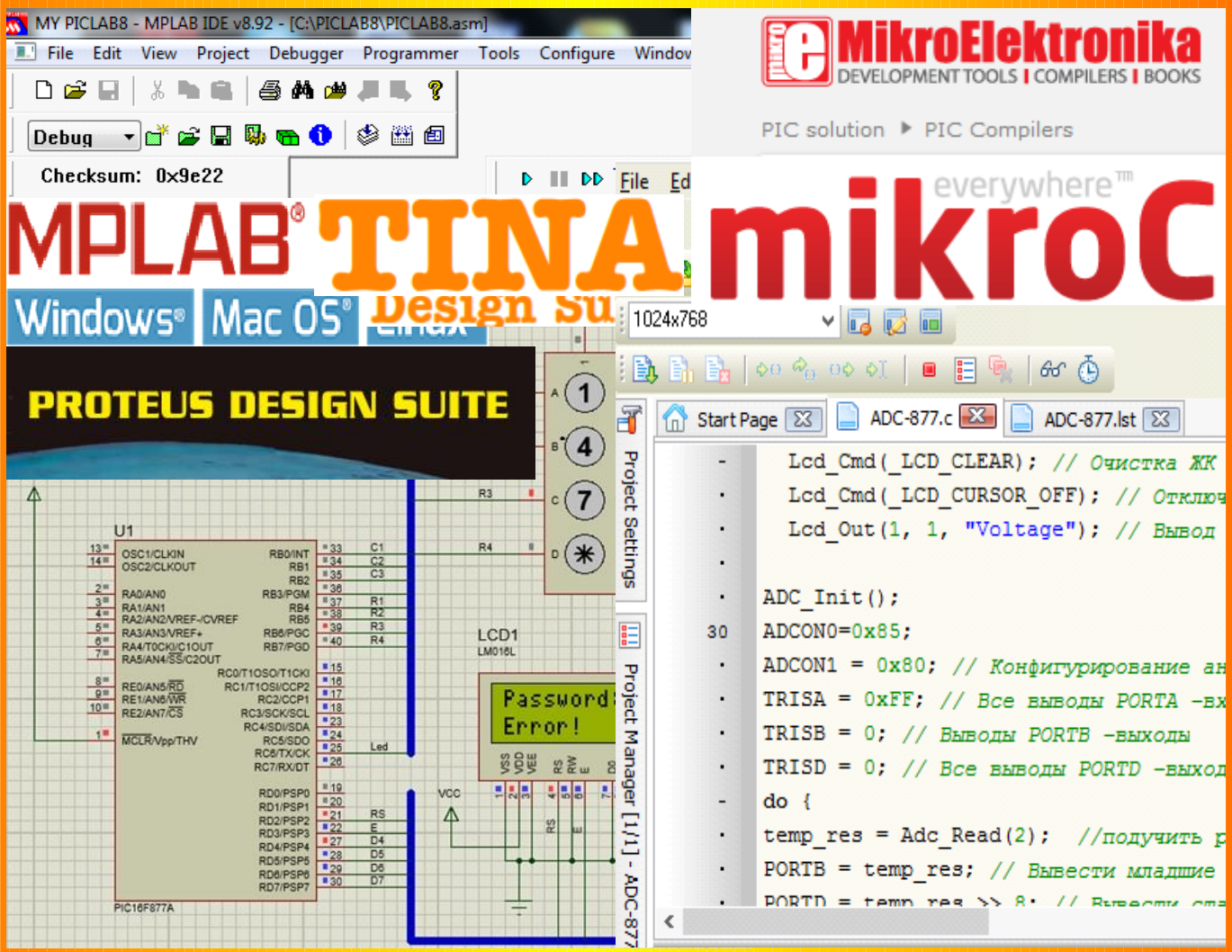
В.А. АЛЕХИН

МИКРОКОНТРОЛЛЕРЫ PIC
ОСНОВЫ ПРОГРАММИРОВАНИЯ И МОДЕЛИРОВАНИЯ В ИНТЕРАКТИВНЫХ СРЕДАХ MPLAB IDE, mikroC, TINA и Proteus



МОСКВА 2015

МИКРОКОНТРОЛЛЕРЫ PIC

ОСНОВЫ ПРОГРАММИРОВАНИЯ И МОДЕЛИРОВАНИЯ В ИНТЕРАКТИВНЫХ СРЕДАХ MPLAB IDE, mikroC, TINA, Proteus

Учебное пособие

МОСКВА 2015

УДК 004.312

ББК 32.973-04

Автор: В.А. Алехин.

Редактор: В.Ю. Маслов

Учебное пособие для лабораторно-практических занятий и курсового проектирования по микропроцессорной технике содержит сведения о современных программах проектирования радиоэлектронных устройств на микроконтроллерах. Изучаются и используются бесплатные интегрированные среды разработки для микроконтроллеров MPLAB IDE и mikroC PRO, программы компьютерного моделирования и проектирования TINA и Proteus. Работая с учебным пособием, студенты могут самостоятельно освоить методы программирования микроконтроллеров с использованием блок-схем, на языках ассемблера и Си, в среде mikroC, программирование и отладку программ, моделирование устройств с микроконтроллерами в программах TINA и Proteus. Для изучения выбраны популярные микроконтроллеры PIC16F84A и PIC16F877A.

Выполнение пятнадцати лабораторных работ поможет изучить функционирование микроконтроллеров, получить практические навыки программирования, отладки и моделирования устройств с микроконтроллерами.

Такое содержание учебного пособия будет способствовать формированию у студентов профессиональных компетенций.

Учебное пособие может быть полезно любителям электроники и инженерам для изучения современных программ проектирования устройств на микроконтроллерах.

Материал предназначен для студентов, изучающих дисциплину «Микропроцессорная техника».

Печатаются по решению редакционно-издательского совета Московского государственного университета информационных технологий, радиотехники и электроники (МИРЭА).

Рецензенты: профессор М.Л. Белов, профессор В.Г. Лысенко

ВВЕДЕНИЕ

Электроника и микропроцессорная техника являются фундаментальной основой производства современного оборудования, технических систем управления, роботов, информационно-измерительных устройств, разнообразных бытовых приборов. Изучение и практическое освоение методов проектирования электронных устройств с микроконтроллерами является актуальной и сложной задачей, требующей использования разнообразных современных программных средств.

Традиционно при изучении микроконтроллеров составленную программу загружали с помощью программатора в реальную микросхему. Для этого надо было извлечь микросхему из панельки в макетной плате с кнопками, светодиодами, внешним питанием, вставить микросхему в панельку программатора, стереть старую программу, записать («прошить») исправленную программу, вернуть микросхему на макетную плату и снова провести натурный эксперимент. Такие операции при отладке программы требуется проводить 20-30 раз. Существенное ускорение изучения дает применение современных компьютерных программ для программирования и интерактивного моделирования устройств на микроконтроллерах.

Производители предлагают широкий ассортимент микроконтроллеров для выполнения самых разнообразных задач. Обоснованный выбор микроконтроллера и успешную разработку устройства можно сделать только после изучения общих принципов функционирования микроконтроллеров, основ программирования и компьютерного проектирования. Обстоятельно изучив хотя бы один из распространненых типов микроконтроллеров и программные средства разработки, Вы сможете более уверенно подходить к решению новых задач.

В учебном пособии изучаются популярные микроконтроллеры PIC16F84A и PIC16F877A компании Microchip Technology Incorporated, которые широко применяются в самых разнообразных электронных устройствах. Выбор этих микроконтроллеров обусловлен, в частности, тем, что Microchip предоставляет бесплатную интегрированную программную среду разработки микроконтроллеров MPLAB IDE [1], которая позволяет писать, отлаживать, оптимизировать текст программы, включает в себя редактор текста, симулятор и менеджер проектов, поддерживает работу эмуляторов, программаторов и других отладочных средств. Среду MPLAB IDE можно применять в компьютерных классах и на домашних компьютерах.

Кроме того, Microchip предоставляет подробные технические описания микроконтроллеров на русском языке [2,3].

В среде MPLAB IDE v.8.92 мы изучим программирование на языках ассемблера и Си. Язык ассемблера является машинно-ориентированным. Система команд привязана к микроконтроллеру или некоторой группе микроконтроллеров. Программа, составленная в ассемблере выполняется наиболее быстро, но написание такой программы весьма трудоемко и программа занимает много места.

Язык Си относится к языкам среднего уровня, он весьма популярен у разработчиков микропроцессорных устройств и применим к широкому кругу микроконтроллеров. Однако для создания исполняемого файла программы требуются компиляторы, которые из программы, написанной на Си, создают программу в ассемблере и исполняемый HEX-файл. Есть много различных компиляторов. Мы выбрали бесплатный компилятор MPLAB XC8 C COMPLIER [4,5] доступный для использования в компьютерном классе и дома.

Чтобы облегчить программирование на Си сопряжения микроконтроллеров с внешними периферийными устройствами, компиляторы имеют специальные библиотеки подпрограмм и библиотечных функций. Однако, в компиляторе MPLAB XC8 эта библиотека недостаточно полная. Поэтому мы будем изучать

среду mikroC PRO for PIC v.6.5.0 [6] компании MikroElektronika. Этот мощный инструмент разработки программ для PIC микроконтроллеров относится к языкам высокого уровня. Он сконструирован, чтобы обеспечить программисту наименее трудоемкие решения по созданию приложений для встраиваемых систем. Очень важно то, что программа mikroC PRO for PIC тоже является «условно бесплатной»: для бесплатного использования исполняемый HEX-файл не должен быть более 2кБайт. В наших примерах это выполняется.

Составленную и отлаженную в MPLAB IDE или mikroC программу мы будем испытавать в моделях микроконтроллерных устройств, используя интерактивные программные среды TINA 9 [7] и Proteus [8]. Программная среда TINA 9, весьма полезная для студентов при изучении курса «Электротехника, электроника и схемотехника» [9], имеет достаточно большую библиотеку микроконтроллеров и позволяет программировать и моделировать устройства с микроконтроллерами. Мы будем использовать эту среду при изучении микроконтроллера PIC16F84A.

Моделирование сложных микропроцессорных устройств с разнообразной периферией мы будем проводить в программной среде Proteus v.8.1 фирмы Labcenter Electronics [10]. Эта система имеет более обширную библиотеку моделей электронных компонентов, периферийных модулей и микроконтроллеров по сравнению с программой TINA. Интерфейс Proteus сложнее, чем в программе TINA, но его освоение даст Вам новые мощные инструменты проектирования микропроцессорных устройств

Процесс отладки программ и моделирование микропроцессорного устройства проходит в интерактивном режиме. MPLAB IDE и mikroC при компиляции показывают Вам ошибки в орфографии и синтаксисе, а TINA и Proteus позволяют найти логические ошибки в программе, если модель функционирует неправильно. Этот процесс напоминает увлекательную игру, в которой Вы при желании и настойчивости обязательно одержите победу.

Освоив программирование и моделирование микроконтроллеров PIC по данному учебному пособию, Вы сможете подобрать подходящий программатор, почитать дополнительные материалы и заняться проектированием настоящих живых микропроцессорных устройств. Программы TINA и Proteus помогут Вам разработать печатные платы и выполнить прошивку микроконтроллеров.

Изучение микроконтроллеров интересует многих разработчиков и любителей электроники. Издано немало полезных книг по данной тематике [11, 12, 13, 14, 15]. Цель этой книги помочь любому студенту, инженеру или начинающему любителю электроники быстро перейти к созданию собственных проектов и устройств на микроконтроллерах. Для получения практических навыков книга содержит 15 лабораторных работ с листингами программ, схемами моделей и подробным описанием выполнения заданий.

Надеюсь, что Вы будете с увлечением работать в программах TINA, MPLAB IDE, mikroC, Proteus, так как каждая задача содержит элементы творчества и доставляет удовольствие от успешного решения.

Желаю успехов!

В.А. Алехин

Доктор технических наук,
профессор кафедры вычислительной техники
Московского государственного университета
информационных технологий, радиотехники и
электроники
(МИРЭА)

Глава 1. ИЗУЧЕНИЕ МИКРОКОНТРОЛЛЕРА PIC16F84A И ЕГО СИСТЕМЫ КОМАНД

Микроконтроллеры семейства PIC (*Peripheral Interface Controller*) компании Microchip объединяют все передовые технологии, имеют широкую номенклатуру и используются в устройствах, предназначенных для разнообразных сфер применения.

Высокая скорость выполнения команд в PIC контроллерах достигается за счет использования двухшинной гарвардской архитектуры, основанной на наборе регистров с разделенными шинами и адресными пространствами для команд и данных.

Микроконтроллеры PIC содержат RISC-процессор с симметричной системой команд, позволяющей выполнять операции с любым регистром, используя произвольный метод адресации. Пользователь может сохранять результат операции в самом регистре-аккумуляторе или во втором регистре, используемом для операции.

RISC (*restricted (reduced) instruction set computer*) - компьютер имеет сокращённый набор команд. Быстродействие в нем увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения - меньшим.

Компания Microchip выпускает несколько основных семейств 8 - разрядных RISC-микроконтроллеров, совместимых снизу вверх по программному коду.

Микроконтроллеры группы PIC16F84X относятся к семейству 8-разрядных КМОП микроконтроллеров группы PIC16CXXX, для которых характерны низкая стоимость, полностью статическая КМОП-технология и высокая производительность. Микроконтроллер PIC16F84A наиболее полно описан в учебной и технической литературе [10], имеет документацию на русском языке [2] и пользуется популярностью у разработчиков устройств. Поэтому и мы начнем с него изучение микроконтроллеров.

1.1. Технические характеристики микроконтроллера PIC16F84A

Описание микроконтроллера надо найти на сайте Microchip и иметь под рукой.

Микроконтроллер PIC16F84A относится к семейству КМОП микроконтроллеров, использует гарвардскую архитектуру с RISC – процессором и имеет следующие основные характеристики:

- используются только 35 простых команд;
- все команды выполняются за один цикл (1 мкс при частоте 4 МГц), кроме команд перехода, которые выполняются за 2 цикла (цикл состоит из четырех тактов генератора);
- рабочая частота от 0 Гц до 4 МГц;
- раздельные шины данных (8 бит) и команд (14 бит);
- 14-битные команды;
- 8-битные данные;
- 1024 x 14 электрически перепрограммируемой программной памяти на кристалле (EEPROM);
- 36 x 8 регистров общего использования;
- 15 специальных аппаратных регистров SFR;
- 64 x 8 электрически перепрограммируемой EEPROM памяти для данных;
- восьмиуровневый аппаратный стек;
- прямая, косвенная и относительная адресация данных и команд;
- четыре источника прерывания.

Периферия и Ввод/Вывод:

- 13 линий ввода-вывода с индивидуальной настройкой;
- входной/выходной ток для управления светодиодами.
- макс. входной ток - 20 мА. ,
- макс. выходной ток - 25 мА.,

Четыре режима возбуждения встроенного генератора выбираются пользователем.

1.2. Особенности архитектуры PIC16F84A

Архитектура микроконтроллера PIC16F84A показана на рис.1.1.

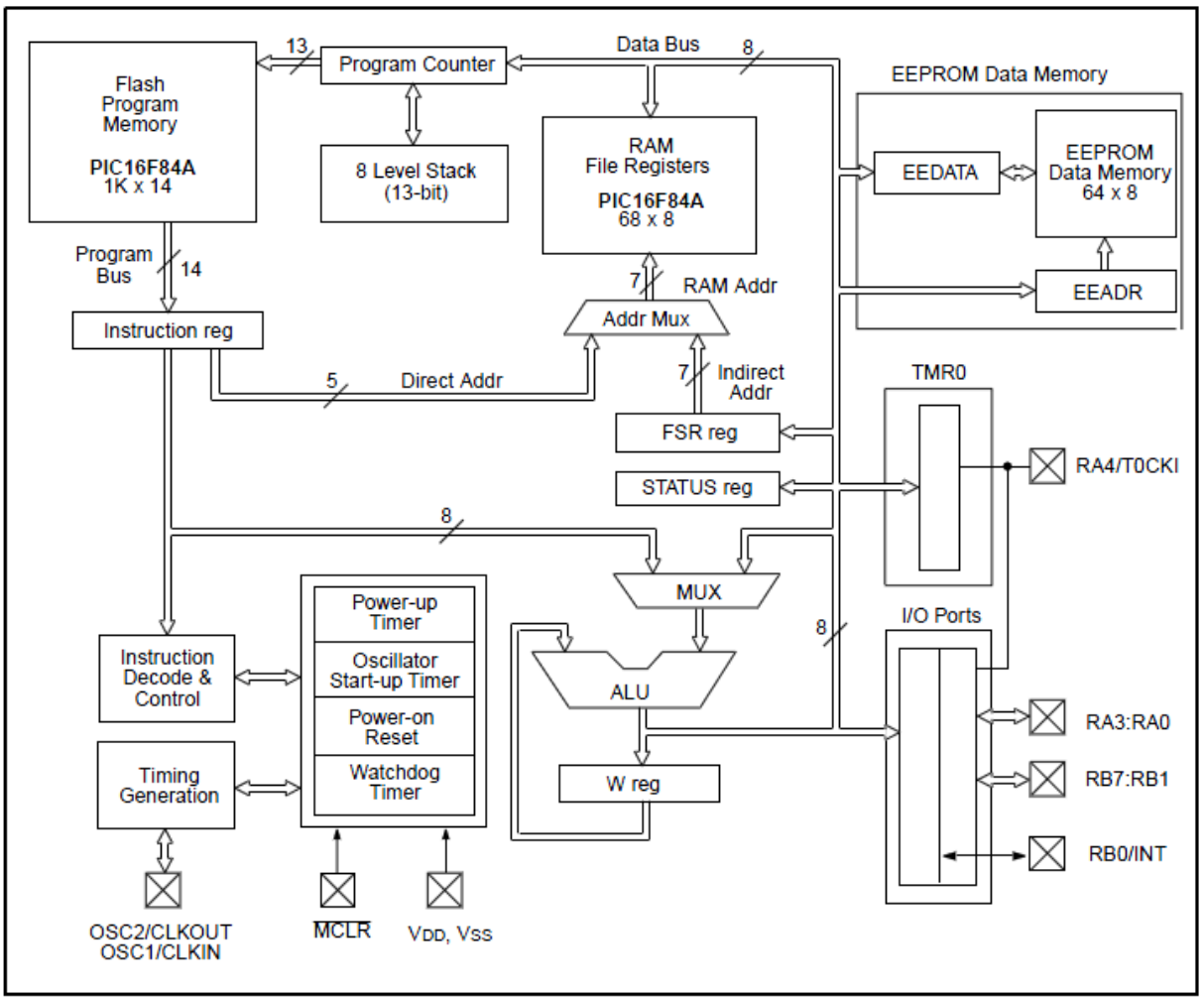


Рис.1.1. Архитектура микроконтроллера PIC16F84

Архитектура основана на концепции раздельных шин и областей памяти для данных и для команд (Гарвардская архитектура). Шина данных (Data bus), память данных (EEPROM Data Memory) (РПЗУ) и регистровое ОЗУ данных (RAM) - имеют ширину 8 бит, а программная шина (Program bus) и программная память (ПЗУ) (Flash Program Memory) имеют ширину 14 бит. Такая концепция обеспечивает простую, но мощную систему

команд, разработанную так, что битовые, байтовые и регистровые операции работают с высокой скоростью и с перекрытием по времени выборок команд и циклов выполнения. 14 - битовая ширина программной памяти обеспечивает выборку 14-битовой команды в один цикл. Двухступенчатый конвейер, использующий 8-ми уровневый стек, обеспечивает одновременную выборку и исполнение команды. Все команды выполняются за один цикл, исключая команды переходов. В PIC16F84A программная память объемом 1К х 14 расположена внутри кристалла. Исполняемая программа может находиться только во встроенном ПЗУ.

Арифметико-логическое устройство АЛУ выполняет действия в 8-битными данными. Адресный мультиплексор (MUX) допускает прямую и косвенную адресацию ОЗУ данных.

Периферия включает в себя 8-битный таймер/счетчик с 8-битным программируемым предварительным делителем (фактически 16 - битный таймер), 13 линий двунаправленного ввода/вывода и другие модули.

1.3. Память

Память в микроконтроллерах (далее МК) бывает трёх типов: память программы, оперативная память, энергонезависимая память.

В *память программы* (Flash Program Memory) загружаются строчки текста программы. Загруженный текст программы не изменяется и не пропадает после выключения питания МК. Размер памяти программ определяется типом используемого МК.

Для PIC16F84A размер памяти программ составляет 1024 строчки.

Оперативная память (RAM) используется для обращения к ней из текста программы и загрузки изменяющихся оперативных данных. При включении МК содержимое ячеек оперативной памяти неизвестно. Данные (числа) загружаются в оперативную память в ходе выполнения программы. Оперативная память МК используется для временного хранения данных. Размер

оперативной памяти относительно памяти программ гораздо меньше. В PIC16F84A оперативная память состоит из 36 ячеек.

Энергонезависимая память (EEPROM) содержит данные, которые при выключение питания МК не пропадают. Содержимое энергонезависимой памяти можно определить в процессе написания программы, а также содержимое может измениться в ходе выполнения программы в МК. В PIC16F84A энергонезависимая память состоит из 64 ячеек.

1.4. Регистры

Память RAM и EEPROM состоит из ячеек, называемых регистрами. Регистр – это устройство, предназначенное для записи, хранения и (или) сдвига цифровой информации, представленной в виде многоразрядного двоичного кода. Память программ может быть организована иначе. В PIC16F84A регистры восьмиразрядные. В регистр может быть записано одно положительное число от 0 до 255.

1.5. Системы счисления

Математика МК использует *двоичную* (бинарную) систему счисления.

Десятичное число 255 в двоичной системе выглядит как восемь единиц `11111111`, а двоичное число 0 – как восемь нулей `00000000`. Обозначение нуля одним символом `0` применяют в любой системе счисления. Таким образом, в регистр записывается не десятичное число, а двоичное.

Двоичные восьмибитные числа называют байтами.

1 БАЙТ = 8 БИТ.

Таким образом, 1 байт представляет собой последовательность нулей и единиц (или набор битов), например, `11011000` , где количество нулей и единиц равно восьми.

Нумерация битов идёт справа налево от нуля до семи. В нашем примере нулевой бит равен нулю, а седьмой бит равен единице. Эта система счисления называется бинарной. В ней используются две цифры `0` и `1`. Восьмибитные числа могут создать 255 комбинаций, т.е. любое число на интервале от 0 до

255. Таким образом, в регистр мы можем записать десятичное число от 0 до 255.

Каждый регистр имеет свой порядковый номер – адрес регистра. Адрес регистра обозначают числом из *шестнадцатеричной системы* счисления, например, 1A.

Шестнадцатеричное число представляет собой комбинацию 16 символов: 0, 1, 2, 3, 4, 5, 6, 7,8, 9, A, B, C, D, E, F. В нашем случае 1A – это десятичное число 26. Перевод чисел из одной системы в другую несложно сделать в стандартном калькуляторе Windows (в настройках калькулятора выбрать вид "инженерный") или используя конвертер BinHexDec.

