификатора на шине. Мастер считывает результат, как логическое «И» над всеми первыми битами идентификаторов всех ведомых устройств. Затем ведомые размещают на шине двоичное дополнение к первому биту своего идентификатора. Мастер считывает состояние шины, которое есть результат логического «И» над всеми дополнениями к первому биту идентификатора всех

ведомых устройств. Если все устройства в первом разряде ИК содержат 1, то Мастер считает 10b. Аналогично, если значения 1 разряда всех устройств равно 0, то Мастер примет 01b. В этих случаях, бит может быть сохранен как значение первого бита всех адресов. После этого Мастер снова выполняет размещение этого бита на шине, чем сигнализирует ведомым о необходимости дальнейшего продолжения передачи разрядов ИК. Если на шине будут присутствовать устройства, как с лог. 0, так и с лог. 1 в первом бите идентификатора, то Мастер примет 00. В этом случае, Мастер должен выбрать с какими адресами продолжать работу, с лог. 0 или 1 в первом разряде. Выбор передается по шине, указывая выбранным подчиненным о необходимости дальнейшей передачи ИК, а остальные ведомые переходят в режим ожидания.

Затем Мастер переходит к считыванию следующих бит и этот процесс повторяется до считывания всех 64 бит. В результате Мастер обнаруживает полный 64-разрядный идентификатор. Для поиска других идентификаторов необходимо снова инициировать команду «Поиск ПЗУ», но в этом случае при возникновении несоответствий сделать другой выбор. Если придерживаться данной последовательности, то в конечном счете можно обнаружить все ведомые устройства. Обратите внимание, что после выполнения первого поиска все ведомые, кроме одного, переходят в режим ожидания. Таким образом, в этом состоянии связаться с активным ведомым можно с помощью команды «Совпадение ПЗУ».

Команды памяти/функций.

Команды памяти/функций – команды, которые специфичны для конкретного типа устройства или их класса работающего по однопроводному интерфейсу. Обычно эти команды относятся к чтению или записи внутренней памяти и регистров ведомых устройств. Количество таких команд фиксировано, но все они поддерживаются конкретным видом ведомого устройства. Последовательность чтения и записи определена для каждой ИС.

Проверка истинности данных циклическим избыточным кодом ЦИК (CRC).

Как уже говорилось, отличительной чертой всех устройств работающих по однопроводному протоколу фирмы Dallas Semiconductor является наличие у каждого устройства индивидуального идентификационного кода. Размер этого кода составляет 64 бита. Младший байт содержит код, свидетельствующий о принадлежности к семейству с однопроводным интерфейсом, определяя одновременно функциональную принадлежность прибора. Следующие 6 байт ПЗУ содержат уникальный серийный номер конкретного экземпляра. И наконец старший байт ПЗУ представляет собой результат проверки истинности данных циклическим избыточным кодом – ЦИК (cyclic redundancy check – CRC). Этот код вычисляется на основе предыдущих семи байтов. Если ЦИК рассчитанный Мастером на основе принятых (семи младших байтов) совпадает с содержимым 8

байта ПЗУ, то считается, что данные приняты верно. Если эти данные не совпадают, то процесс чтения ПЗУ повторяется.

Работу схемы расчета ЦИК проще всего объяснить на примере ее аппаратной реализации. В этом случае схема представляет собой сдвиговый регистр с обратной связью (см. рис. 47).

Команды памяти/функций.

Команды памяти/функций – команды, которые специфичны для конкретного типа ИС или их класса. Обычно эти команды относятся к чтению или записи внутренней памяти и регистров подчиненных ИС. Количество таких команд фиксировано, но все они поддерживаются конкретным видом подчиненного устройства. Последовательность чтения и записи определена для каждой ИС. В связи с этим команды памяти здесь в подробностях не рассматриваются.

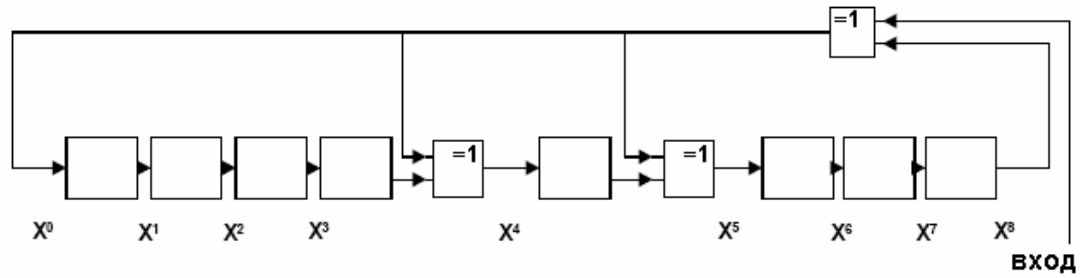


Рисунок 47. Аппаратная реализация 8-битового алгоритма проверки истинности данных ЦИК, соответствующая математической функции $X^8+X^5+X^4+1$.