Для записи исходных данных, констант в программах используют десятичную систему счисления.

В приложении 1 дана таблица чисел, отображаемых в одном байте.

1.6. Формат записи чисел

Чтобы различать, в какой системе счисления задано число, используют следующий формат записи. Десятичное число 26 записывают так:

- в десятичной системе: D26 или d26 или .26 (точка перед числом);
- в двоичной системе: B00011010 или B11010 или b11010;
- в шестнадцатиричной системе: 0x1A или 1Ah.

Все эти примеры записи используются при написании текста программ в MPLAB (об этом позже). Для данных понятнее использовать десятичный формат. Однако, запись .26 понимает MPLAB, но не понимает TINA, а запись d26 – наоборот. Поэтому мы для надежности будем записывать данные тоже в шестнадцатиричной системе.

Адреса регистров традиционно пишут только в шестнадцатеричной форме. При работе с битами регистров нагляднее использовать двоичную запись.

Необходимо отметить, что от нуля до девяти включительно 16-ричная и 10-тичная системы исчисления одинаковы в

написании. Если записано число без атрибутов (B,H,D и "точка"), то по умолчанию оно будет считаться 16-ричным.

1.7. Организация памяти программ и стека

Программный счетчик в PIC16F84A имеет ширину 13 бит и способен адресовать 8Кx14бит объема программной памяти. Однако, физически на кристалле имеется только 1Кx14 памяти (адреса 0000h-03FFh). Обращение к адресам выше 3FFh фактически есть адресация в тот же первый килобайт.

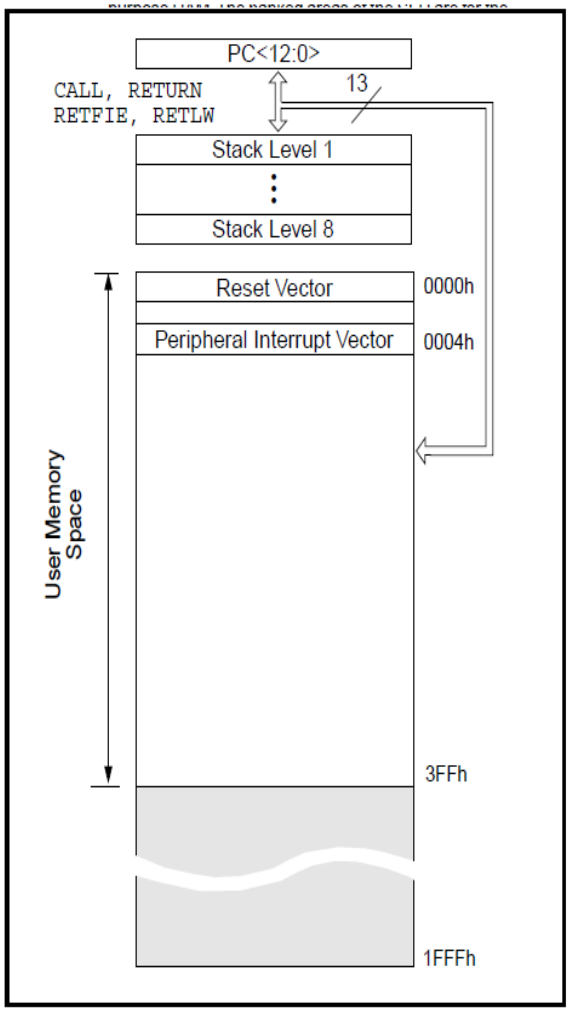


Рис.1.2. Память программ

Организация памяти программ и стека показана на рис.1.2.

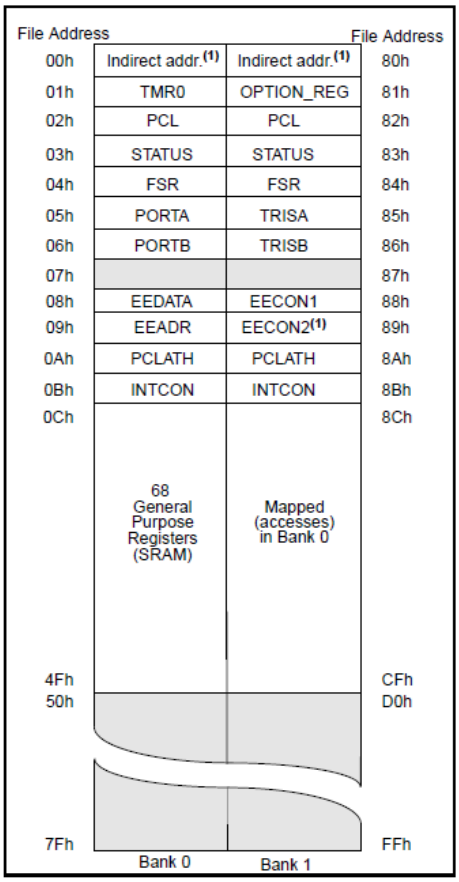
В памяти есть выделенные адреса. Вектор сброса находится по адресу 0000h, вектор прерывания находится по адресу 0004h. По адресу 0004h располагается подпрограмма идентификации и обработки прерываний, а по адресу 0000h – команда перехода на метку, расположенную за подпрограммой обработки прерываний.

1.8. Организация памяти данных

Память данных разбита на две области (рис.1.3). Первые 12 адресов занимают регистры специального назначения (SFR), вторая область – регистры общего назначения (GPR). Область регистров SFR управляет работой МК.

Обе области разбиты на банки 0 и 1. Банк 0 выбирается обнулением бита RP0 регистра статуса (STATUS). Установка RP0 в единицу выбирает банк 1. Каждый банк имеет размер 128 байт. Однако для PIC16F84A память данных существует только до адреса 04Fh.

Некоторые регистры специального назначения продублированы в обоих банках, а некоторые расположены только в одном банке.



Регистры с адресами 0Ch-4Fh могут использоваться как регистры общего назначения, которые представляют собой статическое ОЗУ. Адреса регистров общего назначения банка 1 отображаются на банк 0. Следовательно, когда установлен банк 1, то обращение к адресам 9Ch-CFh фактически адресует банк 0.

К ячейкам ОЗУ можно адресоваться прямо, используя абсолютный адрес каждого регистра, или косвенно, через регистр указатель FSR.

Рис.1.3. Память данных

1.9. Регистры специального назначения

В регистрах специального назначения содержится служебная информация, определяющая настройки работы МК. Под настройками понимается запись восьмибитных чисел в регистры специального назначения, где каждый бит в этих числах определяет ту или иную настройку. Регистры специального назначения находятся в области оперативной памяти.

Регистры специального назначения имеют жестко определенные адреса и регламентированные наименования, которые определены в документации для того или иного МК. Как правило, в разных МК адреса регистров специального назначения совпадают, что подтверждает простоту переноса программ из одних МК в другие.

Регистр статуса (STATUS) содержит признаки операций АЛУ (арифметические флаги), состояние контроллера при сбросе и биты выбора страниц для памяти данных.

Назначение битов регистра STATUS приведено с таблице 1.1.

Таблица 1.1

Регистр STATUS Адрес 03h/83h

Бит 7	IRP	Бит выбора группы банков: 0 – банк 0,1 (000h -0FFh); 1 – банк 2,3 (100h -1FFh).
Биты 6,5	RP1:RP0	Биты выбора банка в пределах группы банков: 00–банк 0 (000h – 07Fh); 01-банк 1 (080h – 0FFh); 10-банк 2 (100h – 17Fh); 11-банк 3 (180h – 1FFh).
Бит 4	-TO	Флаг переполнения сторожевого таймера WDT: 1 - после сброса по включению питания (POR) или выполнения команд CLRWDT, SLEEP; 0 - после переполнения WDT.
Бит 3	-PD	Флаг выключения питания: 1 - после сброса по включению питания (POR) или выполнения команды CLRWDT; 0 - после выполнения команды SLEEP.

Бит 2	Z	Флаг нулевого результата: 1-нулевой результат выполнения операции; 0-результат выполнения операции, отличный от нуля.
Бит 1	DC	Флаг переноса-заема (работа с младшим полубайтом): (для команд ADDWF, ADDLW,SUBWF, SUBLW): 1-был перенос их младшего полубайта; 0-не было переноса из младшего полубайта.
Бит 0	C	Флаг переноса-заема (работа с байтом): 1-был перенос из байта; 0-не было переноса из байта.

Регистр STATUS доступен для любой команды. Однако, если регистр STATUS является регистром назначения для команды, влияющей на биты Z, DC или C, то запись в эти три бита запрещается. Биты – TO и –PD устанавливаются аппаратно и не могут быть изменены программно. Для изменения регистра статуса рекомендуется использовать команды битовой установки BCF, BSF, MOVWF, которые не изменяют остальные биты статуса.

Регистр конфигурации OPTION_REG

Регистр конфигурации (OPTION) является доступным регистром по чтению и записи, который содержит управляющие биты для конфигурации предварительного делителя (предделителя), внешних прерываний, таймера, а также подтягивающих резисторов «pull-up» на выводах PORTB. Назначение битов регистра показано в таблице 1.2.

Регистр OPTION_REG Адрес 81h

Таблица 1.2

Бит 7	-RPBU	Включение подтягивающих резисторов порта B: 1 – подтягивающие резисторы отключены; 0 – подтягивающие резисторы включены.

Бит 6	INTEDG	Выбор активного фронта сигнала на входе внешнего прерывания INT: 1 – прерывание по переднему фронту (0/1); 0 – прерывание по заднему фронту (1/0).
Бит 5	TOCS	Выбор такта для TMR0: 1 – внешний такт с вывода RA4/T0CKI; 0 – внутренний такт CLKOUT.
Бит 4	TOSE	Выбор фронта приращения TMR0 при внешнем такте: 1 – приращение при перепаде (на выводе RA4/T0CKI) от 1 к 0; 0 – приращение при перепаде (на выводе RA4/T0CKI) от 0 к 1.
Бит 3	PSA	Выбор способа включения предделителя: 1 – предделитель включен после сторожевого таймера WDT; 0 – предделитель включен перед TMR0.
Биты 2 - 0	PS2 PS1 PS0	Установка коэффициента деления предделителя.

Коэффициенты деления предделителя устанавливаются битами 2-0 в соответствии с таблицей 1.3.

В том случае, когда предделитель обслуживает сторожевой таймер WDT, таймеру TMR0 назначают коэффициент деления 1:1.

Таблица 1.3

Коэффициенты деления предделителя

Значение	Для Timer0	Для WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4

011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Регистр условий прерываний INTCON

Регистр INTCON является доступным по чтению и записи регистром, который содержит биты доступа для всех источников прерываний. Назначение битов регистра приведено в таблице 1.4.

Бит разрешения всех прерываний GIE устанавливается автоматически при следующих условиях:

- по включению питания;
- по внешнему сигналу –MCLR;
- по внешнему сигналу –MCLR в режиме SLEEP;
- по окончанию задержки таймера WDT при нормальной работе;
- по окончанию задержки таймера WDT в режиме SLEEP.

Бит GIE обнуляется при сбросе. Когда начинает обрабатываться прерывание, бит GIE обнуляется, чтобы запретить дальнейшие прерывания, адрес возврата посылается в стек, а в программный счетчик загружается адрес 0004h. Время реакции на прерывание для внешних событий, таких как прерывание от ножки INT или порта B, составляет приблизительно пять циклов. Это на один цикл меньше, чем для внутренних событий, таких как прерывание по переполнению от таймера Timer0.

Время реакции всегда одинаковое. В подпрограмме обработки прерывания источник прерывания может быть определен по соответствующему биту в регистре флагов.

Этот флаг-бит должен быть программно сброшен внутри подпрограммы. Флаги запросов прерываний не зависят от соответствующих маскирующих битов и бита общего маскирования GIE.

Команда возврата из прерывания RETFIE завершает прерывающую подпрограмму и устанавливает бит GIE, чтобы опять разрешить прерывания.

Прерывание INT может вывести микроконтроллер из режима SLEEP, если перед входом в этот режим бит INTE был установлен в единицу. Состояние бита GIE определяет, будет ли МК переходить в подпрограмму прерывания после выхода из режима SLEEP.

Сброс битов - запросов прерываний должен осуществляться соответствующей программой обработки.

Регистр INTCON Адрес 0Bh/8Bh

Таблица 1.4

Бит 7	GIE	Глобальное разрешение прерываний: 1 – разрешены все немаскированные прерывания; 0 – все прерывания запрещены.
Бит 6	EEIE	Разрешение прерываний по окончанию записи в EEPROM: 1 – прерывание разрешено; 0 – прерывание запрещено.
Бит 5	TOIE	Разрешение прерываний по переполнению TMR0: 1 – прерывание разрешено; 0 – прерывание запрещено.
Бит 4	INTE	Разрешение прерываний по входу RB0/INT: 1 – прерывание разрешено; 0 – прерывание запрещено.
Бит 3	RBIE	Разрешение прерываний по изменению уровня сигналов на входах RB4…RB7: 1 – прерывание разрешено;

		0 – прерывание запрещено.
Бит 2	T0IF	Флаг прерывания по переполнению TMR0 (сбрасывается программно); 1 – произошло переполнение TMR0; 0 – переполнения TMR0 не было.
Бит 1	INTF	Флаг переноса-заема (работа с байтом): 1-был перенос из байта; 0-не было переноса из байта.
Бит 0	RBIF	Флаг прерывания по изменению уровня сигналов на выводах RB4…RB7 (сбрасывается программно); 1 – зафиксировано изменение уровня сигнала на одном из входов RB4…RB7; 0 – не было изменения уровня сигналов ни на одгом их входов RB4…RB7.

1.10. Счетчик команд

Счетчик команд PCL и PCLATH имеет разрядность 13 бит. Младший байт счетчика (PCL) доступен для чтения и записи и находится в регистре 02h. Старший байт счетчика не может быть напрямую записан или считан и берется из регистра PCLATH (PC latch high) с адресом 0Ah. Содержимое PCLATH передается в старший байт счетчика команд, когда он загружается новым значением.

Рис.1.4. Загрузка программного счетчика

В зависимости от того, загружается ли в программный счетчик новое значение во время выполнения команд CALL, GOTO, или в младший байт программного счетчика (PCL) производится запись из АЛУ, старшие биты программного счетчика загружаются из PCLATH разными способами так, как показано на рис.1.4.

1.11. Стек и возврат из подпрограмм

Кристалл PIC16F84 имеет восьмиуровневый аппаратный стек шириной 13 бит.

Область стека не принадлежит ни к программной области, ни к области данных, а указатель стека пользователю недоступен. Текущее значение программного счетчика посылается в стек, когда выполняется команда CALL или производится обработка прерывания. При выполнении процедуры возврата из подпрограммы (команды RETLW , RETFIE или RETURN), в программный счетчик выгружается содержимое стека. Регистр PCLATH (0Ah) не изменяется при операциях со стеком. Стек работает как циклический буфер и допускает 8 загрузок.

1.12. Прямая и косвенная адресация

Когда производится прямая 9-битная адресация, младшие 7 бит берутся как прямой адрес из кода операции, а два бита указателя страниц (RP1,RP0) из регистра статуса (03h) (рис.1.5).

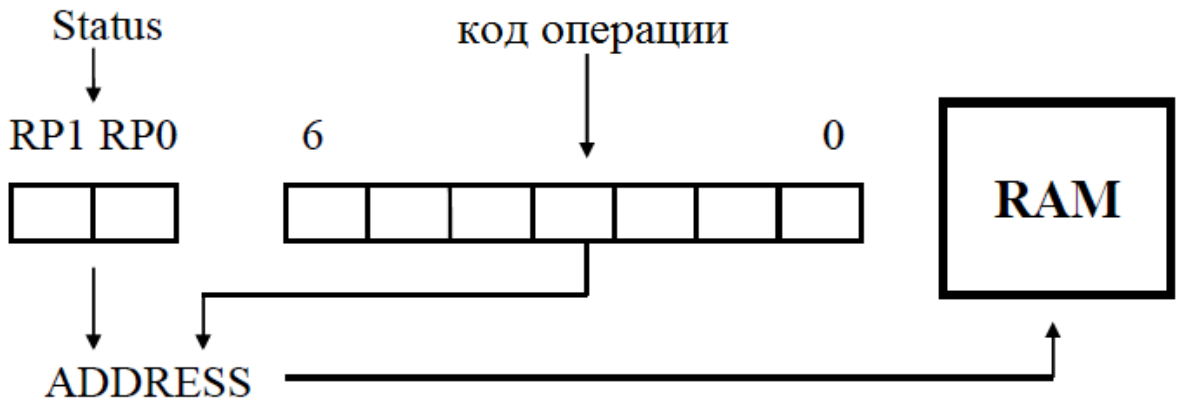


Рис.1.5. Схеме прямой адресации

Признаком косвенной адресации является обращение к регистру INDF.

При косвенной адресации любая команда, которая использует INDF (адрес 00h) в качестве регистра, фактически обращается к указателю косвенной адресации, который хранится в FSR (04h). Чтение косвенным образом самого регистра INDF даст результат 00h. Запись в регистр INDF косвенным образом будет выглядеть как NOP (невыполняемая команда), но биты статуса могут быть изменены.

Необходимый 9-битный адрес формируется объединением содержимого 8-битного FSR регистра и бита IRP из регистра статуса (рис.1.6).

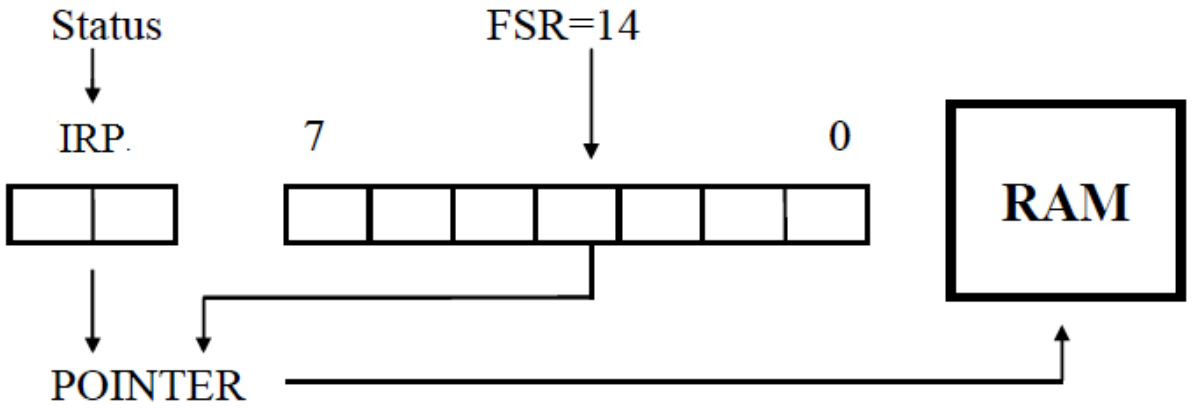


Рис.1.6. Схема косвенной адресации

1.13. Порты ввода-вывода

Кристалл имеет два порта: 5-ти битный порт A и 8-ми битный порт B с побитовой индивидуальной настройкой на ввод или на вывод.

Порт A - это порт шириной 5 бит, соответствующие ножки кристалла RA <4:0>. Линии RA<3:0> двунаправленные, а линия RA4 - выход с открытым стоком. Адрес регистра порта A 05h. Относящийся к порту A управляющий регистр TRISA расположен на первой странице регистров по адресу 85h. TRISA<4:0> - это регистр шириной 5 бит. Если бит управляющего TRISA регистра имеет значение единица, то соответствующая линия будет устанавливаться на ввод. Ноль переключает линию на вывод и одновременно выводит на нее содержимое соответствующего регистра защелки.

Порт B - это двунаправленный порт, шириной в восемь бит (адрес регистра 06h). Относящийся к порту B управляющий регистр TRISB расположен на первой странице регистров по адресу 86h. Если бит управляющего TRISB регистра имеет значение единица, то соответствующая линия будет устанавливаться на ввод. Ноль переключает линию на вывод и одновременно выводит на нее содержимое соответствующего регистра защелки. У каждой ножки порта B имеется небольшая активная нагрузка (около 100мкА) на линию питания. Она автоматически отключается, если эта ножка запрограммирована как вывод. Более того, управляющий бит RPBU OPTION_REG <7> может отключить (RPBU =1) все нагрузки. Сброс при включении питания также отключает все нагрузки.

Четыре линии порта B (RB<7:4>) имеют способность вызвать прерывание при изменении значения сигнала на любой из них. Если эти линии настроены на ввод, то они опрашиваются и защелкиваются в цикле чтения Q1. Новая величина входного сигнала сравнивается со старой в каждом командном цикле. При несовпадении значения сигнала на ножке и в защелке, генерируется высокий уровень. Выходы детекторов “несовпадений” RB4, RB5, RB6, RB7 объединяются по ИЛИ и генерируют прерывание RBIF (запоминаемое в INTCON<0>). Любая линия, настроенная как вывод, не участвует в этом сравнении. Прерывание может вывести кристалл из режима SLEEP.

В подпрограмме обработки прерывания следует сбросить запрос прерывания одним из следующих способов:

1) Запретить прерывания при помощи обнуления бита RBIE INTCON<3>.

2) Прочитать порт B. Это завершит состояние сравнения.

3) Обнулить бит RBIF INTCON<0>.

Прерывание по несовпадению и программно устанавливаемые внутренние активные нагрузки на этих четырех

линиях могут обеспечить простой интерфейс например с клавиатурой, с выходом из режима SLEEP по нажатию клавиш.

Ножка RB0 совмещена с входом внешнего прерывания INT.

1.14. Модуль таймера и регистр таймера

Структура таймера Timer0 с использованием предделителя показана на рис.1.7. Режим таймера выбирается путем сбрасывания в ноль бита T0CS, который находится в регистре OPTION_REG. В режиме таймера регистр TMR0 будет инкрементироваться от внутреннего источника частоты каждый командный цикл (без предделителя). После записи информации в TMR0, инкрементирование его начнется после двух командных циклов. Такое происходит со всеми командами, которые производят запись или чтение-модификацию-запись TMR0 (например: MOVF f1, CLRF f1). Избежать этого можно при помощи записи в TMR0 скорректированного значения. Если TMR0 нужно проверить на равенство нулю без остановки счета, следует использовать инструкцию MOVF f1,W.

Режим счетчика выбирается путем установки в единицу бита T0CS, который находится в регистре OPTION_REG. В этом режиме TMR0 будет инкрементироваться либо положительным, либо отрицательным фронтом на ножке RA4/T0CKI от внешнего источника. Направление фронта определяется управляющим битом T0SE в регистре OPTION_REG. При T0SE=0 будет выбран передний фронт. Предделитель может быть использован или совместно с TMR0, или с Watchdog таймером. Вариант подключения делителя контролируется битом PSA в регистре OPTION_REG. При PSA=0 делитель будет подсоединен к TMR0.