Процедуру выделения ошибки часто описывают в виде полиноминальной функции формальной переменной X . Показатели степени ненулевых членов полинома показывают, на какие разряды сдвигового регистра замыкаются петли обратной связи. Число разрядов сдвигового регистра (в аппаратной реализации) или степень полинома (при математическом описании) определяют разрядность вычисляемого ЦИК. Обычно при цифровом обмене данными используют 16-разрядные ЦИК. Фирма Dallas Semiconductor при чтении 64 битовых ПЗУ своих устройств использует 8-разрядные ЦИК. Расположение точек входа петель обратной связи, обозначенных на рис. 47 логическими элементами «исключающее ИЛИ» (или, что то же самое, степени ненулевых членов полинома) определяет свойства ЦИК и способность алгоритма выявить определенные классы ошибок при передаче данных. Алгоритм, приведенный на рисунке, способен распознать следующие классы ошибок:

- Любое нечетное число ошибок в любом месте 64-битовой последовательности.
- Все двойные ошибки (ошибки в двух смежных битах) в любом месте 64-битовой последовательности.
- Любые кластеры ошибок в пределах 8-битового «окна», т.е. кластеры, содержащие от 1 до 8 неправильных битов.
- Большинство кластерных ошибок с размерами кластера, превышающим 8 разрядов.

Входные данные подаются на один из входов двухвходового логического элемента «исключающее ИЛИ» (XOR). Ко второму входу этого элемента подключается выход старшего разряда 8-разрядного сдвигового регистра. Математически сдвиговый регистр может рассматриваться как схема деления. Входные данные – это делимое, а сдвиговый регистр с цепями обратной связи служит делителем. Целая часть полученного частного отбрасывается, а остаток представляет собой искомый ЦИК, который окажется в сдвиговом регистре, как только последний бит данных пройдет на вход. Из такого аппаратного описания алгоритма видно, что результат (значение ЦИК) очень сложным образом зависит от предыстории контролируемой последовательности битов. Поэтому комбинации ошибок, способными пройти незамеченными через такой контроль, очень редки.

Приложение 2.

Протокол X10.

Исторически X10 (он же X-10) является первым стандартом передачи сигналов домашней автоматизации. Технология X10 была изобретена и запатентована в конце 70-х годов шотландской инженерной фирмой PICO Electronics, со штаб-квартирой в Англии. Инженеры PICO впоследствии перебазировались в Нью-Йорк, и продолжили свои работы по развитию методов дистанционного управления проигрывателями, используя готовую электропроводку для передачи сигналов. После выполнения девяти экспериментальных проектов именно десятый оказался наиболее успешным (отсюда название X10) и именно его результаты определили стандарт передачи сигналов по силовой электропроводке. Группа разработчиков X10 назвала свою новую компанию по автоматизации домов X10 USA (X10 Inc).

Сегодня, много компаний производят X10-совместимые устройства: X10 Inc., Leviton, Marmitek, IBM, SmartLinc, PowerHouse и другие.

Что такое X10?

X10 - широко используемый стандарт в области домашней автоматизации. X10 определяет метод и протокол передачи управляющих сигналов-команд (включить, выключить, ярче, темнее и т.д.) по силовой электропроводке на электронные модули, к которым подключены управляемые электробытовые и осветительные приборы.

Всего в сеть X10 может быть объединено до 256 групп устройств с разными адресами.

Технология передачи сигналов X10.

X10 - протокол взаимодействия передатчиков и приемников, включает в себя передачу и прием сигналов по силовым линиям (бытовая электросеть).
Этими сигналами являются пакеты импульсов, которые кодируют цифровую информацию.

Бинарная единица в протоколе X10 представляет собой пакет длительностью 1

мс заполненный импульсами частотой 120 Гц и амплитудой 5 В. Бинарный ноль - отсутствие такого сигнала.