Содержимое делителя программе недоступно.

Коэффициент деления программируется. Прерывание по TMR0 вырабатывается тогда, когда происходит переполнение TMR0 таймера/счетчика при переходе от FFh к 00h. Тогда устанавливается бит запроса T0IF в регистре INTCON<2>. Данное прерывание можно замаскировать битом T0IE в регистре INTCON<5>. Бит запроса T0IF должен быть сброшен программно при обработке прерывания. Прерывание по TMR0 не

может вывести процессор из SLEEP, так как таймер в этом режиме отключен.

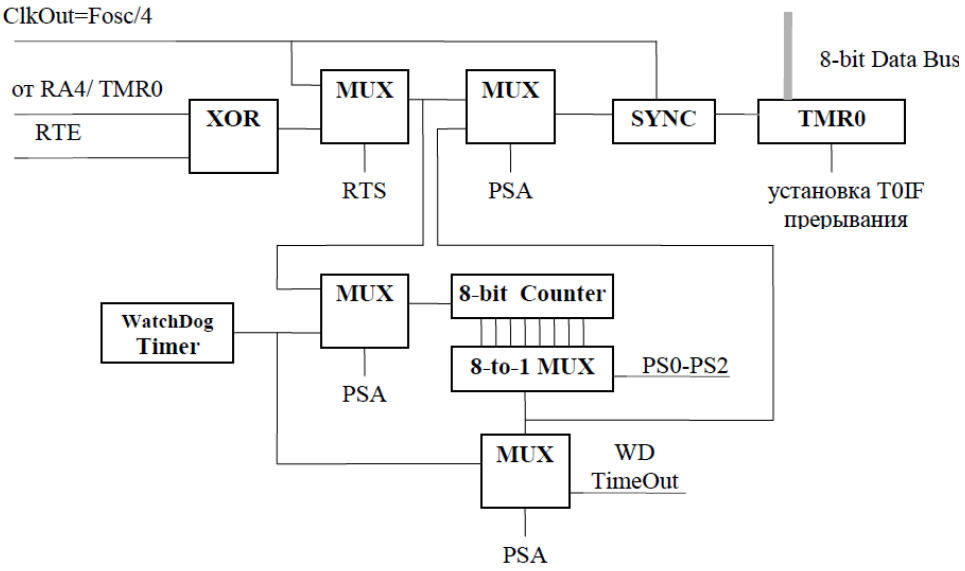


Рис.1.7. Структура таймера Timer0

1.15. Память данных в РПЗУ (EEPROM)

Память данных EEPROM позволяет прочитать и записать байт информации. При записи байта автоматически стирается предыдущее значение и записываются новые данные (стирание перед записью). Все эти операции производит встроенный автомат записи EEPROM. Содержимое ячеек этой памяти сохраняется при выключении питания.

Кристалл PIC16F84A имеет память данных 64x8 EEPROM бит, которая позволяет запись и чтение во время нормальной работы (во всем диапазоне питающих напряжений). Эта память не принадлежит области регистров ОЗУ. Доступ к ней осуществляется через два регистра: EEDATA (08h), который содержит в себе восьмибитовые данные для чтения/записи и EEADR (09h), который содержит в себе адрес ячейки к которой идет обращение. Дополнительно имеется два управляющих регистра: EECON1 (88h) и EECON2 (89h).

При считывании данных из памяти EEPROM необходимо записать требуемый адрес в EEADR регистр и затем установить

бит RD EECON1<0> в единицу. Данные появятся в следующем командном цикле в регистре EEDATA и могут быть прочитаны. Данные в регистре EEDATA защелкиваются.

При записи в память EEPROM, необходимо сначала записать требуемый адрес в EEADR регистр и данные в EEDATA регистр. Затем выполнить специальную последовательность команд, производящую непосредственную запись:

```
movlw       55h
movwf       EECON2
movlw       AAh
movwf       EECON2
bsf         EECON1,WR    ;установить WR бит, начать
                         ;запись
```

Во время выполнения этого участка программы, все прерывания должны быть запрещены для точного выполнения временной диаграммы. Время записи – примерно 10мс. Фактическое время записи будет изменяться в зависимости от напряжения, температуры и индивидуальных свойств кристалла. В конце записи бит WR автоматически обнуляется, а флаг завершения записи EEIF, он же запрос на прерывание, устанавливается.

Для предотвращения случайных записей в память данных предусмотрен специальный бит WREN в регистре EECON1. Рекомендуется держать бит WREN выключенным, кроме тех случаев, когда нужно обновить память данных. Более того, кодовые сегменты, которые устанавливают бит WREN и те, которые выполняют запись должны храниться на различных адресах, чтобы избежать случайного выполнения их обоих при сбое программы.

Регистр EECON1 (адрес 88h) - это управляющий регистр шириной пять бит. Младшие пять бит физически существуют, а старшие три бита читаются всегда как `0`.

Регистр EECON1 Адрес 88h

-	-	-	EEIF	WRERR	WREN	WR	RD
7	6	5	4	3	2	1	0

RD - Бит чтения.

RD =1 : Запускает чтение памяти данных EEPROM. Чтение занимает один цикл. Устанавливается программно. Обнуляется аппаратно.

WR - Бит записи.

WR =1: Запускает запись в память данных EEPROM. Устанавливается программно. Обнуляется аппаратно.

WREN - бит разрешения записи в память данных EEPROM.

WREN = 1: Разрешена запись.

WREN = 0: Запрещена запись. После включения питания WREN обнуляется.

WRERR - флаг ошибки записи устанавливается, когда процесс записи прерывается сигналом сброса /MCLR (во время обычного режима или режима SLEEP) или сигналом сброса от WDT таймера (во время обычного режима). Рекомендуем проверять этот флаг WRERR и при необходимости производить перезапись данных. Данные и адрес сохраняются в регистрах EEDATA и EEADR.

EEIF - флаг завершения записи.

EEIF = 1: флаг устанавливается, когда завершена запись. Соответствующий бит разрешения прерывания - EEIE устанавливается в регистре INTCON.

Регистр EECON2 не является физическим регистром. Он используется исключительно при организации записи данных в EEPROM. Чтение регистра EECON2 дает нули.

1.16. Алгоритм сброса при включении питания

Кристалл PIC16F84A имеет встроенный детектор включения питания. Таймер запуска начинает счет выдержки времени после того, как напряжение питания пересекло уровень около 1,2...1,8 Вольт. По истечении выдержки около 72мс считается, что напряжение достигло номинала и запускается другой таймер выдержка на стабилизацию кварцевого генератора. Программируемый бит конфигурации позволяет

разрешать или запрещать выдержку от встроенного таймера запуска.

Таймер на стабилизацию генератора отсчитывает 1024 импульса от начавшего работу генератора. Считается, что кварцевый генератор за это время вышел на режим. При использовании RC генераторов выдержка на стабилизацию не производится.

Затем включается таймер ожидания внешнего сброса MCLR. Это необходимо для тех случаев, когда требуется синхронно запустить в работу несколько PIC контроллеров через общий для всех сигнал MCLR . Если такого сигнала не поступает, то через время Tost вырабатывается внутренний сигнал сброса и контроллер начинает ход по программе. Время Tost программируется битами конфигурации в EEPROM.

1.17. Сторожевой (Watchdog) таймер

Сторожевой таймер WDT предназначен для предотвращения катастрофических последствий от случайных сбоев программы. Он также может быть использован в приложениях, связанных со счетом времени, например, в детекторе пропущенных импульсов. Идея использования сторожевого таймера состоит в регулярном его сбрасывании под управлением программы или внешнего воздействия до того, как закончится его выдержка времени и не произойдет сброс процессора. Если программа работает нормально, то команда сброса сторожевого таймера CLRWDT должна регулярно выполняться, предохраняя поцессор от сброса. Если же микропроцессор случайно вышел за пределы программы (например, от сильной помехи по цепи питания) либо зациклился на каком-либо участке программы, команда сброса сторожевого таймера скорее всего не будет выполнена в течение достаточного времени, и произойдет полный сброс процессора, инициализирующий все регистры и приводящий систему в рабочее состояние.

Watchdog таймер представляет собой полностью независимый встроенный RC генератор, который не требует никаких внешних цепей. Он будет работать, даже если основной

генератор остановлен, как это бывает при исполнении команды SLEEP.

Таймер вырабатывает сигнал сброса. Выработка таких сбросов может быть запрещена путем записи нуля в специальный бит конфигурации WDTE. Эту операцию производят на этапе прожига микросхем.

Выдержка времени WDT

Номинальная выдержка WDT составляет 18мс (без использования делителя).

Она зависит от температуры, напряжения питания, от особенностей типов микросхем. Если требуются большие задержки, то к WDT может быть подключен встроенный делитель с коэффициентом деления до 1:128; который программируется путем записи в регистр OPTION_REG. Здесь могут быть реализованы выдержки до 2.5 секунд.

Команды “CLRWDT” и “SLEEP” обнуляют WDT и делитель, если он подключен к WDT. Это запускает выдержку времени сначала и предотвращает на некоторое время выработку сигнала сброса. Если сигнал сброса от WDT все же произошел, то одновременно обнуляется бит “TO” в регистре статуса.

В приложениях с высоким уровнем помех, содержимое регистра OPTION_REG подвержено сбою. Поэтому регистр OPTION_REG должен обновляться через равные промежутки времени.

1.18. Типы генераторов

Кристаллы PIC16F84A могут работать с четырьмя типами встроенных генераторов:

- XT кварцевый резонатор;
- HS высокочастотный кварцевый резонатор;
- LP микропотребляющий кварцевый резонатор;
- RC - RC цепочка

Задание типа используемого тактового генератора осуществляется в процессе программирования микросхемы. В случае задания вариантов XT, HS и LP к микросхеме

подключается кварцевый или керамический резонатор либо внешний источник тактовой частоты, а в случае задания варианта RC - резистор и конденсатор.

Пользователь может запрограммировать два конфигурационных бита (FOSC1 и FOSC0) для выбора одного из четырех режимов: RC, LP, XT, HS.

Кристаллы PIC16... могут также тактироваться и от внешних источников.

1.19. Биты конфигурации

Кристалл PIC16F84A имеет пять битов конфигурации, которые хранятся в EEPROM по адресу 2007h и устанавливаются на этапе программирования кристалла. Эти биты могут быть запрограммированы (читаются как `0`) или оставлены незапрограммированными (читаются как `1`) для выбора нужного варианта конфигурации. Слово конфигурации содержит 14 бит (Таблица 1.5)

Слово конфигурации Таблица 1.5

Биты 13-4	CP	Биты защиты программ:. 1=защита памяти программ выключена 0=все программы защищены.
Бит 3	----- PWRTE	Бит разрешения выдержки времени после вкдючения питания: 1=выдержка есть; 0=выдержки нет.
Бит 2	WDTE	Бит разрешения работы WatchDog Timer: 1-разрешен; 0-запрещен.
Биты <1:0>	FOSC1:FOSC0	Биты выбора типа генератора: 00-LP генератор; 01-XT генератор; 10-HS генератор; 11-RC генератор.

1.20. Система команд микроконтроллера PIC16F84A

Простая и эффективная система команд включает в себя всего 35 команд.

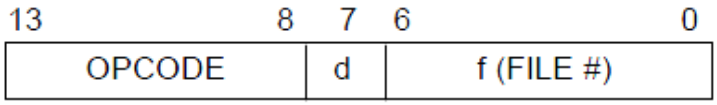
Каждая команда PIC16F84A - это 14-битовое слово, которое разделено по смыслу на следующие части: код операции, поле для одного и более операндов, которые могут участвовать или нет в этой команде.

Система команд PIC16F84A включает в себя:

- байт-ориентированные команды;
- бит-ориентированные команды и операции с константами;
- команды передачи управления.

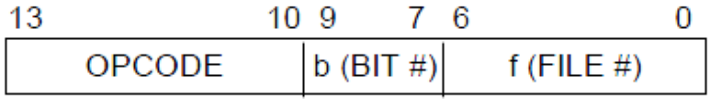
Основные форматы команд показаны ниже.

Команды работы с байтами



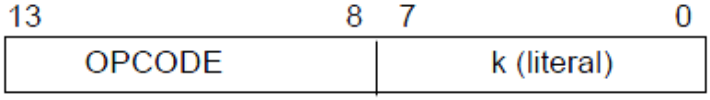
d = 0 для назначения W
d = 1 для назначения f
f = 7-битовый адрес регистра

Команды работы с битами



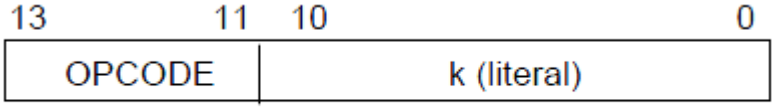
b = 3-х разрядный номер бита
f = 7-битовый адрес регистра

Команды управления и операций с константами



k = 8-ми разрядная константа

Команды CALL и GOTO



k = 11-ти разрядная константа

Для байт-ориентированных команд “f” обозначает собой регистр, с которым производится действие; “d” - бит определяет, куда положить результат. Если “d” =0, то результат будет помещен в W регистр, при “d”=1 результат будет помещен в “f”, упомянутом в команде.

Байт-ориентированные команды

Мнемокод		Название команды	Флаги	Приме ч.
ADDWF	f,d	Сложение W с f	C,DC,Z	2,3
ANDWF	f,d	Логическое И W и f	Z	2,3
CLRF	f	Сброс регистра f	Z	3
CLRW		Сброс регистра W	Z	
COMF	f,d	Инверсия регистра f	Z	2,3
DECF	f,d	Декремент регистра f	Z	2,3
DECFSZ	f,d	Декремент f, пропустить команду, если 0		2,3
INCF	f,d	Инкремент регистра f	Z	2,3
INCFZ	f,d	Инкремент регистра f, пропустить команду, если 0		2,3
IORWF	f,d	Логическое ИЛИ W и f	Z	2,3
MOVF	f,d	Пересылка регистра f	Z	2,3
MOVWF	f	Пересылка W в f		3
NOP		Холостая команда		
RLF	f,d	Сдвиг f влево через перенос	C	2,3
RRf	f,d	Сдвиг f вправо через перенос	C	2,3
SUBWF	f,d	Вычитание W из f	C,DC,Z	2,3
SWAPF	f,d	Обмен местами тетрад в f		2,3
XORWF	f,d	Исключающее ИЛИ W и f	Z	2,3
ADDLW	k	Сложение константы с W.	C,DC,Z	
ANDLW	k	Логическое И константы и W	Z	
IORLW	k	Логическое ИЛИ константы и W	Z	
SUBLW	k	Вычитание W из константы.	C,DC,Z	
MOVLW	k	Пересылка константы в W		
XORLW	k	Исключающее ИЛИ константы и W	Z	
OPTION		Загрузка W в OPTION_REG регистр		1
TRIS	F	Загрузка TRIS регистра		1

Для бит-ориентированных команд “b” обозначает номер бита, участвующего в команде, а “f” - это регистр , в котором этот бит расположен.

Для команд передачи управления и операций с константами, “k” обозначает восьми или одинадцатибитную константу.

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

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

Один командный цикл состоит из четырех периодов генератора. Таким образом, для генератора с частотой 4 МГц время исполнения командного цикла будет 1мкс.

Бит-ориентированные команды

Мнемокод		Название команды	Флаги	Примеч.
BCF	f,d	Сброс бита в регистре f		2,3
BSF	f,d	Установка бита в регистре f		2,3
BTFSC	f,b	Пропустить команду, если бит равен 0		
BTFSS	f,b	Пропустить команду, если бит равен 1		

Переходы

Мнемокод		Название команды	Флаги	Примеч.
CALL	k	Вызов подпрограммы		
CLRWDT		Сброс Watchdog таймера	TO,PD	
Мнемокод		**Название команды**	**Флаги**	**Примеч.**
GOTO	k	Переход по адресу		
RETLW	k	Возврат из подпрограммы с загрузкой константы в W		
RETFIE		Возврат из прерывания.		
RETURN		Возврат из подпрограммы.		
SLEEP		Переход в режим SLEEP	TO,PD	

Примечание 1:

Команды TRIS и OPTION помещены в перечень команд для совместимости с семейством PIC16C5X. Их использование не рекомендуется. В PIC16F84 регистры TRIS и OPTION доступны для чтения и записи как обычные регистры с номером.

Предупреждаем, что эти команды могут не поддерживаться в дальнейших разработках PIC16CXX.

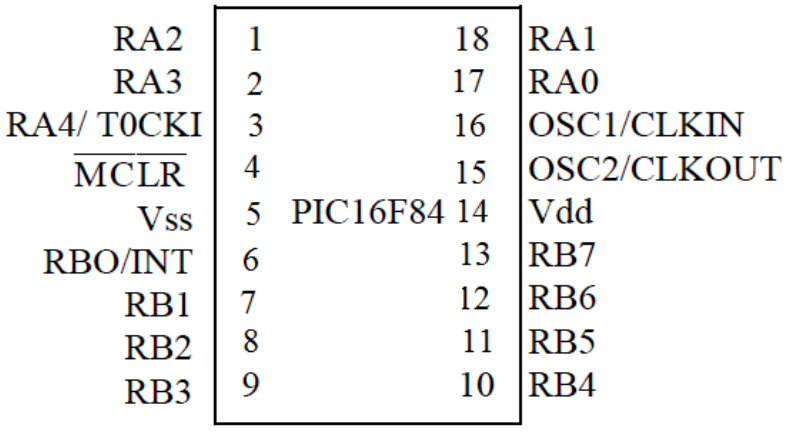
Примечание 2:

Когда модифицируется регистр ввода/вывода, например MOVF 6,1, значение, используемое для модификации считывается непосредственно с ножек кристалла. Если значение защелки вывода для ножки, запрограммированной на вывод равно “1”, но внешний сигнал на этом выводе “0” из-за “навала” снаружи, то будет считываться “0”.

Примечание 3:

Если операндом этой команды является регистр f1 (и, если допустимо, d=1), то делитель, если он подключен к TMR0, будет обнулен.

1.21. Разводка ножек микроконтроллера PIC16F84A



Обозначение	Нормальный режим	Режим записи EEPROM
RA0 – RA3	Двунаправленные линии ввода/вывода. Входные уровни ТТЛ	
RA4/T0CKI	Вход через триггер Шмитта. Ножка порта ввода/вывода с открытым стоком или вход частоты для таймера/счетчика TMR0	

RB0/INT	Двунаправленная линия порта ввода/ вывода или внешний вход прерывания Уровни ТТЛ	
RB1 – RB5	Двунаправленные линии ввода/ вывода. Уровни ТТЛ	
RB6	Двунаправленные линии ввода/ вывода. Уровни ТТЛ.	Вход тактовой частоты для EEPROM
RB7	Двунаправленные линии ввода/ вывода. Уровни ТТЛ.	Вход/выход EEPROM данных.
$\overline{MCLR}$ /Vpp	Низкий уровень на этом входе генерирует сигнал сброса для контроллера. Активный низкий.	Сброс контроллера Для режима EEPROM- подать Vpp.
OSC1/CLKIN	Для подключения кварца,	

Обозначение	Нормальный режим	Режим записи EEPROM
	RC или вход внешней тактовой частоты	
OSC2/CLKOUT	Генератор, выход тактовой частоты в режиме RC генератора, в остальных случаях - для подкл.кварц	
Vdd	Напряжение питания	Напряжение питания
Vss	Общий(земля)	Общий

Контрольные вопросы

1. Что означает название микроконтроллеров PIC ?
2. Какую архитектуру имеют микроконтроллеры PIC и в чем особенности этой архитектуры ?
3. В чем особенности микроконтроллеров с RISC-процессором ?
4. Назовите основные характеристики микроконтроллера PIC16F84A .
5. Особенности архитектуры микроконтроллера PIC16F84A.
6. Какой формат записи чисел применяют при программировании микроконтроллеров ?

7. Расскажите об организации памяти программ и стека PIC16F84A .
8. Организация памяти данных.
9. Назовите регистры специального назначения PIC16F84A.
10. Объясните назначение битов регистра STATUS.
11. Объясните назначение битов регистра OPTION_REG.
12. Объясните назначение битов регистра INTCON.
13. Назначение и работа счетчика команд.
14. Назначение и работа стека.
15. Как осуществляется прямая и косвенная адресация при выполнении программ ?
16. Порты ввода-вывода. Их назначение и особенности работы.
17. Назначение и работа таймера Timer0.
18. Что такое EEPROM ? Порядок записи и чтения данных в EEPROM.
19. Для чего служит вывод -MCLR микроконтроллера ?
20. Назначение и работа сторожевого таймера.
21. Какие типы генераторов можно применять в PIC16F84A ?
22. Для чего нужны биты конфигурации и как их устанавливают ?
23. Система команд микроконтроллера PIC16F84A.
24. Назовите несколько байт-ориентированных команд и объясните их работу.
25. Назовите бит-ориентированные команды и объясните из работу.
26. Назовите команды переходов и объясните их работу.
27. Объясните назначение выводов микроконтроллера PIC16F84A.

Глава 2. МОДЕЛИРОВАНИЕ И ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ В ПРОГРАММНОЙ СРЕДЕ TINA

2.1. Краткие сведения о программе TINA

Программная среда TINA (TINA Design Suite) является мощным и удобным инструментом для анализа, проектирования и моделирования в реальном времени аналоговых, цифровых и микропроцессорных схем. Эта программная среда разработана компанией DesignSoft с участием компании Texas Instruments и позволяет проводить исследование схем при вариации параметров, оптимизацию, выполнять частотный и спектральный анализ, исследовать переходные характеристики, выполнять радиочастотный анализ, программировать и моделировать микроконтроллеры. Результатом проектирования может быть печатная плата устройства, так как TINA имеет программу проектирования печатных плат (PCB Design).

Бесплатная студенческая версия TINA - TI v.9 весьма удобна для изучения электротехники, электроники и схемотехники, но, к сожалению, не содержит микроконтроллеров. В этом учебном пособии мы использовали программу TINA v9 (далее TINA). Последняя разработка компании DesignSoft TINA v.10 в разделе микроконтроллеров отличается дополнительной возможностью программирования на языке Си. Представляет интерес облачная on-line версия TINACloud, удобная для работы на планшетах со схемами средней сложности.

Подробно программу TINA-TI и использование разнообразных инструментов студенты изучают в дисциплине «Электротехника, электроника и схемотехника» [8]. Здесь мы приводим основные сведения, которые необходимы для моделирования и программирования микроконтроллеров.

2.2. Интерфейс программы

Рабочее окно программы TINA содержит следующие элементы (рис.2.1):

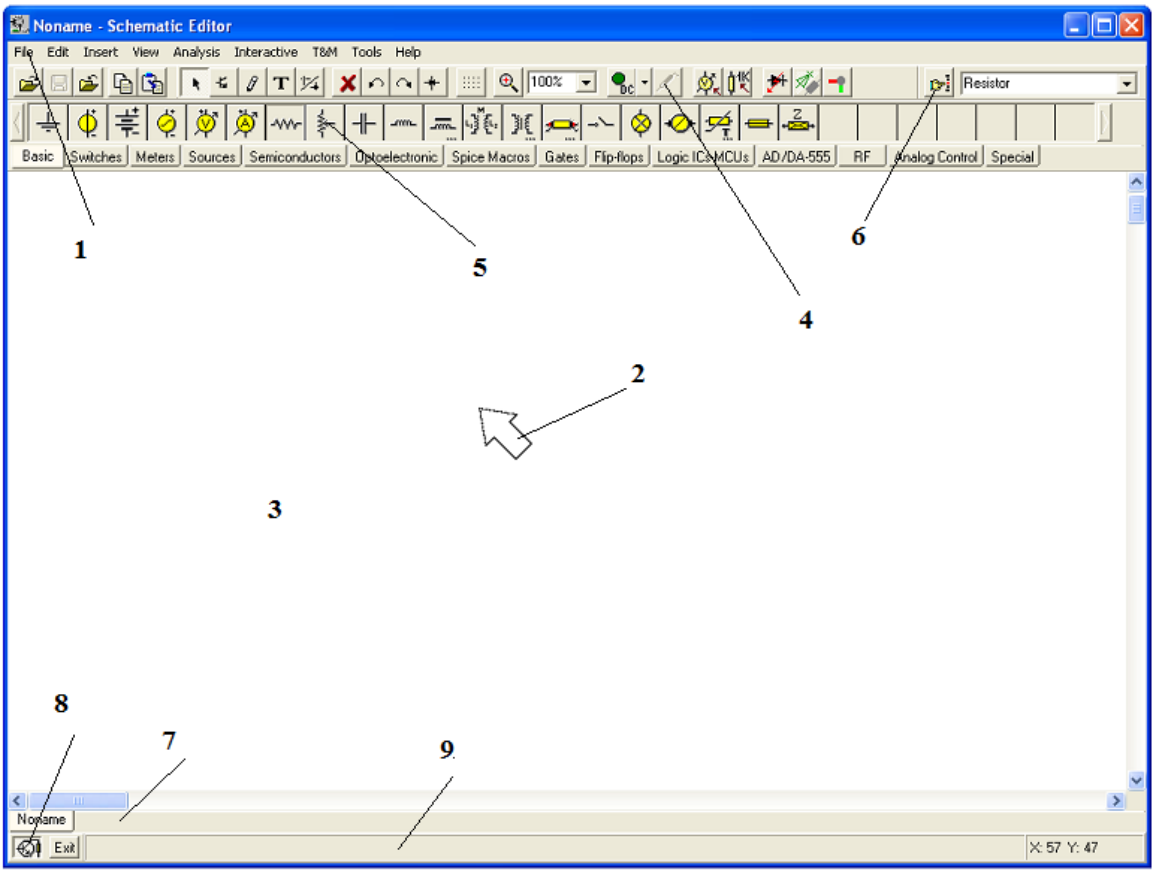


Рис. 2.1 Окно программы TINA

1. Строка меню (рис.2.2) содержит вкладки *File, Edit, Insert, View, Analysis, Interactive, T&M, Tools, Help*. Поясним содержание некоторых вкладок.

На вкаладке *View* можно задать отображение значений и этикеток компонентов, сетки, изменять масштаб, включать трехмерное отображение.

На вкладке *Analysis* мы задаем режимы моделирования схемы: на постоянном токе, на переменном токе, переходной процесс, Фурье-анализ, различные виды анализа цифровых схем, символьный анализ с получением аналитического описания цепи, анализ шумовых характеристик, оптимизация, включение отладчика микроконтроллеров и т.д.

На вкладке *Interactive* имеется кнопка *Start* и задается режим интерактивного анализа. Интерактивный режим позволяет испытать схему в «реальной ситуации», используя

интерактивные элементы управления (клавиатуры, выключатели) и наблюдая на происходящим на дисплеях и индикаторах.

Интерактивный режим также очень полезен для образовательных и демонстрационных целей, для настройки схем с микроконтроллерами, которые нельзя проверить в статике. Сначала надо выбрать, какой вид интерактивный режим требуется (*DC, AC, TR, или VHDL*) с помощью кнопки  , а затем нажать на кнопку *Start* и проводить моделирование.

На вкладке *T&M* представлены различные приборы, которые можно использовать в моделировании.

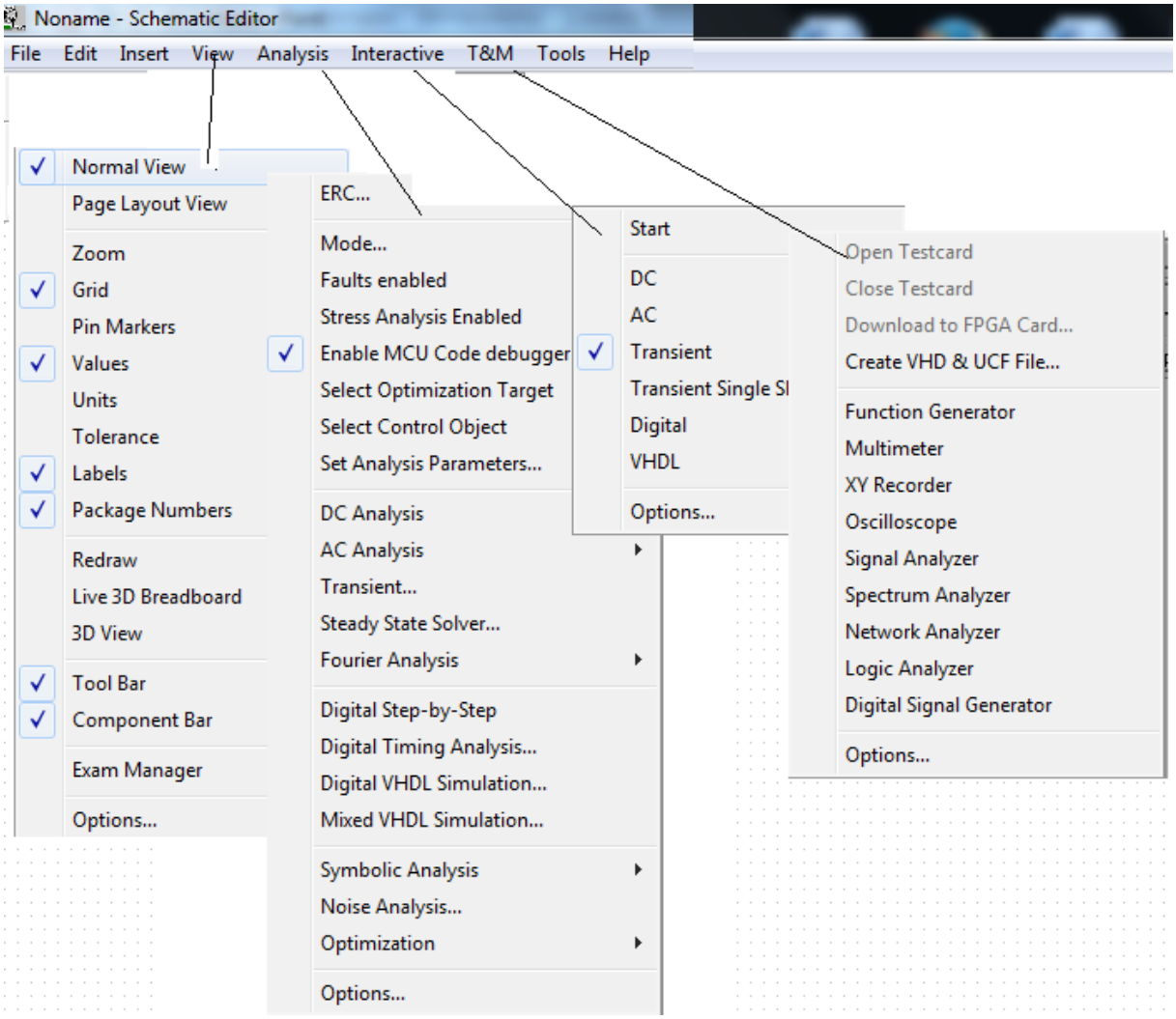


Рис.2.2. Вкладки строки меню

2. *Курсор*, указатель (стрелка). Курсор можно двигать мышью и выполнять следующие операции:

выделять область схемы;
переносить на схему символы компонентов;
определять конец проводника и соединять компоненты;
изменять конфигурацию проводников;
изменять конфигурацию схем;
разрывать соединения в схемах;
увеличивать схему, используя кнопку масштаба.

3. Окно схемы, в котором собирают схему цепи, включают приборы, проводят измерения и т.п. Операция *View/Greed On/Off* создает или удаляет сетку с рабочего окна. Операция *View/Pin Markers On/Off* показывает или удаляет контакты компонентов.

4. *Панель инструментов* позволяет выбрать многие команды редактирования: выделение, масштаб, проволочное соединение и т.д. Рассмотрим наиболее важные команды на панели инструментов. Панель инструментов содержит меню команд (рис.2.3).

Рис.2.3. Меню команд

Кроме обычных команд (открытие файла, сохранение и т.п.) в меню команд входят:

 - Режим выделения позволяет выделять или перетаскивать компоненты курсором при нажатии левой кнопки мыши. Выделенные компоненты окрашиваются в красный цвет. Для снятия выделения надо щелкнуть левой кнопкой мыши на свободном участке поля. Выделенный компонент можно удалять, поворачивать и т.п., щелкнув правой кнопкой мыши.

 - Вставка последнего компонента.

 - Служит для вставки проводников в схему (пишущий карандаш).

 - Вставка текста или комментариев в схемы и результаты анализа.

 - Позволяет разъединять компоненты или удалять соединяющие точки между проводниками и проводными соединениями.

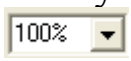
 - Повороты выделенного компонента.

 - Зеркальное отражение выделенного компонента.

Группа кнопок  :

 - включение/выключение сетки;

 - увеличение масштаба выбранной части текущего вида;

 - установка масштаба от 10% до 200%.

Интерактивное меню включает:

 - режим постоянного тока,

 - режим переменного тока,

 - непрерывный переходной режим,

 - однократный переходной режим с установленным временем анализа,

 - цифровой режим,

 - режим работы с компонентами цифровых цепей и вычислительными операциями, которые описываются на языке VHDL (**Very** High Speed Integrated Circuit **Hardware Description Language).**

В панель инструментов также входят кнопки:

 - список режимов работы,

 - выбор цели оптимизации или изменения установок,

 - выбор управляемого объекта для изменения параметров или оптимизации,

 - позволяет установить погрешности компонентов,

 - позволяет показать трехмерное изображение компонента,

 - вызывает диалог, который инициирует модель проектирования печатных плат,

- поиск компонентов из библиотеки.

5. *Панель компонентов.*

Компоненты расположены в группах, поименованных на кнопках панели инструментов. Сначала выбирают группу, а затем требуемый компонент. Щелкнув на выбранном компоненте, курсором переместите компонент на рабочее поле окна.

6. *Поиск компонентов*. Этот инструмент позволяет найти по имени любой компонент из каталога.

7. *Открытие таблицы файлов*. Можно иметь несколько разных файлов и выбрать нужный файл из таблицы.

8. *Включение и выключение редактора схем.*

9. *Линия помощи* - дает краткое описание выделенного компонента.

2.3. Сборка цепи и соединение компонентов

Компоненты выбирают из панели компонентов и их символы перемещают мышью на требуемые позиции. Компоненты можно поворачивать и зеркально отображать. После размещения компонентов, двойным щелчком левой кнопкой мыши открывают окно для установки параметров и этикетки компонента. Значения параметров могут лежать в пределах 10^{-12} – 10^{+12}. Программа TINA автоматически присваивает этикетку каждому компоненту и отображает численное значение его параметра. Набор компонентов весьма обширный и включает пассивные компоненты, источники сигналов, ключи, полупроводниковые элементы, цифровые схемы, микроконтроллеры и т.д.

Проводники устанавливают короткозамкнутое соединение между двумя контактами компонентов. Для создания проводника можно воспользоваться курсором, протянув проводник от одного компонента до другого. Курсор действует как пишущее перо. Подведите курсор к контакту компонента. Он отобразится как пишущий карандаш. Нажмите на левую кнопку мыши и

протяните нужный проводник. Удобнее пользоваться пишущим карандашом. Он позволяет начинать проводник в любом месте рабочего поля. Когда Вы намерены завершить создание проводника, нажмите правую кнопку мыши. В открывшемся меню можно редактировать положения проводника.

2.4. Входы и выходы

Некоторые виды анализа (передаточные характеристики на постоянном токе, частотные характеристики и т.д.) не могут выполняться, пока не определены входы и выходы цепи. Они определяют, где прилагается воздействие и откуда снимать реакцию цепи. Выходы также определяют, какие графики будут отображаться в выбранном режиме анализа. Источники и генераторы должны быть подключены к входам, а измерители – к выходам. Измерители могут также служить для определения количественного значения входных сигналов, которые будут использоваться при вычислении переходных характеристик и функций в режиме переменного тока.

В качестве выходов мы будем использовать приборы из панели компонентов на вкладке *Meters*:

 - пин измерения напряжения;

 - вольтметр;

 - амперметр, а также различные индикаторы и дисплеи.

Подробно моделирование микроконтроллеров в программе TINA Вы сможете изучить в процессе выполнения лабораторных работ.

2.5. Сборка схемы генератора импульсов

Начнем изучение работы в программе TINA с простейшей схемы генератора импульсов. Устройство на микроконтроллере PIC16F84A должно генерировать последовательность прямоугольных импульсов с периодом 200 мкс и скважностью 2.

Считаем, что Вы уже успешно установили программу TINA. На панели компонентов выбираем вкладку *Logic ICs – MCUs*, на вкладке *MCU* открываем каталог микроконтроллеров и среди

множества прочих выбираем PIC16F84A. Перетаскиваем этот микроконтроллер на рабочее поле и собираем схему рис.2.3.

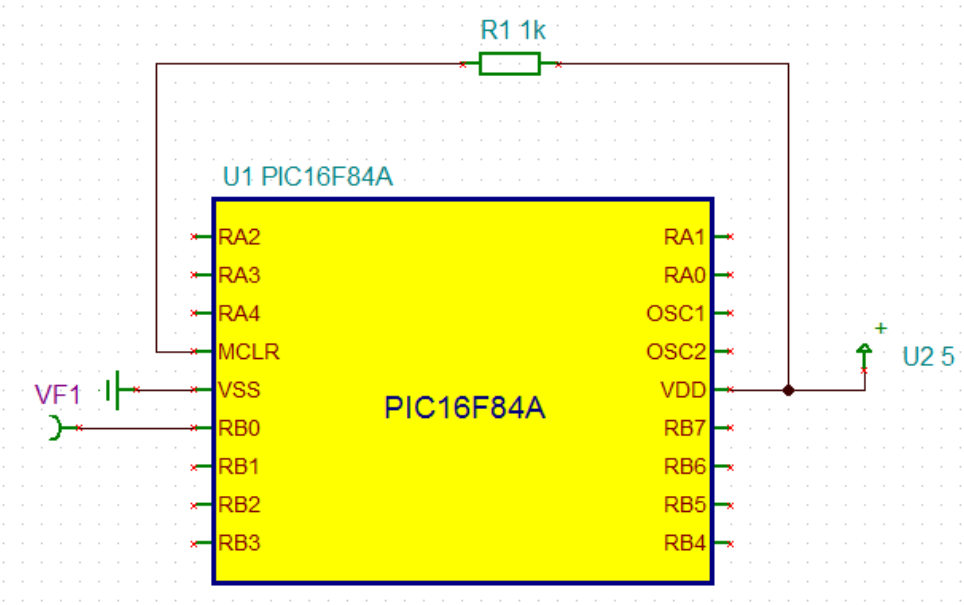


Рис.2.4. Схема генератора импульсов

Вместо подключения кварцевого генератора в свойствах микроконтроллера мы зададим требуемую тактовую частоту 4МГц. Питание подано на вывод VDD. Вывод VSS заземлен. На вывод сброса микроконтроллера $\overline{\text{MCLR}}$ через резистор R1 подаем высокий уровень. Выходом будет служит вывод RB0, к которому подключен пин напряжения VF1 из меню *Meters*.

Создадим папку TI-GEN и сохраним в ней схему под именем TIGEN1.

ТИНА включает в себя мощный инструмент моделирования цифровых схем на языке VHDL, который предлагается использовать для микроконтроллеров, если в схемах устройств нет аналоговых компонентов, которые не описаны на VHDL. В наших схемах такие компоненты (например, резистор), как правило, присутствуют. Поэтому мы будем проводить моделирование в смешаннном режиме. Для этого в главном меню надо выбрать *Analysis-Options*, включить *Enable VHDL mixed mode* и *Enable MCU Code Debugger*. Дальнейшее описание составлено для этого режима.

2.6. Редактор блок-схем программы TINA

Написание кода в ассемблере достаточно трудоемко. Для несложных задач можно упростить разработку программного обеспечения, если вместо ручного кодирования программы использовать редактор блок-схем (*Flowchart*) и отладчик программы TINA. Это позволяет генерировать и отлаживать код микроконтроллера, используя только символы и линии управления потоком, а затем по блок-схеме получить код программы, выполнять или отлаживать его в цифровой или смешанной схеме.

Составим блок-схему для нашего генератора импульсов. Назовем ее GEN1FLOW. В схеме рис.2.4 выделяем микроконтроллер левой кнопкой мыши, нажимаем на правую кнопку и в выпадающем меню выбираем *Properties*. В открывшемся окне свойств (рис.2.5) установим тактовую частоту 4 МГц и выбираем *MCU-[ASM File Name]*.

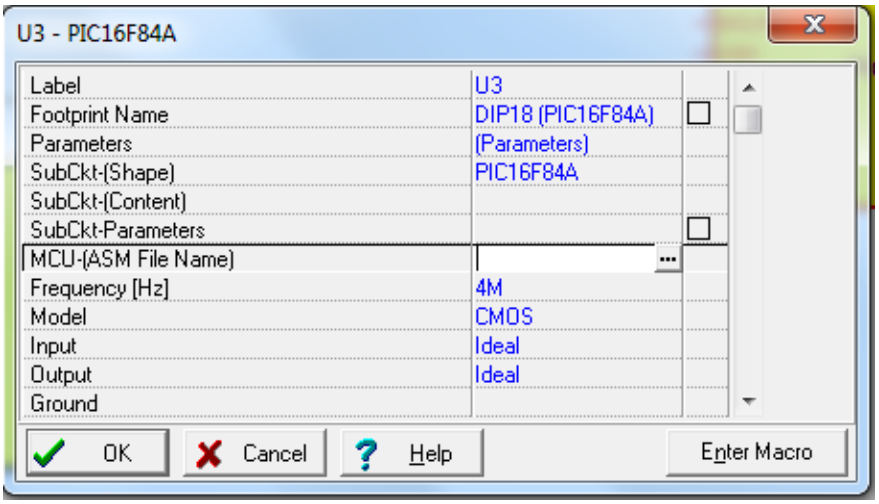


Рис.2.5. Окно свойств микроконтроллера

Нажимаем ... и в окне *MCU Input File Selection* (рис.2.6) выбираем *Flowchart*, нажимаем кнопку *Flowchart* и TINA откроет редактор блок-схем (рис.2.7).

Окно редактора состоит из двух частей. Слева находятся символы, которые мы можем поместить на правое рабочее поле. Описание и настройку символов можно найти на вкладке *Help* редактора блок-схем.

Перенесем на рабочее поле символы, с помощью которых мы сможем изменять сигнал с `1` на `0` на выводе RB0 порта B с задержкой 100 мкс и соединим символы линиями, начиная с символа Start с учетом направления потока (рис.2.8).