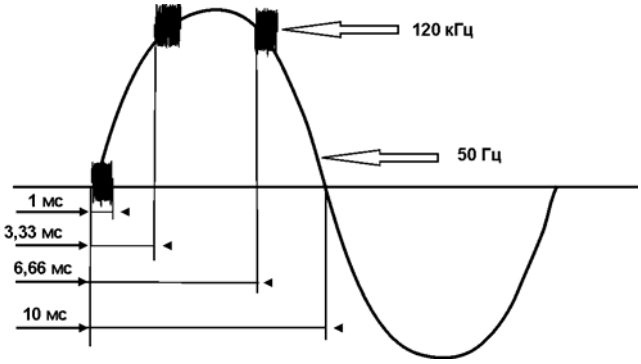


Рис. 48. Осциллограмма сетевого напряжения, в котором присутствуют сигналы протокола X10.

Передача импульсов синхронизирована переходом переменного напряжения через нулевой уровень в пределах 200 мкс интервала. Так как дом может быть подключенным к трёхфазной сети и на каждой фазе могут присутствовать приемники X10 сигналов, то единичный импульс повторяется три раза, а каждый пакет сдвинут по времени от предыдущего на 2.778 мс для сети 60 Гц. Если же вы имеете дело с сетью 50 Гц, то сигнал повторяется через 3,33 мс.

Для передачи команды X10 требуется одиннадцать периодов сетевого напряжения.

Первые два цикла передают стартовый код, следующие четыре цикла представляют код дома (с A по P) и последние пять циклов передают код прибора (с 1 по 16) или код функции (ВКЛ, ВЫКЛ и т.д.), т.е. ключевой код.

Рис. 49. Формат передачи полного кода.

Этот полный код (стартовый код + код дома + ключевой код) всегда передается дважды непрерывным блоком (Рис. 49) Между блоками разных команд всегда должен быть перерыв в три цикла силового напряжения. Исключением из этого правила являются блоки команд ЯРЧЕ/ТЕМНЕЕ, которые передаются последовательно (минимум два блока) без задержек.

Стартовый код - это уникальный код, всегда равный 1110 и не имеющий дополняющих бит в смежных полупериодах, т.е. значащие биты передаются на каждый переход силового напряжения через нуль.

Внутри каждого блока, код дома и ключевой код должны передаваться с дополняющими до единицы кодами в смежных полупериодах силового

напряжения. Например, если единичный импульс передан в первой половине периода, то во второй не должно быть никакого сигнала (нулевой бит). Смотри рисунок 50.

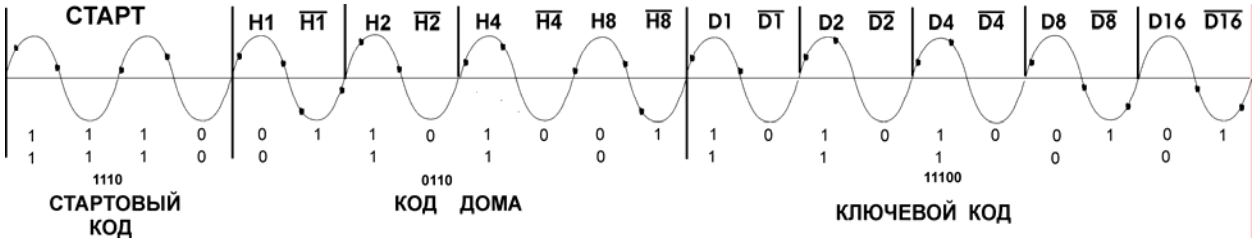


Рис. 50. Распределение кодов на осциллограмме сетевого напряжения.

Таблица 34.