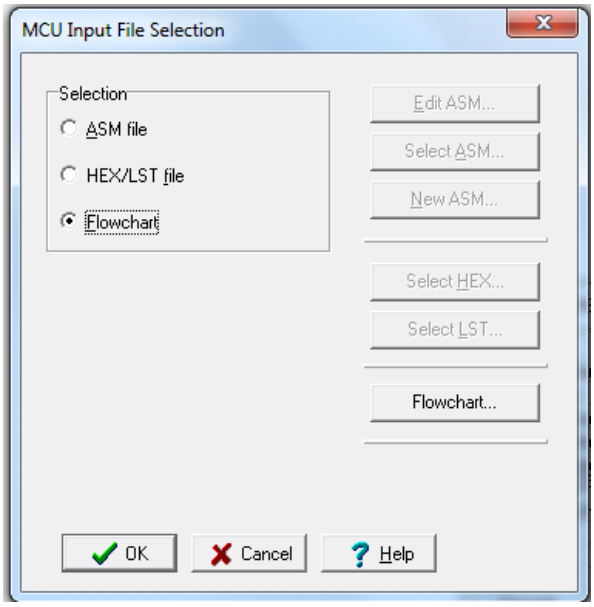


Рис.2.6. Окно выбора входного файла

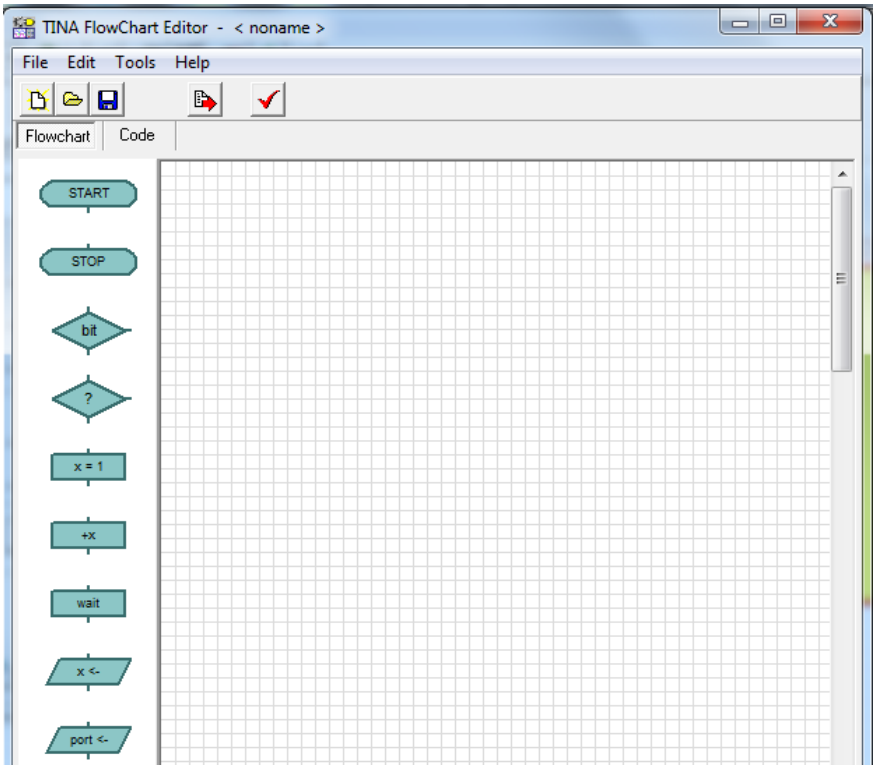


Рис.2.7. Окно редактора блок-схем

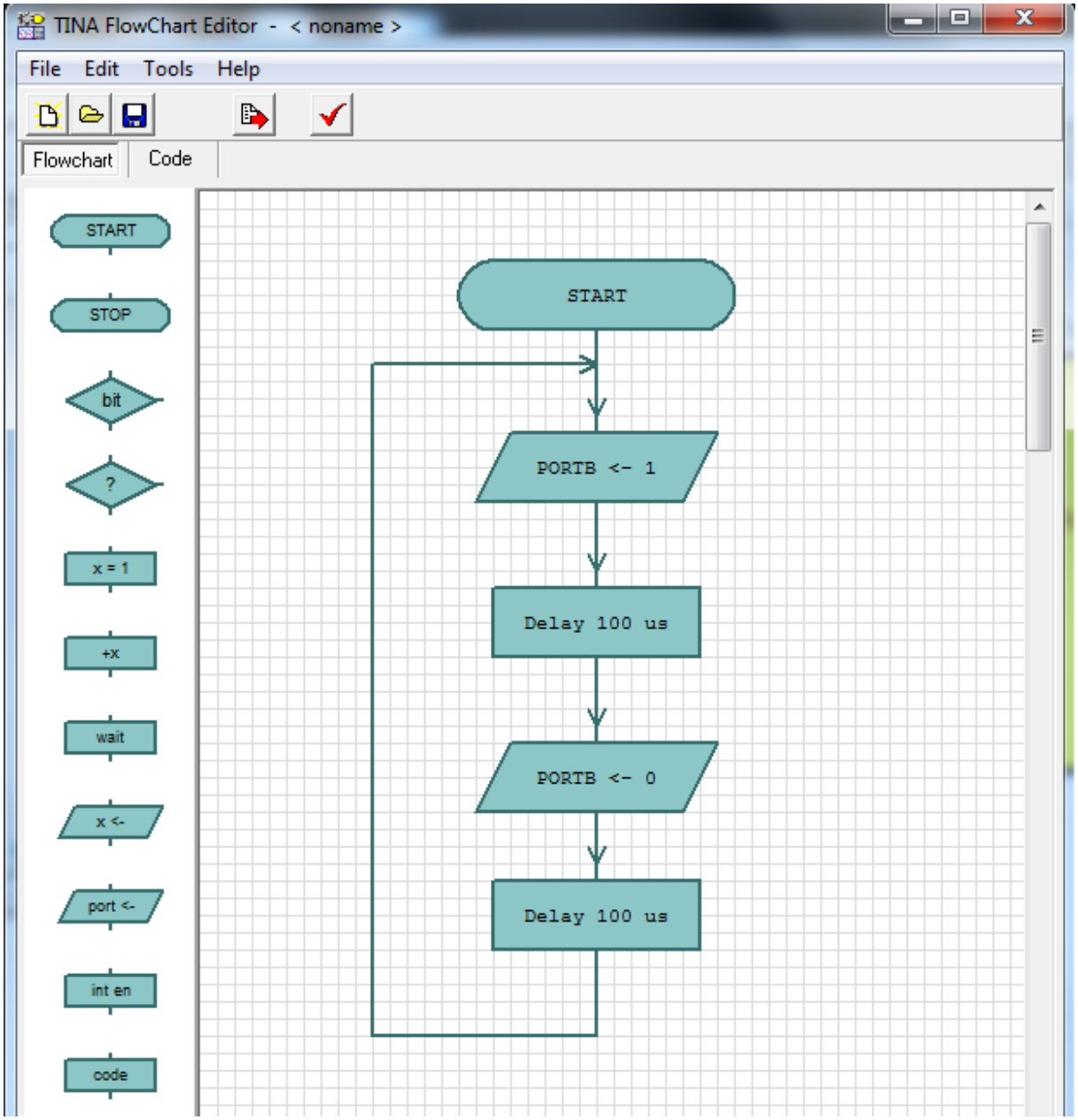


Рис.2.8. Блок-схема генератора импульсов

Выполним настройку параметров символов (рис.2.9). Верхний выходный символ PORTB<-1 на активном выводе RB0 устанавливает `1`. Задержка равна 100 мкс. Нижний выходной символ PORTB<-0 на активном выводе RB0 устанавливает `0`.

Чтобы сделать формальную проверку правильности соединений, нажмем  . В окне *Flowchart Check* должно быть «*Message: No Error*».

Далее нажимаем кнопку  *Save To Macro*, чтобы сохранить блок-диаграмму и преобразовать ее в код программы.

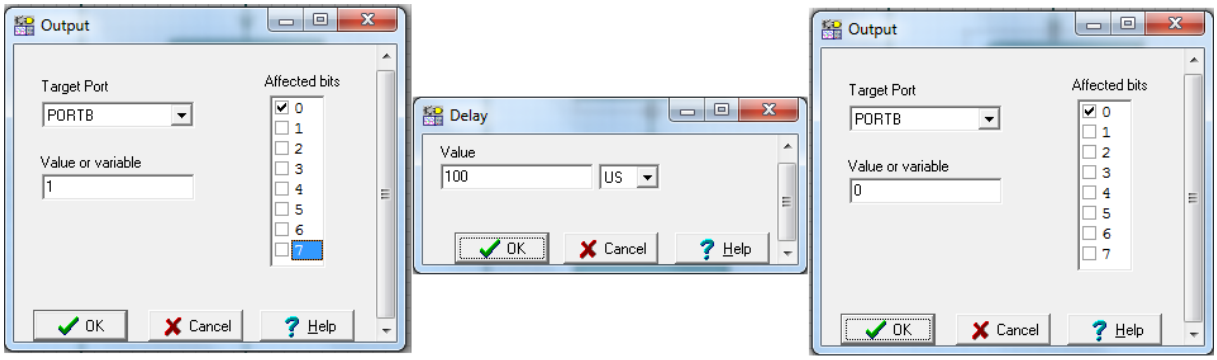


Рис.2.9. Настройка параметров блоков

Сохраним блок-схему в папке TI-GEN под именем GEN1FLOW.

Нажимаем в отладчике *Code* и в рабочем окне получаем код программы в ассемблере (рис.2.10).

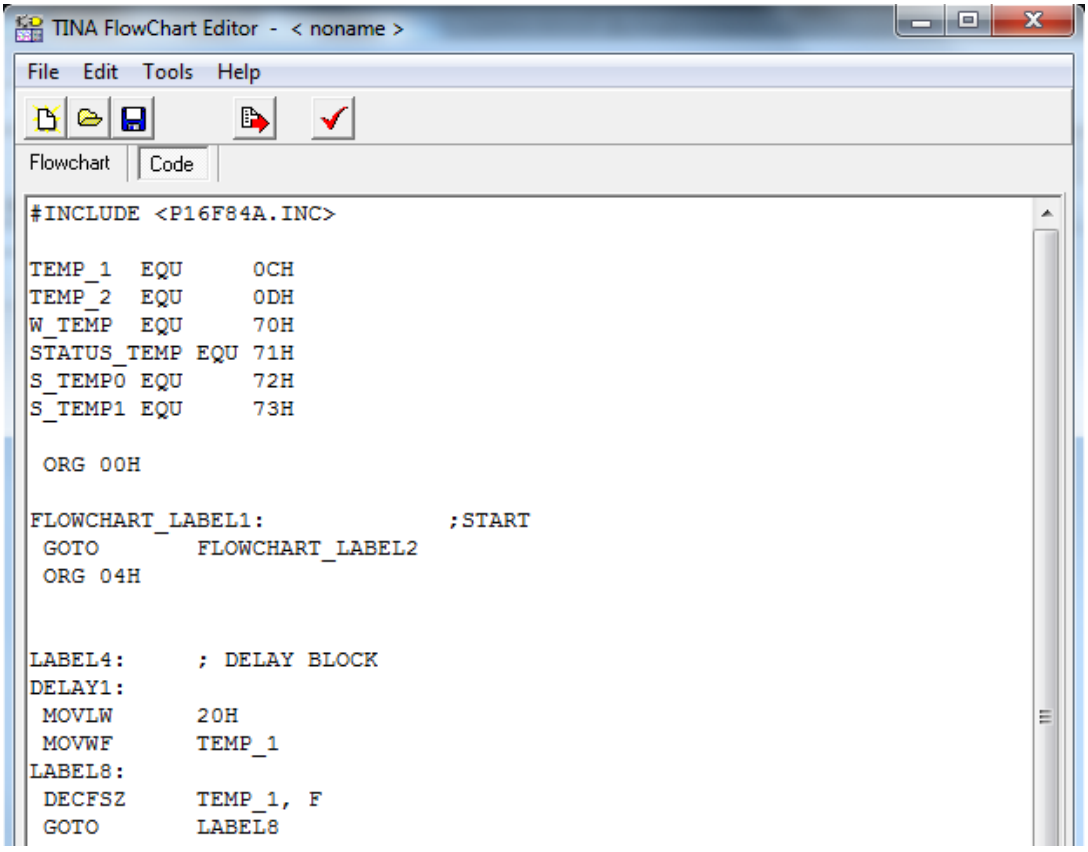


Рис.2.10. Код программы генератора импульсов в ассемблере

Выделим весь текст программы, скопируем операцией Ctrl-C, вставим файл в блокнот Notepad++ и сохраним в папке TI-GEN под именем GEN-asm с расширением *asm.

Текст программы представлен на листинге 2.1.

Программа генератора импульсов

Листинг 2.1.

```
#INCLUDE <P16F84A.INC>

    TEMP_1  EQU     0CH
    TEMP_2  EQU     0DH
    W_TEMP  EQU     70H
STATUS_TEMP EQU 71H
    S_TEMP0 EQU     72H
    S_TEMP1 EQU     73H

    ORG 00H

FLOWCHART_LABEL1:               ;START
    GOTO    FLOWCHART_LABEL2
    ORG 04H


LABEL4: ; DELAY BLOCK
DELAY1:
    MOVLW   20H
    MOVWF   TEMP_1
LABEL8:
    DECFSZ  TEMP_1, F
    GOTO    LABEL8
    RETURN
LABEL7: ; DELAY BLOCK
DELAY2:
    MOVLW   20H
    MOVWF   TEMP_2
LABEL9:
    DECFSZ  TEMP_2, F
    GOTO    LABEL9
    RETURN
```

```
FLOWCHART_LABEL2:              ;PORTB <- 1
    BSF     STATUS, RP0
    BSF     STATUS, RP1
    BCF     STATUS, RP1
    MOVLW   0FEH
    MOVWF   TRISB
    BCF     STATUS, RP0
    MOVLW   1
    MOVWF   PORTB
FLOWCHART_LABEL3:              ;DELAY 100 US
    CALL   LABEL4
FLOWCHART_LABEL5:              ;PORTB <- 0
    BSF     STATUS, RP0
    BSF     STATUS, RP1
    BCF     STATUS, RP1
    MOVLW   0FEH
    MOVWF   TRISB
    BCF     STATUS, RP0
    MOVLW   0
    MOVWF   PORTB
FLOWCHART_LABEL6:              ;DELAY 100 US
    CALL   LABEL7
    GOTO    FLOWCHART_LABEL2

    END
```

2.7. Отладка программы

В главном меню программы TINA выбираем *Analysis* и включаем *Enable MCU Code debuger*. Выбираем режим *TR* и на вкладке *Interactive* нажимаем *Start*. В окне отладчика можно работать с блок-схемой (рис.2.11), с кодом в ассемблере (рис.2.12) и одновременно с блок-схемой и кодом (рис.2.13). На вкладке *File* (рис.2.13) можно выполнить сохранение блок – схемы, а также сохранить файлы программы с расширением *ASM, *LST, *HEX.

Сохраним все четыре файла в папке TI-GEN, а затем откроем файлы *ASM, *LST и *HEX с помощью Notepad++.

Код программы в ассемблере (листинг 2.1) не самый компактный для поставленной задачи, поскольку он содержит

дополнительные кооманды и метки, которые образовались в результате преобразования блок-схемы в код ассемблера.

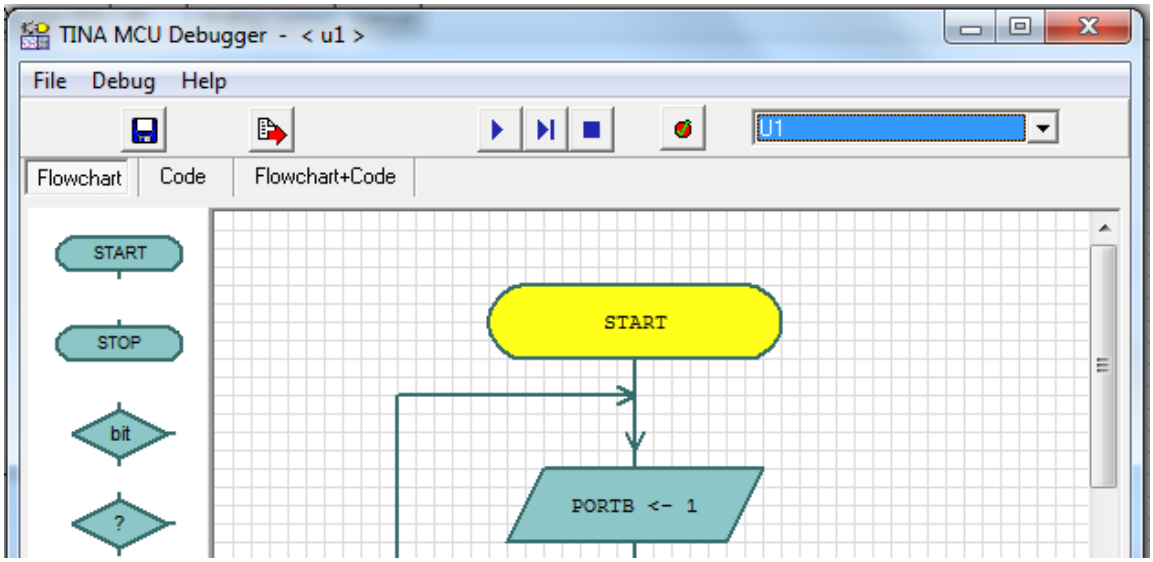


Рис.2.11. Отладка по блок-схеме

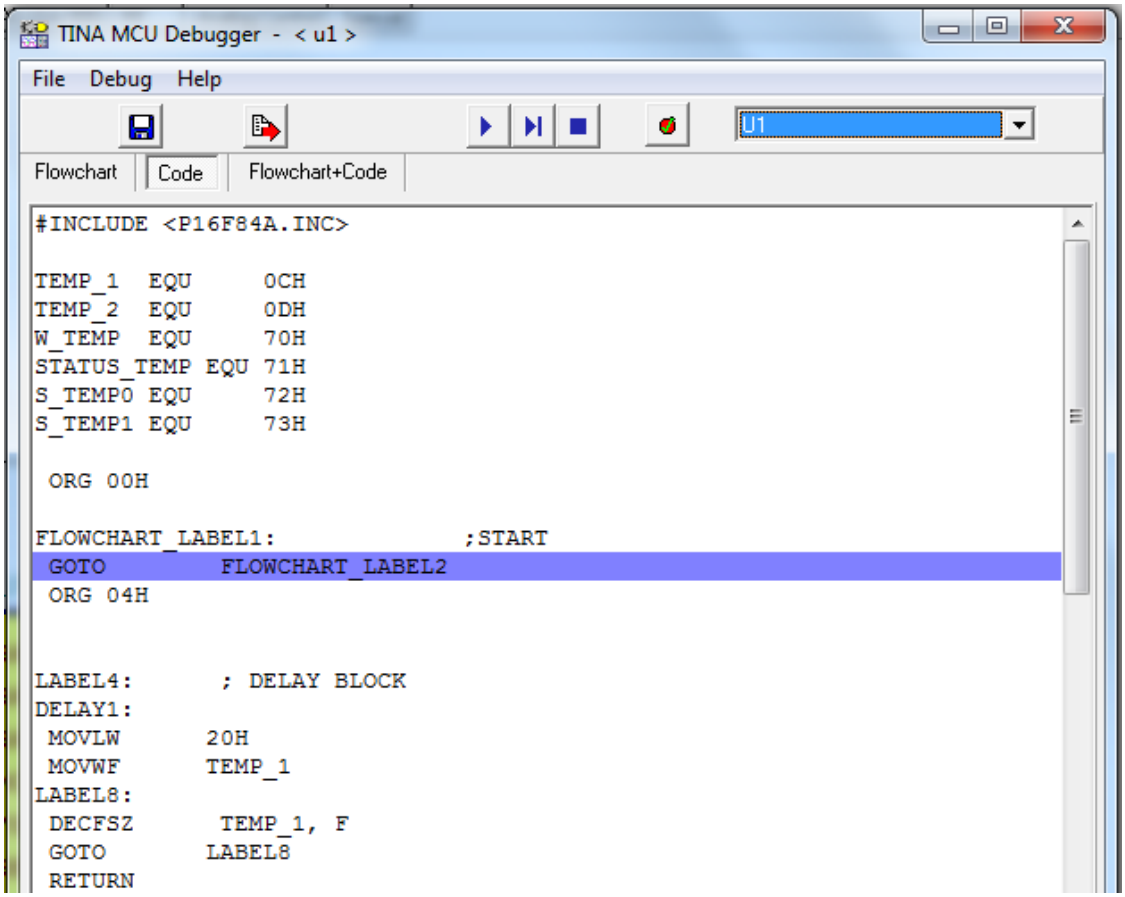


Рис.2.12. Отладка по коду в ассемблере

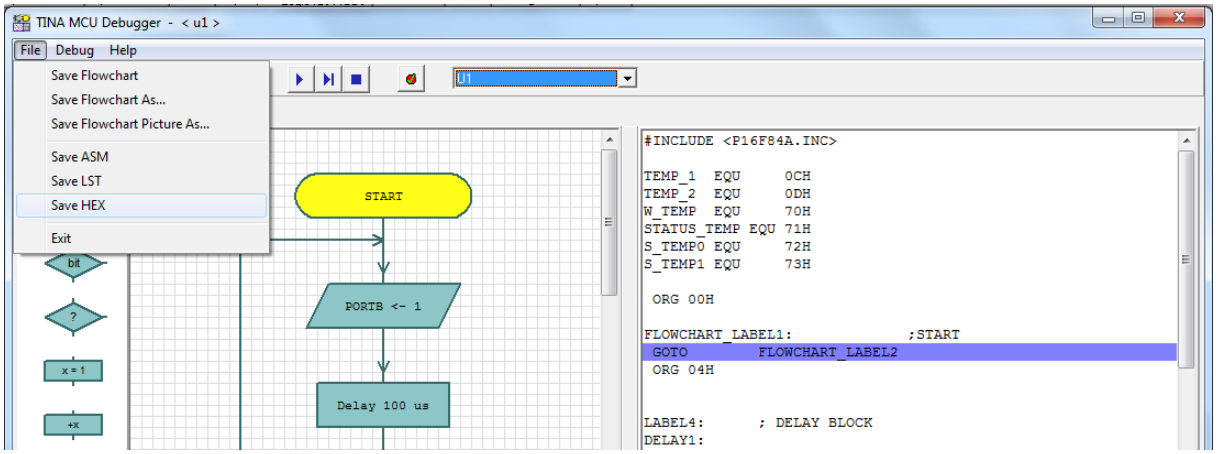


Рис.2.13. Отладка по блок-схеме и коду одновременно

Файл программы в формате LST (рис.2.14) содержит адреса ячеек памяти, в которых будет сохранен машинный код (Loc), машинный код, соответствующий каждой инструкции (Object Code), номер линии в листе файла (Line), исходный код, включая комментарии (Source text and comments). Кроме этого LST-файл содержит численные значения адресов ячеек памяти, которыми будут заменяться символы (например: TEMP_1 имеет значение адреса 12, что соответствует 0CH).

```
1  Loc  Object Code    Line    Source text and comments
2
3                       0001    #INCLUDE <P16F84A.INC>
4                       0001    LIST
5                       0002    ; P16F84A.INC  Standard Header File, Version 2.00
6                       0134    LIST
7                       0002
8          000C         0003    TEMP_1  EQU     0CH
9          000D         0004    TEMP_2  EQU     0DH
10         0070         0005    W_TEMP  EQU     70H
11         0071         0006    STATUS_TEMP EQU 71H
12         0072         0007    S_TEMP0 EQU     72H
13         0073         0008    S_TEMP1 EQU     73H
14                      0009
15 0000                 0010    ORG 00H
16                      0011
17 0000                 0012    FLOWCHART_LABEL1:               ;START
18 0000    280E         0013    GOTO    FLOWCHART_LABEL2
19 0001                 0014    ORG 04H
20                      0015
21                      0016
22 0004                 0017    LABEL4: ; DELAY BLOCK
23 0004                 0018    DELAY1:
24 0004    3020         0019    MOVLW   20H
25 0005    008C         0020    MOVWF   TEMP_1
```

Рис.2.14. Файл программы в формате LST

Исполняемый HEX-файл содержит машинные коды в шестнадцатиричном формате (рис.2.15).

```
:020000040000FA
:020000000E28C8
:100080020308C008C0B062808002030 8D008D0BCA
:100018000B2808008316031703 13FE30860083128B
:10002800013086000420831603170313FE30860070
:0A00380083120030860009200E2814
:00000001FF
```

Рис.2.15. HEX-файл программы

Отладка программы по блок-схеме

В свойствах микроконтроллера выбираем *MCU Input File Selection* (рис.2.6), *Flowchart*, нажимаем кнопку *Flowchart* и нажимаем кнопку  *Save To Macro*, чтобы сохранить блок-диаграмму. Далее выбираем режим *TR*, в меню *Interactive* нажимаем *Start*. Многократно нажимаем кнопку *Step forward*  и наблюдаем пошаговое выполнение программы (рис.2.16). Напряжение на выходе RB0 изменяется с 50 мВ до 4,95 В.