Таблица значений кодов протокола X10

КОДЫ ДОМОВ					КЛЮЧЕВЫЕ КОДЫ					
	H1	H2	H4	H8		D1	D2	D4	D8	D16
A	0	1	1	0	1	0	1	1	0	0
B	1	1	1	0	2	1	1	1	0	0
C	0	0	1	0	3	0	0	1	0	0
D	1	0	1	0	4	1	0	1	0	0
E	0	0	0	1	5	0	0	0	1	0
H	1	1	0	1	6	1	1	0	1	0
F	1	0	0	1	7	1	0	0	1	0
G	0	1	0	1	8	0	1	0	1	0
I	0	1	1	1	9	0	1	1	1	0
J	1	1	1	1	10	1	1	1	1	0
K	0	0	1	1	11	0	0	1	1	0
L	1	0	1	1	12	1	0	1	1	0
M	0	0	0	0	13	0	0	0	0	0
N	1	0	0	0	14	1	0	0	0	0
O	0	1	0	0	15	0	1	0	0	0
P	1	1	0	0	16	1	1	0	0	0
					Все модули ВЫКЛ	0	0	0	0	1
					Весь свет ВКЛ	0	0	0	1	1
					ВКЛ	0	0	1	0	1
					ВЫКЛ	0	0	1	1	1
					ТЕМНЕЕ	0	1	0	0	1
					ЯРЧЕ	0	1	0	1	1

Весь свет ВЫКЛ	**0**	**1**	**1**	**0**	**1**
Дополнительный код	**0**	**1**	**1**	**1**	**1**
[1] HALT запрос	**1**	**0**	**0**	**0**	**1**
HALT ответ	**1**	**0**	**0**	**1**	**1**
[2] Предустановка диммера	**1**	**0**	**1**	**x**	**1**
[3] Дополнительные данные	**1**	**0**	**1**	**1**	**1**
Статус = ВКЛ	**1**	**1**	**0**	**0**	**1**
Статус = ВЫКЛ	**1**	**1**	**0**	**1**	**1**
Запрос статуса	**1**	**1**	**1**	**1**	**1**

1) - HAIL запрос (запрос-приветствие) передается для нахождения передатчиков в зоне покрытия. Это позволяет выставить различные коды домов в случае получения ответа Hail Acknowledge.

2) - В коде функции Pre-Set Dim, бит D8 вместе с четырьмя битами кода дома составляет блок из 5 бит {D8H8H4H2H1}, определяющий абсолютный уровень диммера.

3) - Функция Extended Data (дополнительные данные) предшествует последовательности байт (8 бит) произвольной длины, которые представляют аналоговые данные после аналогово-цифрового преобразования. Код функции и байты данных передаются непрерывно, без пауз. Первый байт данных может указывать на количество байт в последовательности. Если при передаче в последовательности байт допущены паузы, то модуль – приемник может выполнить ошибочную операцию.

Функция Extended Code эквивалентна Extended Data: последовательность байт (без пауз), которые представляют дополнительные коды. Это позволяет разработчикам использовать больше 256 имеющихся кодов.

Первые из 16 ключевых кодов определяют номер модуля, который в дальнейшем будет принимать и выполнять команды (ВКЛ, ВЫКЛ, ЯРЧЕ, ТЕМНЕЕ) до переопределения управляемого модуля. Бит D16 называется "функциональным битом", если он равен 1, то передается функция, иначе код модуля.

Приведем пример. Чтобы включить 5-ый модуль в "доме К", нужно послать по электросети следующую строку бит:

1110010110100101011001111001011010010101100100000011100101101001011001
1011100101101001011001 10.

Эта посылка содержит 94 бита, и займет 47 циклов силового напряжения или 0,94 с (почти секунда!). Поэтому, когда вы нажимаете на кнопку ВКЛ, свет

включается с запаздыванием. Реакция на команды "Весь свет ВКЛ" или "Все модули ВЫКЛ" заметно быстрее, т.к. не передается код модуля.

Недостатки протокола X10 и борьба с ними.

Низкая скорость передачи информации.

Передача импульсов синхронизирована с переходом через ноль напряжения электросети, например, команда "ВКЛ", содержащая 94 бита, займет 47 циклов силового напряжения или 0,94 сек. (почти секунда!). Но если после этого послать команду "ВЫКЛ" на этот же модуль, то она выполнится в два раза быстрее, т.к. не надо передавать код устройства.

Низкая помехозащищенность

X10 использует амплитудную модуляцию, поэтому помехи в электросети легко могут "забить" полезный сигнал. Основные источники помех в электросети - электродвигатели (холодильник, стиральная машина, электродрель и т.п.) и приборы с тиристорными регуляторами (кроме устройств X10). Помехоподавляющие конденсаторы электробытовых приборов также могут фильтровать высокочастотный 120 кГц сигнал X10.

Проблема ложного срабатывания.

Ложные срабатывания от помех в электросети, вызванных бытовыми электроприборами маловероятны.