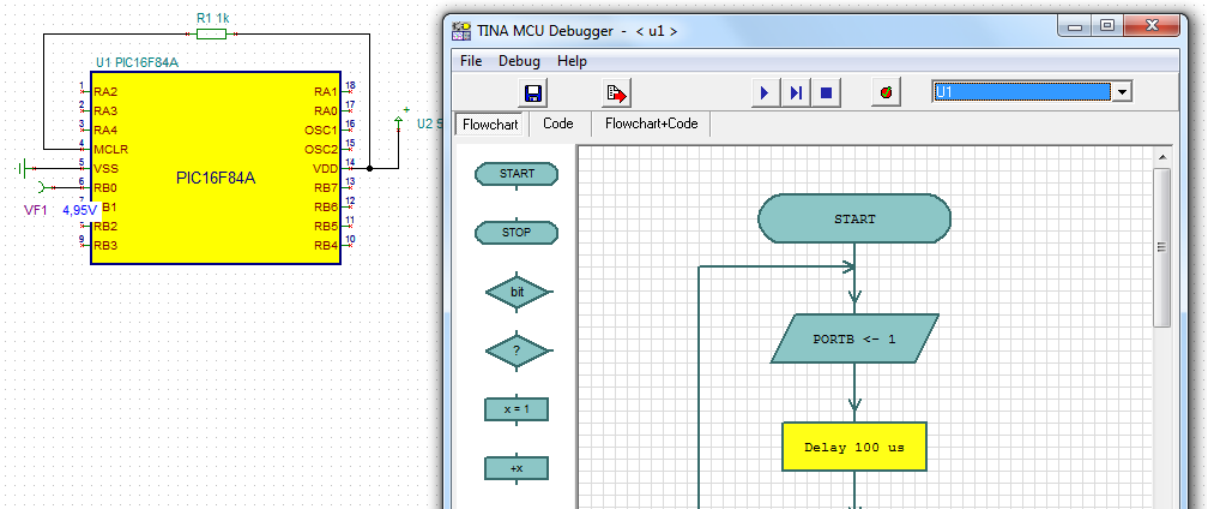


Рис.2.16. Отладка по блок-схеме

Включим непрерывный режим *Run*  . Программа будет выполняться непрерывно. Для останова программы нажимаем *Stop*  .

Измерим период импульсного сигнала. Для этого остановим интерактивный анализ, в главном меню выбираем *Analysis-Transient.* В окне (рис.2.17) устанавливаем временной интервал анализа от 0 до 1 мс и нажимаем Ok.

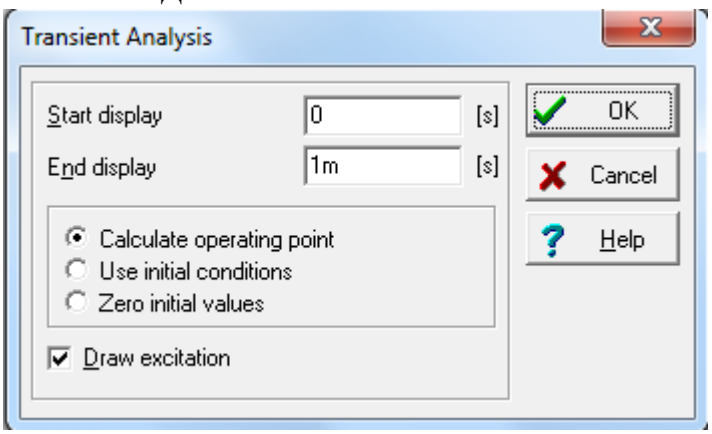


Рис.2.17. Установка времени анализа

На временной диаграмме с помощью курсоров определяем период сигнала, равный 219,03 мкс. Увеличение периода по сравнению с установленными задержками обусловлено затратами времени на выполение всех остальных команд в цикле программы.

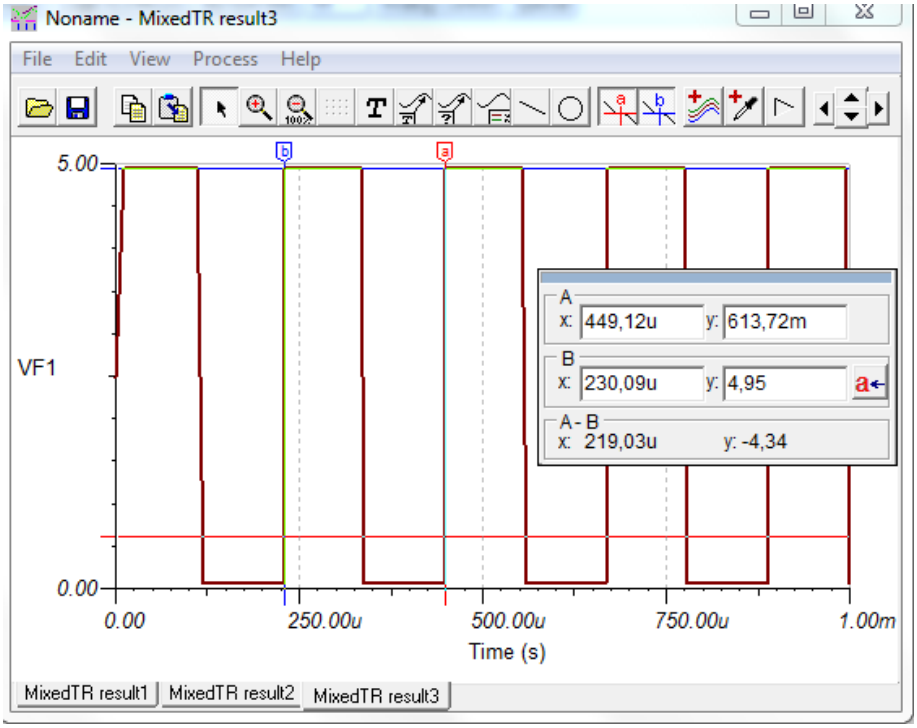


Рис.2.18. Временная диаграмма импульсного сигнала

Отладка программы по коду в ассемблере

В окне отладчика нажмем кнопку *Code*. Включаем пошаговый режим  и наблюдаем последовательное выполнение команд. Обратите внимание на то, что в блоке задержки DELAY1 программа многократно зацикливается на командах:

```
DECFSZ   TEMP_1,F
GOTO     LABEL8.
```

Длительность задержки задается командами:

```
MOVLW     20H
MOVWF     TEMP_1
```

Если изменить число 20H, равное десятичному числу 32, измениться и длительность первой задержки.

Зададим точки останова  (*Breakpoint*) на командах `CALL  LABEL4` и `CALL LABEL7` (рис.2.19). Делаем пуск в непрерывном режиме *Run*. Первый останов показывает значение счетчика команд *PCL*=16 и содержание регистров. Регистр PORTB=1. Останов на команде `CALL LABEL7` показывает, что PORTB=0.

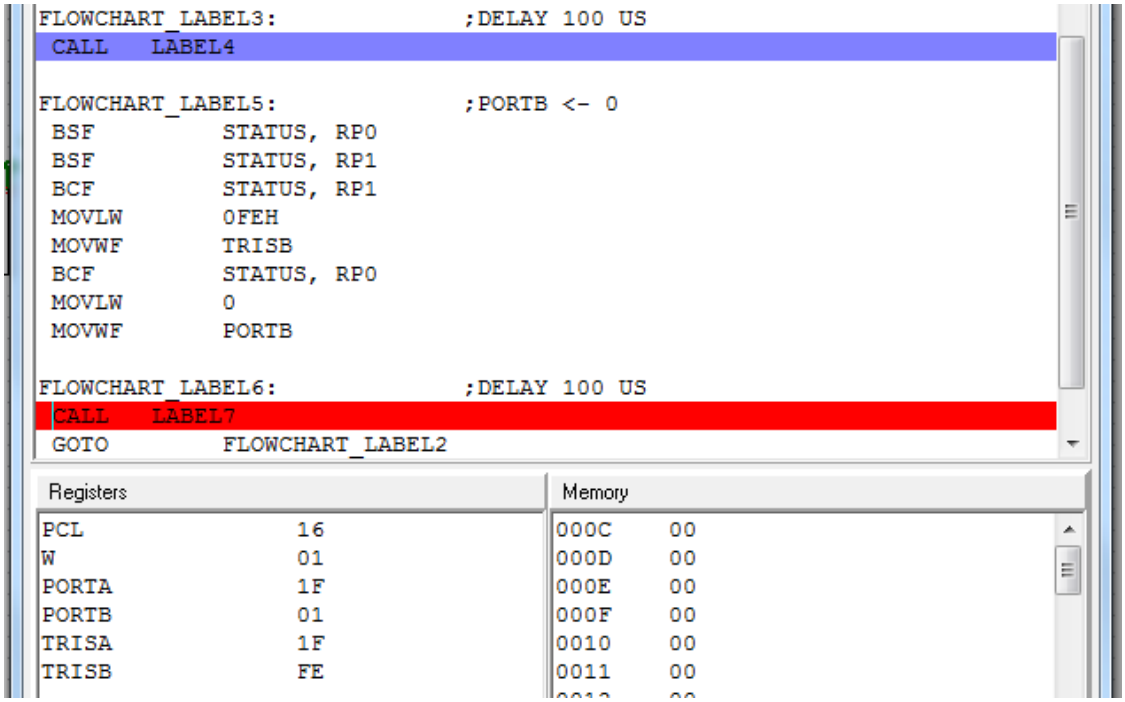


Рис.2.19. Задание точек останова

2.8. Загрузка HEX- файла программы

Готовый HEX-файл программы можно загрузить в микроконтроллер следующим образом. В свойствах микроконтроллера выбираем *MCU Input File Selection-HEX/LST file-Select HEX* (рис.2.20). Далее из папки TI-GEN открываем файл GEN1.hex. Выполним проверку в режиме *Analysis-Transient* и убедимся, что программа работает правильно.

Действуя аналогично, загрузите файлы GEN1.asm и GEN1.lst и проверьте функционирование генератора.

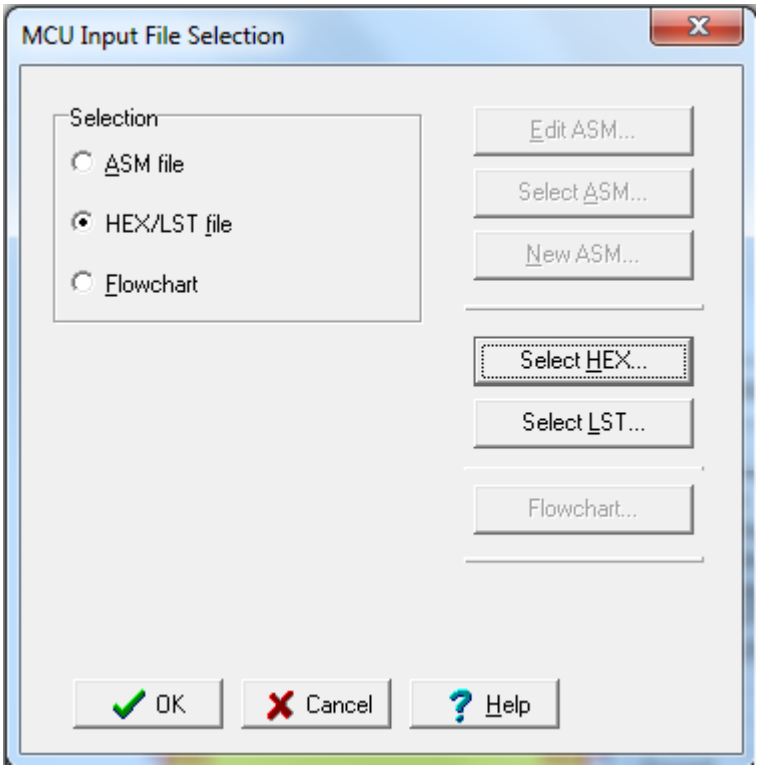


Рис.2.20. Загрузка HEX-файла

2.9. Редактирование кода в ассемблере

Изменим форму импульсного сигнала так, чтобы уровень `1` был в течении 1/4 периода. Так как период условно равен десятичному числу 64, то первую задержку зададим числом десятичным числом 16 (или 10H), а вторую задержку зададим числом 48 (или 30H).

В свойствах микроконтроллера выбираем *MCU Input File Selection- ASM file-Edit ASM* (рис.2.21). Измениям численные

константы в командах MOVLW, выполняем компиляцию и сохраняем отредактированный файл .

Далее выполняем *Analysis-Transient* и получаем новую форму импульсного сигнала (рис.2.22).

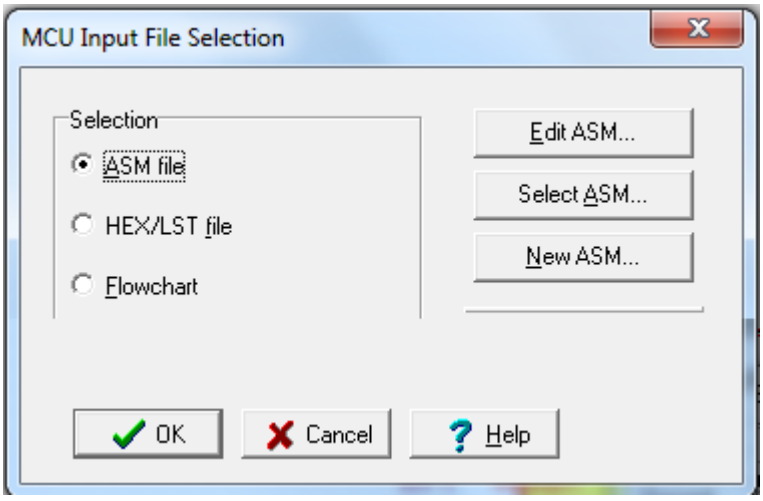


Рис.2.21. Установка редактирования в ассемблере

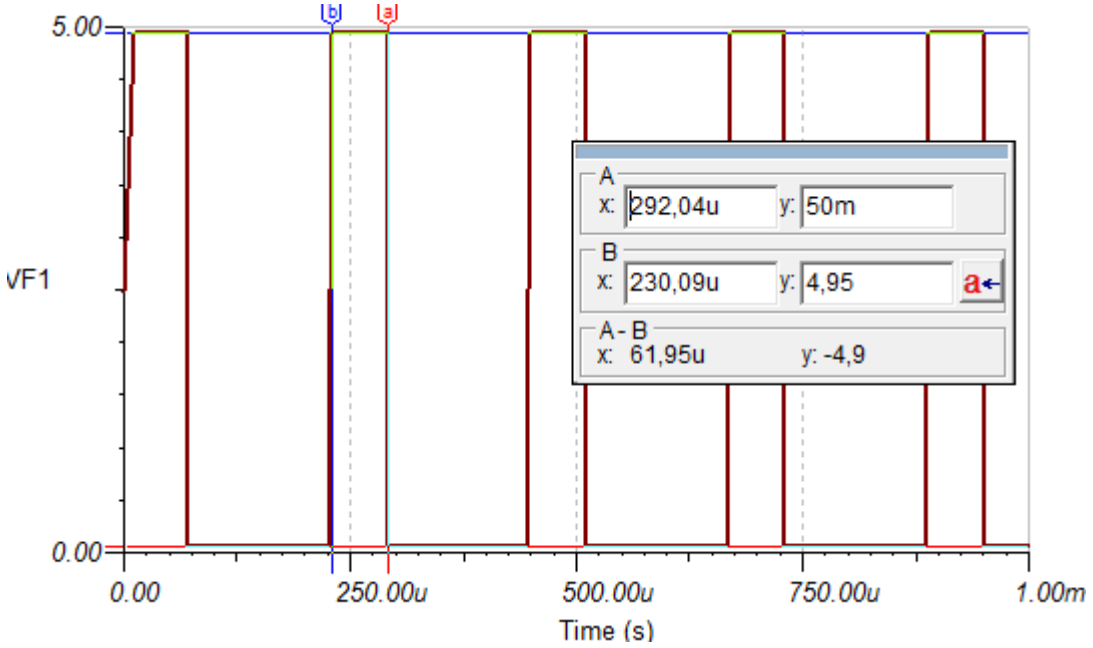


Рис.2.22. Изменение формы сигнала после редактирования

Контрольные вопросы

1. Какие исследования можно проводить в программной среде TINA ?
2. Какие режимы анализа используются в программе TINA ?

3. Назовите группы панели компонентов программы TINA.
4. Какие компоненты входят в группу Basic ?
5. Используя вкладки Help из свойств компонентов, назовите назначение компонентов из группы Meters.
6. Назовите назначение компонентов из группы Sources.
7. Как выбрать и установить на рабочем поле нужный микроконтроллер ?
8. Как выполнить соединение компонентов на схеме ?
9. Расскажите о назначении и использовании режимов *DC-Analysis, AC-Analysis, Transient.*
10. Расскажите о назначении и использовании режима *Interactive.*
11. Как открыть в программе редактор блок-схем ?
12. Используя вкладки Help, расскажите о назначении и настройке символов редактора блок-схем.
13. Как правильно соединять символы блок-схемы, чтобы учесть последовательность выполнения программы ?
14. Какие программные файлы формируются с помощью блок-схемы ?
15. Как выполнить отладку программы по блок-схеме ?
16. Как выполнить отладку по коду в ассемблере ?
17. Для чего применяют точки останова ?
18. Какую информацию содержат файлы с расширением LST?
19. Как проверить работу модели генератора в режиме *Analysis-Transient* ?
20. Как загрузить в микроконтроллер готовый HEX-файл программы ?
21. Как загрузить готовый LST-файл ?
22. Как загрузить готовый ASM- файл ?
23. Как можно оьредактировать программу по блок-схеме?
24. Как отредактировать ASM-файл программы в микроконтроллере и проверить результат этого ?

Глава 3. ПРОГРАММИРОВАНИЕ И ОТЛАДКА В СРЕДЕ MPLAB IDE

3.1 Краткие сведения о среде MPLAB IDE

MPLAB IDE это бесплатная интегрированная среда разработки для микроконтроллеров PICmicro фирмы Microchip Technology Incorporated. MPLAB IDE позволяет писать, отлаживать и оптимизировать текст программы. MPLAD IDE включает в себя редактор текста, симулятор (моделирует выполнение программы в микроконтроллере с учетом состояния портов ввода/вывода) и менеджер проектов, поддерживает работу эмуляторов (моделируют работу микроконтроллера в масштабе реального времени непосредственно в устройстве пользователя), программаторов и других отладочных средств фирмы Microchip и других производителей.

Настаиваемые инструментальные средства, тематическая помощь, «выпадающие» меню в MPLAB IDE позволяют Вам:

- получить код программы;
- наблюдать выполнение программы с помощью симулятора, или в реальном времени, используя эмулятор и дополнительную аппаратную часть;
- определять время выполнения программы;
- просматривать текущее значение переменных и специальных регистров и т.д.

MPLAB IDE позволяет создавать исходный текст программы в полнофункциональном текстовом редакторе, легко выполнять исправление ошибок при помощи окна результатов компиляции, в котором указываются возникшие ошибки и предупреждения. Используя менеджер проектов, можно указать исходные файлы программы, объектные файлы, библиотеки и файлы сценария.

MPLAB IDE обеспечивает разнообразные средства симуляции и эмуляции исполняемого кода для выявления логических ошибок. Их основные особенности:

- большое количество сервисных окон, позволяющих контролировать значения регистров памяти данных и выполнение инструкций микроконтроллера:
- окна исходного кода программы, листинга программы, исполняемого кода программы позволяют оценить качество компиляции;
- пошаговое выполнение программы, система точек останова, трассировки, стимулов предназначена для быстрой и удобной отладки программы.

Мы будем изучать одну из последних версий MPLAB IDE v.8.92. Эту программу Вам следует найти на сайте Microchip и установить на компьютере.

3.2. Создание проекта в среде MPLAB IDE

Чтобы создать код, который будет выполняться PIC – контроллером, исходные файлы должны быть помещены в проект программы MPLAB IDE. Используя менеджер проектов, можно указать исходные файлы программы, объектные файлы и файлы сценария. Код будет создан с использованием выбранных инструментальных средств (ассемблера, компилятора, компоновщика). В MPLAB IDE этот процесс контролирует *Project manager*.

Для создания любого проекты сначала надо создать папку, в которой будут сохраняться все файлы.

Создадим папку C:\MP-GEN. Открываем программу MPLAB IBE и выполняем следуюшие шаги:

Выбор устройства

В главном меню выбираем *Configure- Select device* и применяемый PIC-контроллер (рис.3.1). Зеленые индикаторы означают полную поддержку, желтые означают, что некоторые функции не поддерживаются. Красные индикаторы говорят о том, что указанные функции для выбранного микроконтроллера не поддерживаются.

Создание проекта состоит из нескольких шагов.

1. Создание проекта и проверка выбора устройства

В главном меню выбираем *Project - Project Wizard*. Открывается окно приветствия (рис.3.2). Проект это путь, по которому файлы будут организованы для компиляции и ассемблирования. В нашем первом проекте мы будем использовать простой файл в ассемблере и компилятор.

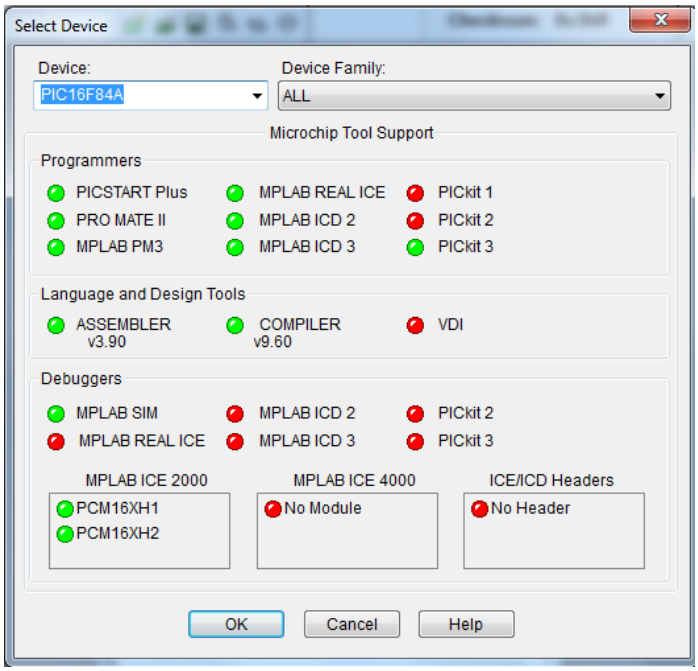


Рис.3.1. Окно выбора устройства

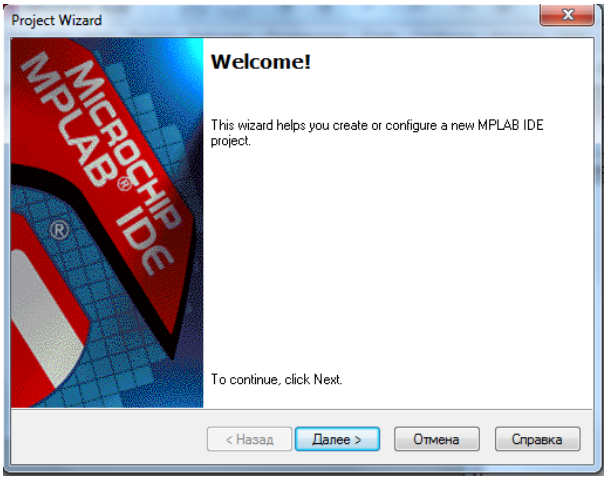


Рис.3.2. Окно приветствия MPLAB IDE

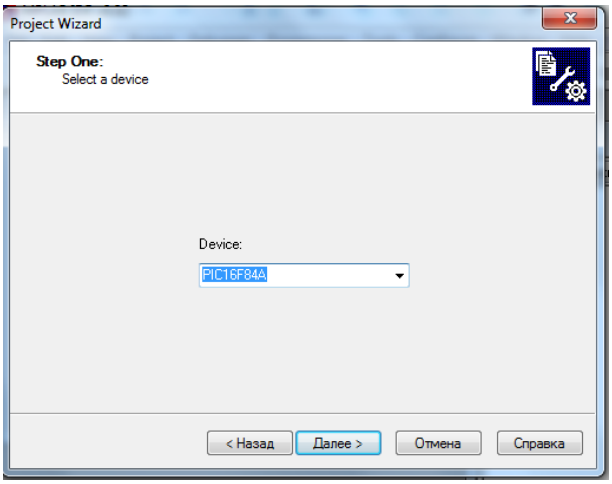


Рис.3.3. Выбор устройства

Убеждаемся, что в проекте используется выбранный нами микроконтроллер (Рис.3.3).

2. Установка инструментов языка

Устанавливаем языковые средства, которые используются в проекте. Для создания программы в ассемблере выбираем *Microchip MPASM Toolsuite* и соответствующие инструменты, показанные в диалоговом окне (рис.3.4).

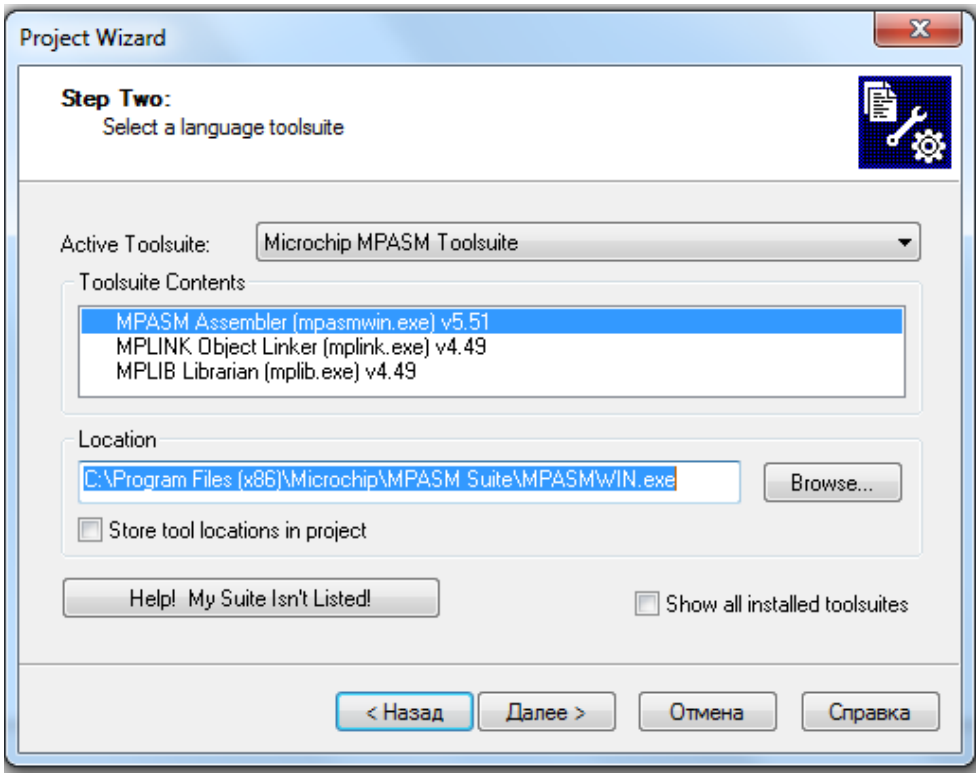


Рис.3.4. Выбор инструментальных средств языка для ассемблера

3. Создание нового проекта

Этот шаг мастера позволяет наименовать новый проект и поместить его в папку. В нашем первом примере мы будем проектировать генератор прямоугольных импульсов на микроконтроллере. Создаем проект в папке C:\MP-GEN, назовем наш проект MPGEN1 и нажимаем *Далее* (рис.3.5).

4. Добавление файлов в проект

Этот шаг мастера позволяет выбрать файлы для проекта. Для создания новых программ можно использовать шаблонный

файл MPLAB IDE. Шаблонные файлы – это простые файлы, которые используются для старта проекта. Они имеют особые секции для любого исходного файла и содержат информацию, которая поможет Вам написать и организовать Ваш код. Для каждого микроконтроллера есть два шаблонных файла: один - для абсолютного кода (без использования компилятора) в кодах системы команд и другой - для перемещаемого кода (после использования компилятора) в объектной директории.

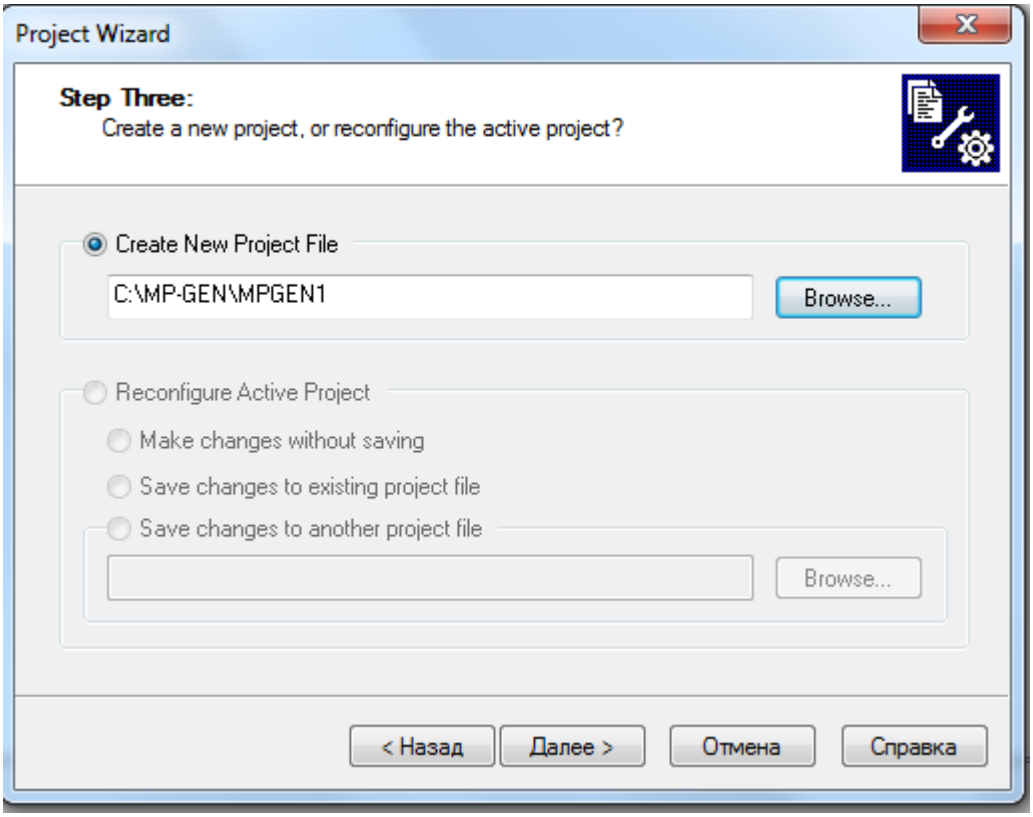


Рис.3.5. Наименование и размещение проекта

Так как в этом примере мы будем использовать компилятор, можно выбрать шаблонный файл *C:\Program Files\Microchip\MPASM Suite\Template\Object\16F84ATMPO.ASM*..

Код, использованный в этом файле, является перемещаемым и требует компиляции для исполнения в выходной программе. В нашем проекте компилятор используется автоматически при каждом изменении программы.

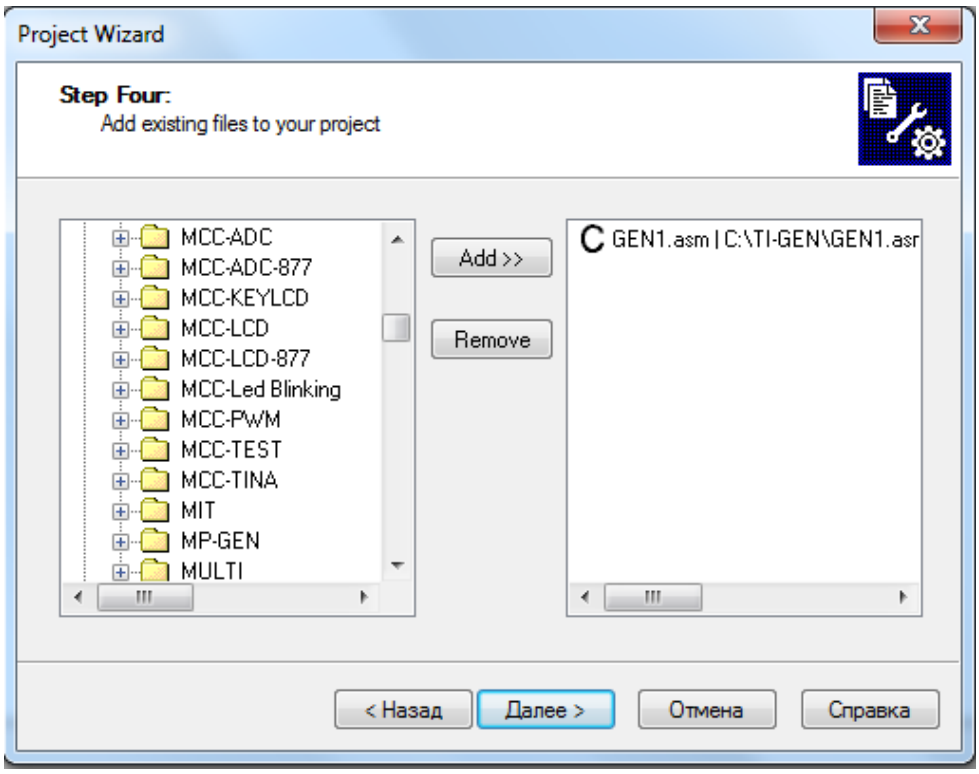


Рис.3.6. Добавление файлов в проект

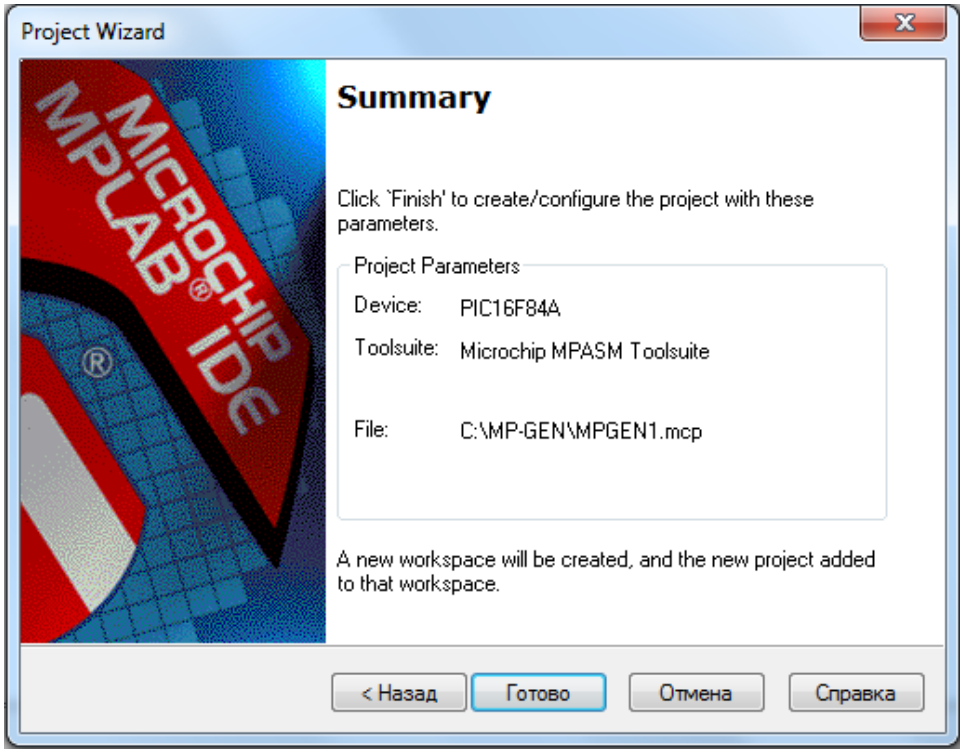


Рис.3.7. Итоговое окно завершения создания проекта

Добавим в проект файл C:\TI-GEN\GEN1.asm, который мы создали с помощью блок-схемы в программе TINA (рис.3.6). *После добавления файла в правом окне надо несколько раз щелкнуть по букве в начале файла, чтобы получить букву C !*. В этом случае файл будет скопирован в проект.

Итоговое окно (рис.3.7) подтверждает успешное создание проекта.

Убедимся, что диалоговое окно заполнено правильно и нажмем *Готово* для завершения проекта.

Посмотрим содержание папки C:\MP-GEN (Рис.3.8).

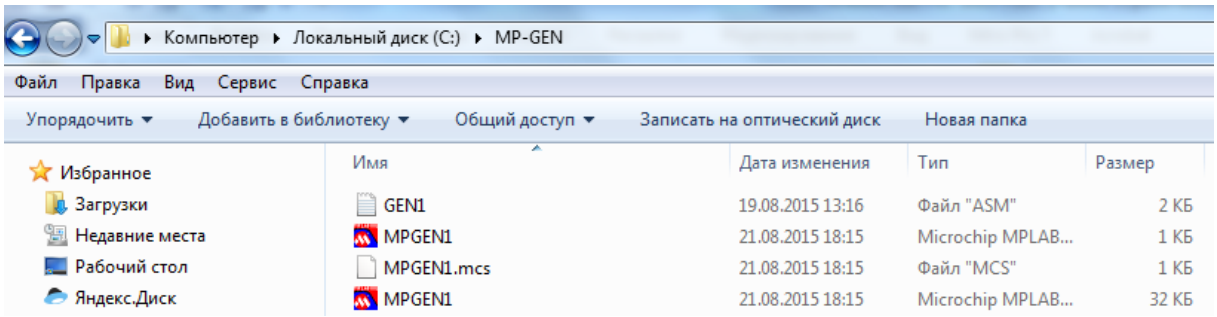


Рис.3.8. Содержание папки проекта

В окне *Window-MPGEN1.mcw* показаны все файлы проекта в текущий момент времени (рис.3.9).

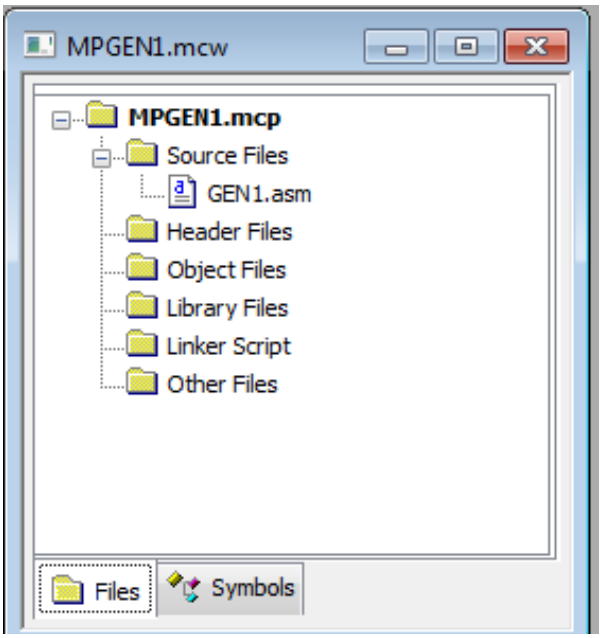
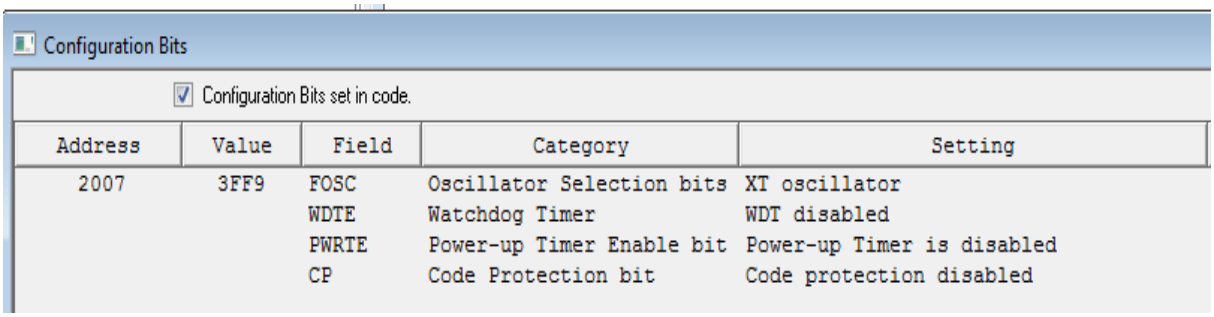


Рис.3.9. Файлы проекта

3.3. Установка битов конфигурации

В главном меню выбираем *Configure-Configuration Bits*, снимаем галочку в окошке *Configuration Bits set in code*, устанавливаем конфигурацию, показанную на рис.3.10, и снова включаем галочку. Отметим, что для продолжительной отладки надо отключить таймер WDT. После завершения отладки перед прошивкой микроконтроллера таймер WDT следует включить.



3.10. Установка битов конфигурации

3.4. Компиляция проекта

Из меню *Project* мы можем ассемблировать и компилировать файлы. Выберем симулятор, в котором мы будем проводим моделирование. Для этого в главном меню выбираем *Debbuger-Select Tool-MPLAB SIM.*

Выполняем компиляцию. Для этого выбираем *Project-Build All* и на открывшейся вкладке выбираем код *Absolute.*

Если в исходном файле нет ошибок, получим в выходном окне подтверждение успешной компиляции (рис.3.11) и открываем окно отладки.

Если в программе есть ошибка, появится запись: *Error[108] C:\MP-GEN\GEN1.ASM 19 : Illegal character (2)*. Щелкнем по этой строчке и увидим ошибку в тексте (рис.3.12).

Чтобы вывести подробные сведения об ошибках, выбираем *Project - Build Options - Project,* далее *MPASM Assembler*, выбираем *Output* из вкладки *Categories* и *Errors only* из вкладки *Diagnostic level*. Нажимаем OK.

Если компиляция выполнена успешно, в папке проекта появляются новые файлы с расширениями *HEX, *Lst.

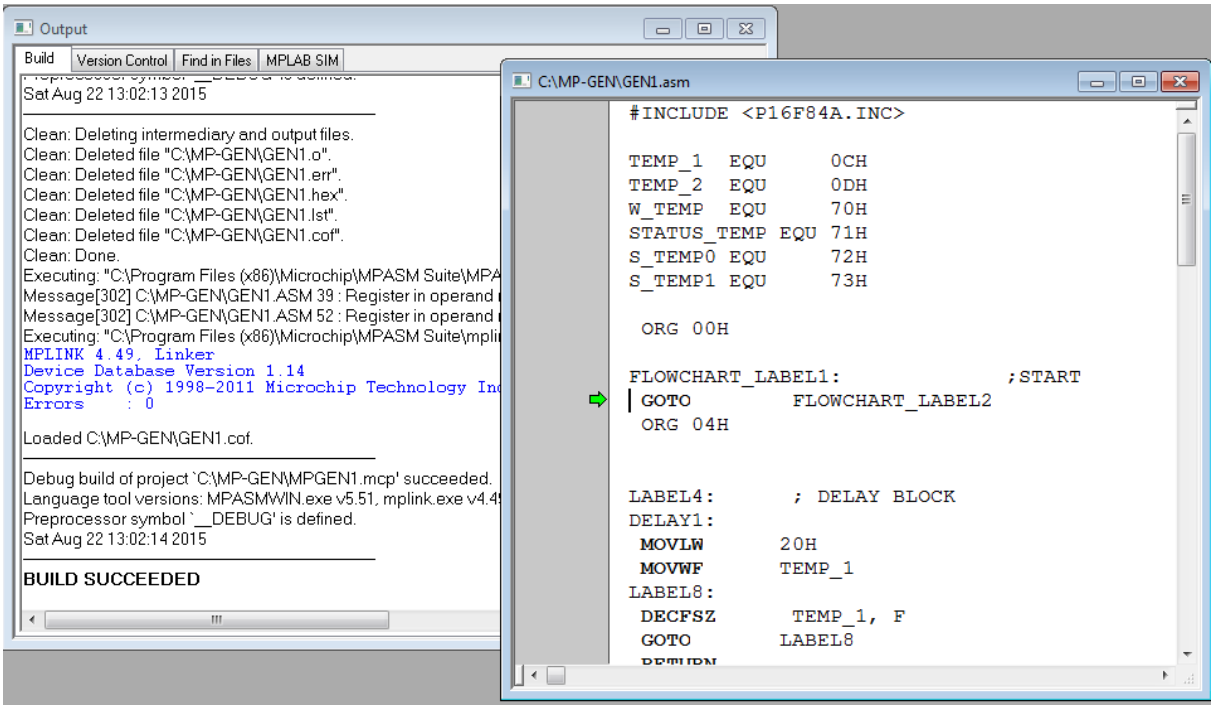


Рис.3.11. Выходное окно при успешной компиляции и окно отладки

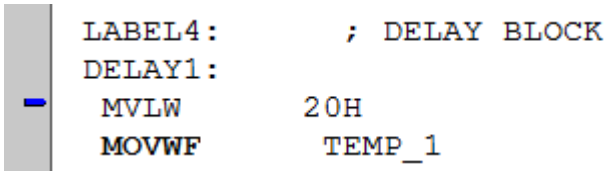


Рис.3.12. Ошибка в тексте программы

3.5. Испытание кода в симуляторе

Протестировать код можно в отладчике. Выбираем в главном меню *Debugger - Select Tool.* На вкладке выбираем *MPLAB SIM.* Появляются дополнительные пункты в меню *Debugger* и *View* (рис.3.13) и дополнительная панель отладчика (рис.3.14).