Более вероятны ложные срабатывания, если, например, два устройства X10 одновременно подают в электрическую сеть свои управляющие сигналы. Так как проблема "столкновений" в протоколе X10 практически никак не решена, то такие ситуации возможны. Хотя вероятность таких коллизий и мала (длительность одной посылки управляющих сигналов порядка одной секунды), но ненулевая.

Отсутствие обратной связи приемника с передатчиком.

В X10 нет сигналов квитирования (квитков), которые бы подтверждали принятие и исполнение приемниками команд от передатчиков. Хотя команды повторяются дважды, существует вероятность того, что если помехи электросети "съедят" сигнал, то ожидаемого действия не произойдет.
В современных модулях существует возможность запрашивать статус модуля, тем самым контролировать выполнение команд.

Возможны конфликты устройств X10 разных производителей.

Изначальное несовершенство протокола X10 потребовало внесения в него различных дополнений. Одно из таких дополнений – extended codes (расширенные или дополнительные коды). В силу того, что каждый производитель разрабатывал эти коды самостоятельно, устройства разных фирм-изготовителей не всегда корректно ретранслируют и выполняют управляющие сигналы, передаваемые устройствами других фирм.

Возможен несанкционированный доступ к устройствам X10 по электросети.

Если в двух соседних квартирах, использующих одну и ту же фазу осветительной сети, используются устройства X10, то, естественно, возникает вопрос о том, как избежать попадания управляющих сигналов X10 из одной

квартиры через электрическую сеть в другую квартиру.

Возможна внешняя атака на домашнюю сеть X10 посредством «чужого» радиопульта.

Протокол X10 не предусматривает никакой системы паролей и предполагает совместимость любого передатчика управляющих сигналов с любым приемником (исполнительным устройством). Наличие системы паролирования заметно усложнило бы устройства X10 и создало разработчикам в 80-е годы дополнительные трудности, поскольку первоначально устройства X10 изготовлялись на жесткой логике, без применения микропроцессоров. В современных условиях добавление паролей к передаваемым управляющим сигналам X10 - вполне посильная задача для микропроцессорной техники (например, по аналогии с динамическими кодами, применяемыми в автомобильных сигнализациях); но внедрение такого механизма нарушило бы совместимость с ранее разработанными устройствами X10.

Подборка ссылок в INTERNET.

Из огромного множества ссылок на различные ресурсы в INTERNET, в которых говорится о PIC-контроллерах и их программировании, а также некоторых вещах, о которых говорится в этой книге, я выбрал лишь малую часть. Я надеюсь, что они помогут Вам в освоении программирования микроконтроллеров.

http://www.microchip.com – официальный сайт разработчика PIC-микроконтроллеров.

http://www.microchip.ru – русскоязычный сайт дилеров компании Microchip.

http://www.microengineeringlabs.com – официальный сайт разработчика компилятора PIC Basic.

http://www.melabs .com – официальный сайт разработчика компилятора PIC Basic.

http://www.picbasic.co.uk – официальный сайт разработчика компилятора Basic PROTON.

http://www.parallax.com/index.asp - официальный сайт разработчика компилятора BasicStamp.

http://www.mecanique.co.uk - официальный сайт разработчика программы MicroCode Studio.

http://www.mikroelektronika.co.yu - официальный сайт разработчика компилятора MicroBasic.

http://www.mikroelektronika.co.yu/russian - русскоязычный сайт этой же фирмы.

http://www.rentron.com/PicBasic2.htm - сайт, где Вы сможете найти ответы на вопросы программирования различных микроконтроллеров и на различных языках программирования.

http://www.kazus.ru – русскоязычный сайт, где Вы сможете найти много интересного и поучаствовать в дискуссиях на форумах с людьми, которых интересуют те же проблемы.

Литература.

1. Chuck Hellebuyck Programming PIC Microkontrollers With PICBASIC - Elsevier Science, 2003
2. Таверньe К. PIC-микроконтроллеры. Практика применения. – М: ДМК Пресс, 2002.
3. Предко М. Справочник по PIC-микроконтроллерам. – М.: ДМК Пресс, 2002.
4. Предко М. Устройства управления роботами. Схемотехника и программирование. – М: ДМК Пресс, 2004.
5. Ульрих В. А. Микроконтроллеры PIC16X7XX. – С.-Петербург: Наука и техника, 2002.
6. By Les Johnson. Experimenting with the PICBASIC PRO COMPILER. - Rosetta Technologies. 2000.