Сохраним созданное рабочее поле, выполнив *File-Save Workspace.*

Далее выполним *Debugger-Reset-Processor Reset* и зеленая стрелка установится на старте программы. Одиночный шаг программы выполняется выбором *Debugger-Step Into.*

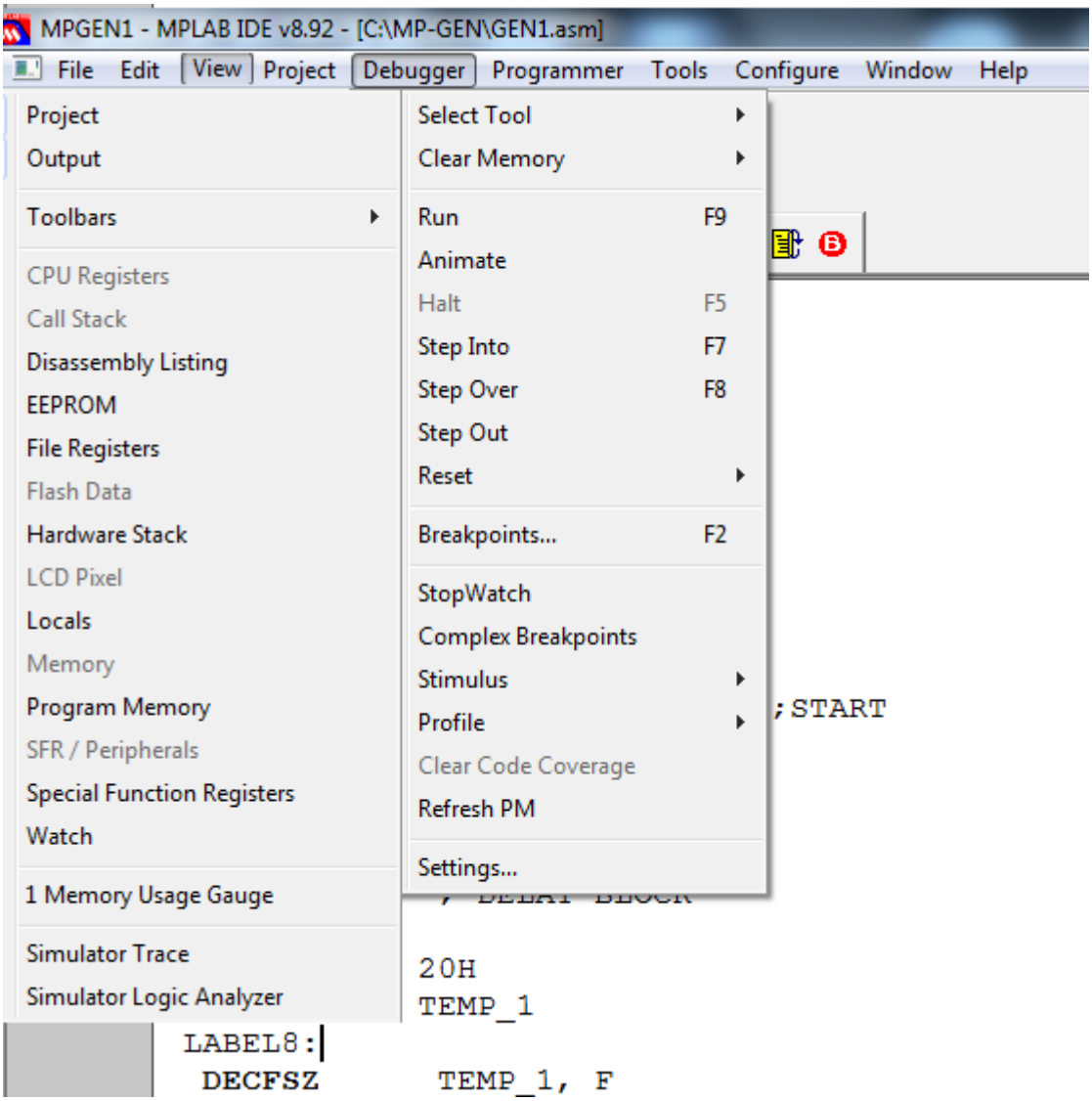


Рис.3.13. Дополнительные команды в режиме отладки

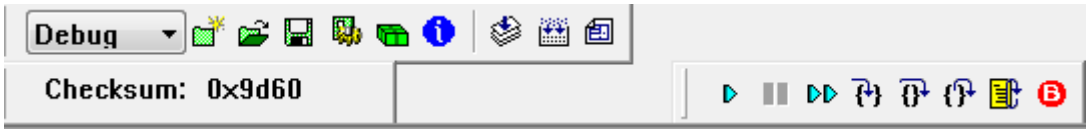


Рис.3.14. Окно отладки программы

Команды отладчика и горячие кнопки показаны в таблице 3.1.

В режиме *Run* после сброса выполняются участки программы до паузы.

В режиме *Animate* происходит пошаговое циклическое выполнение программы. В нашей программе можно наблюдать циклы задержки на метках `LABEL8` и `LABEL9` (Рис.3.15).

Чтобы проверить, работает ли код как требуется, наблюдают значения, которые записываются в порты. Выбираем *View - Watch* , чтобы открыть пустое окно наблюдения (рис.3.16). Вверху окна есть две кнопки. Левую кнопку *Add SFR* используют для добавления в окно регистров специального назначения и портов. Выберем *PortB* и добавим в окно наблюдения.

Таблица 3.1

Debugger Menu	Toolbar Buttons	Hot Key
Run		F9
Halt		F5
Animate		
Step Into		F7
Step Over		F8
Step Out		
Reset		F6

```
LABEL8:
    DECFSZ      TEMP_1, F
    GOTO        LABEL8
    RETURN
```

```
LABEL9:
    DECFSZ      TEMP_2, F
    GOTO        LABEL9
    RETURN
```

Рис.3.15. Циклы задержки на метках `LABEL8` и `LABEL9`

Правая кнопка позволяет добавлять обозначения регистров в окно наблюдения. Добавим в окно *TrisB*. В окне наблюдения мы видим адрес, имя и значение выбранных портов.

Можно еще раз выполнить *File - Save Workspace* и сохранить окно наблюдения.

Установим в программе точки останова после `LABEL8` и `LABEL9`. Для этого выделим строку, правой кнопкой мыши откроем меню и выберем *Set Breakpoint*. Программа будет останавливаться в точках прерывания (рис.3.16). Точки оснанова можно установить и снять двойным щелчком на строке с командой.

После первого прерывания *PortB* имеют значение 0x01, после второго прерывания значение равно 0x00. Это показывает, что программа работает правильно.

```
LABEL8:
 DECFSZ      TEMP_1, F
 GOTO        LABEL8
 RETURN
LABEL7:      ; DELAY BLOCK
DELAY2:
 MOVLW       20H
 MOVWF       TEMP_2
LABEL9:
 DECFSZ      TEMP_2, F
 GOTO        LABEL9
 RETURN
```

Рис.3.16. Точки прерывания в программе

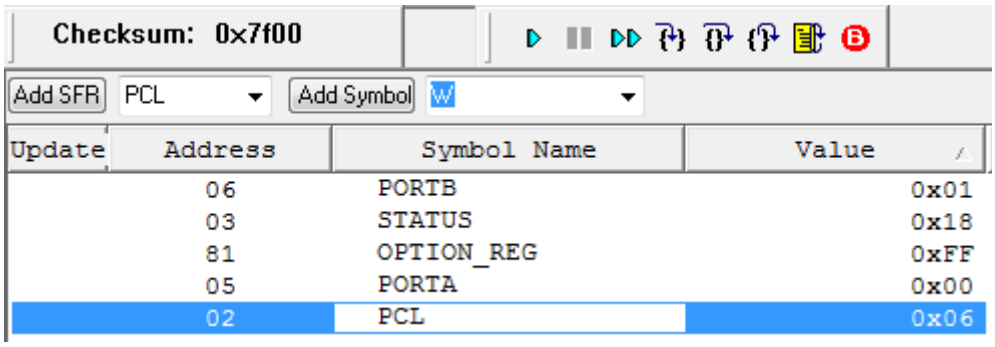


Рис.3.17. Окно наблюдения

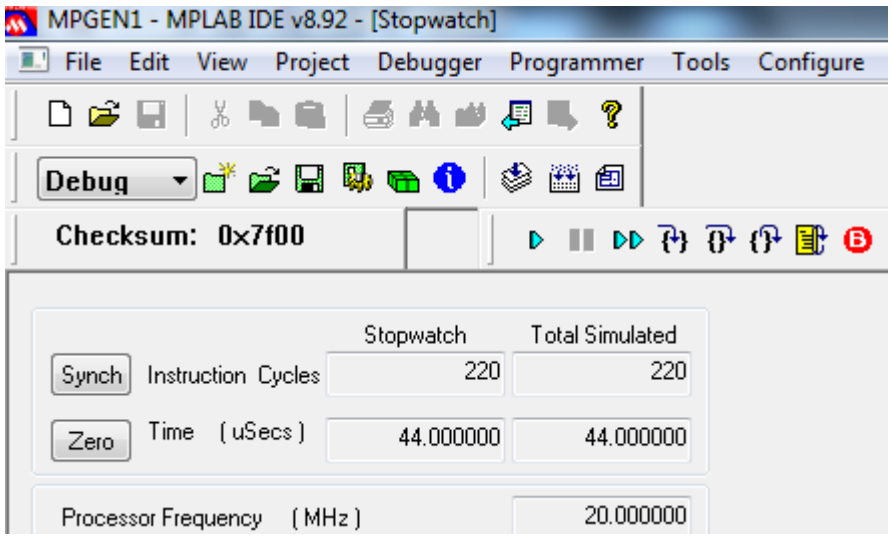


Рис.3.18. Определение времени исполнения программы

Для выхода из циклической программы применяют команду *Debugger-Step Out*.

Для определения времени выполнения программы можно использовать *Debugger-Stop Watch*.

Установим прерывание в конце программы, сделаем сброс и пуск. После остановки выбираем *Windows-Stopwatch* (рис.3.18).

При тактовой частоте процессора 20 МГц программа выполняется за 44 мкс. Отметим, что на тактовой частоте 4 МГц время выполнения будет в 5 раз больше и составит 220 мкс, а, следовательно, совпадет с периодом импульсного сигнала, измеренного в программе TINA (рис.2.18).

Более подробно с возможностями среды TINA мы познакомимся при выполнении лабораторных работ.

3.6. Лабораторная работа №1

Изучение программирования и отладки микроконтроллеров в средах TINA и MPLAB

Цель и содержание работы.

Изучение применения программ TINA и MPLAB IDE для проектирования и отладки устройств на микроконтроллерах. Простая программа будет составлена и отлажена с использованием блок-схемы в среде TINA. Затем файл программы в ассемблере будет загружен в MPLAL IDE, отредактирован, повторно отлажен и снова испытан в модели микроконтроллера в программе TINA.

Работа в среде TINA

1. Запустите программу TINA и соберите генератор чисел (рис.3.19) на микроконтроллере PIC16F84A. Установите тактовую частоту микроконтроллера 4МГц. Используя перемычки (джамперы), соедините выходы порта B микроконтроллера со входами дисплея *LCD Display* из меню *Meters*. При этом строго соблюдайте обозначения меток на перемычках. Неиспользованные входы дисплея рекомендуется подключить к низкому уровню.

Создайте папку, например, *TI-LAB1* и сохраните файл под именем *LAB1*.

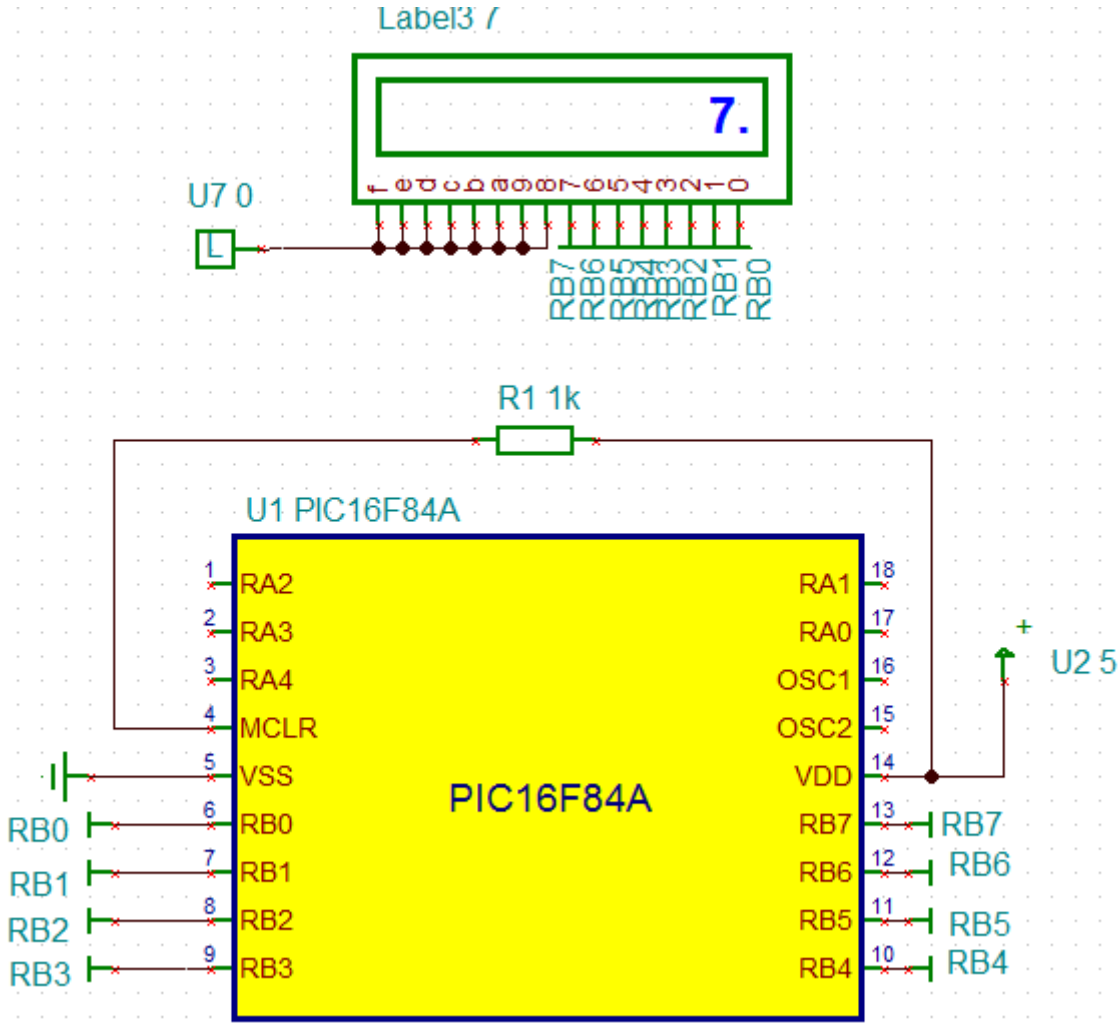


Рис.3.19. Модель генератора чисел

2. Установить в меню *Analysis-Options* опцию *Enable MCU Code debbuger.*

3. Для микроконтроллера открыть *Properties*, выбрать *MCU Input File Selection-Flowchart.*

Сформулируем очень простую задачу: генератор чисел должен поочередно выводить на дисплей два разных числа, заданных в исходных данных для бригады, например, числа 7 и 45.

Составить блок-схему программы (рис.3.20).

В окне первого блока *Output* установить число 7 и включить все биты. В окне первого блока *Delay* установить задержку 500

мкс. Во втором блоке *Output* установить целое число 45 (*2D* в коде *HEX*). Вторую задержку также сделать 500 мкс. Соединить все блоки в цикл.

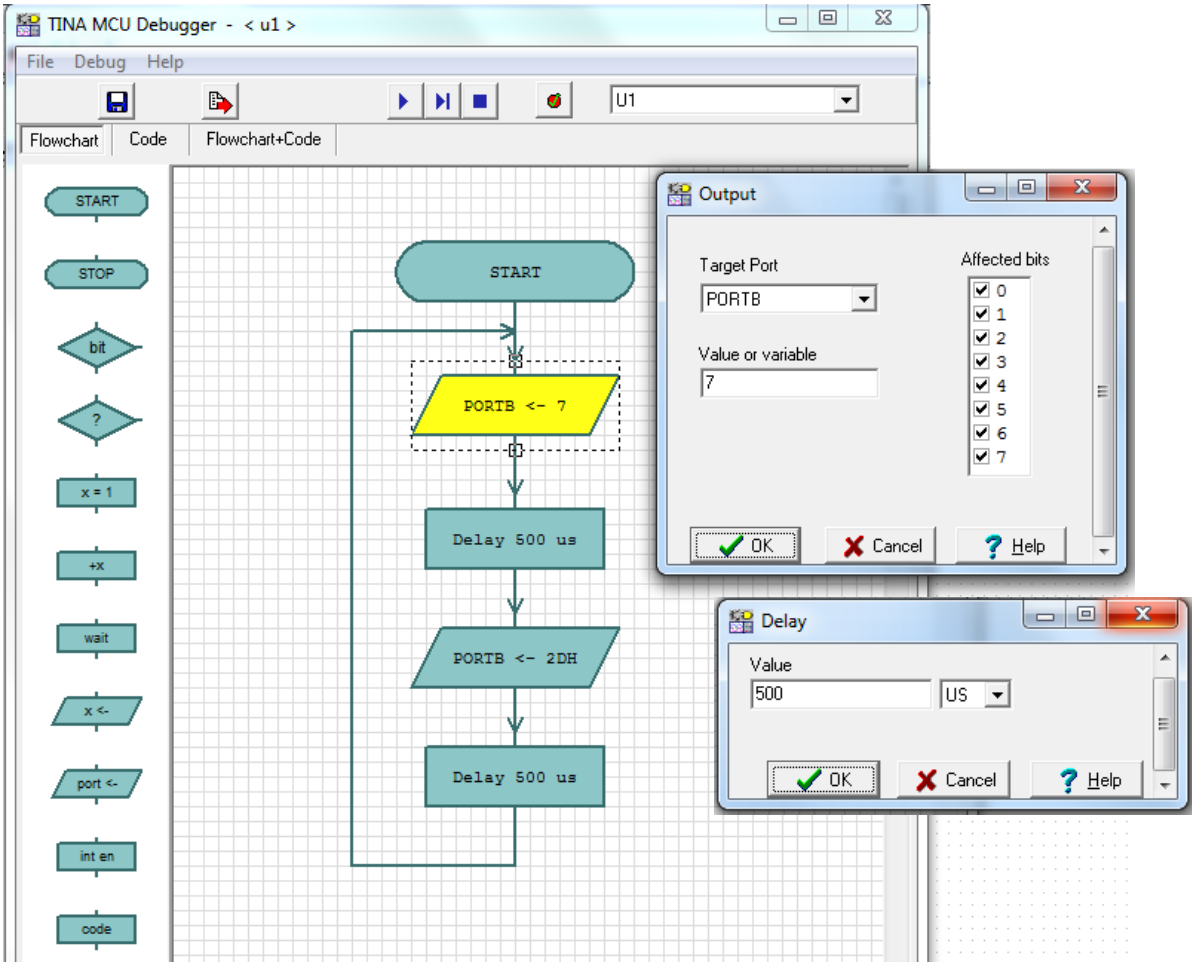


Рис.3.20. Блок-схема программы генератора чисел

После окончания сборки блок-схемы нажимаем -*Save To Macro*.

4. Проверить работу схемы. Для этого надо выбрать интерактивный режим *TR* и в меню *Interactive* нажать кнопку *Start*. Откроется окно отладчика с блок-схемой. Нажать *Run* и наблюдать изменение чисел на дисплее.

Запустить программу в пошаговом режиме. Наблюдать изменение чисел на дисплее.

5. В отладчике открыть окно *Code* (рис.3.21). Запустить программу в непрерывном и пошаговом режиме. Наблюдать зацикливание программы в определенных местах при

выполнении задержки (рис.3.22). Скопировать программу (например, в Notepad++) и объяснить работу программных блоков задержки.

6. В отладчике открыть окно *Flowchart+Code*. Наблюдать работу программы в непрерывном и пошаговом режиме (рис.3.23).

7. Найдите в программе команду `movlw 7` и установите на ней точку останова. Запустите программу в непрерывном режиме и убедитесь, что произойдет останов на данной команде.

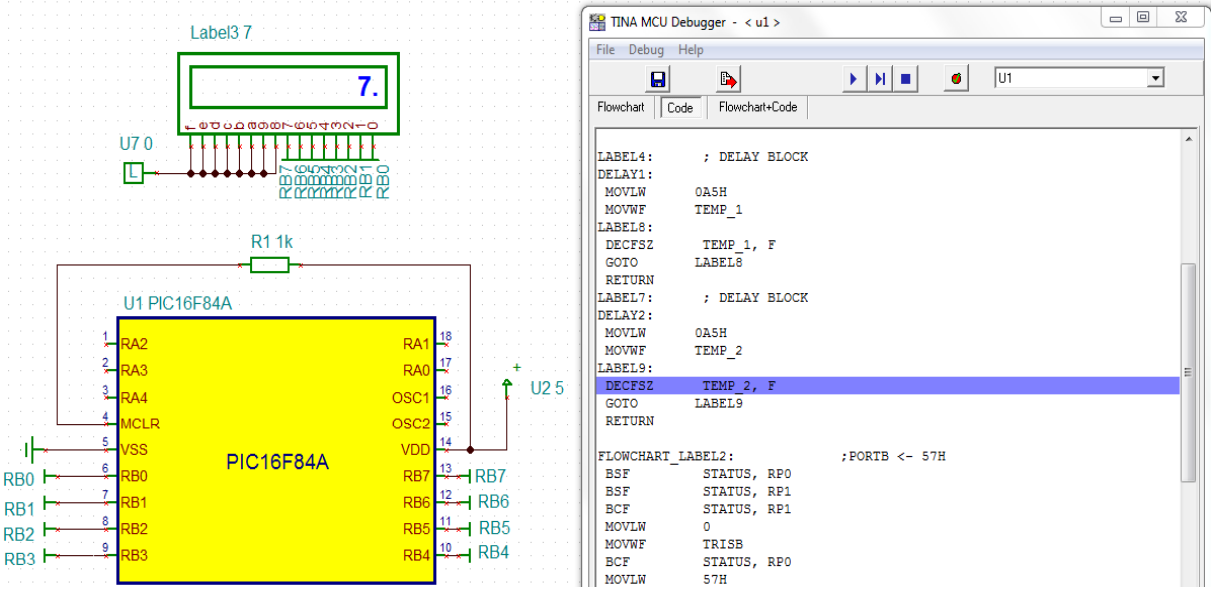


Рис.3.21. Отладчик среды TINA в режиме *Code*

8. В режиме *Flowchart* редактирование выполняется по блок-схеме. В отладчике вернитесь в окно *Flowchart*, во втором блоке в окне *Output* установите число 77, нажмите *Save To Macro* и снова запустите программу. Убедитесь в смене чисел на дисплее.

9. Выполните сохранение файлов программы. Для этого в отладчике в меню *File* выполните сохранение в папку *TI-LAB1* следующих файлов: *FL2.tfc, LAB1.asm, LAB1.hex, LAB1.lst.*

10. Откройте файлы с помощью программы *Notepad++* и сохраните файлы для последующего изучения и анализа.

```
FLOWCHART_LABEL1:              ;START
 GOTO        FLOWCHART_LABEL2
 ORG 04H


LABEL4:       ; DELAY BLOCK
DELAY1:
 MOVLW       0A5H
 MOVWF       TEMP_1
LABEL8:
 DECFSZ       TEMP_1, F
 GOTO        LABEL8
 RETURN
LABEL7:       ; DELAY BLOCK
DELAY2:
 MOVLW       0A5H
 MOVWF       TEMP_2
LABEL9:
 DECFSZ       TEMP_2, F
 GOTO        LABEL9
 RETURN
```

Рис.3.22. Программный блок задержки

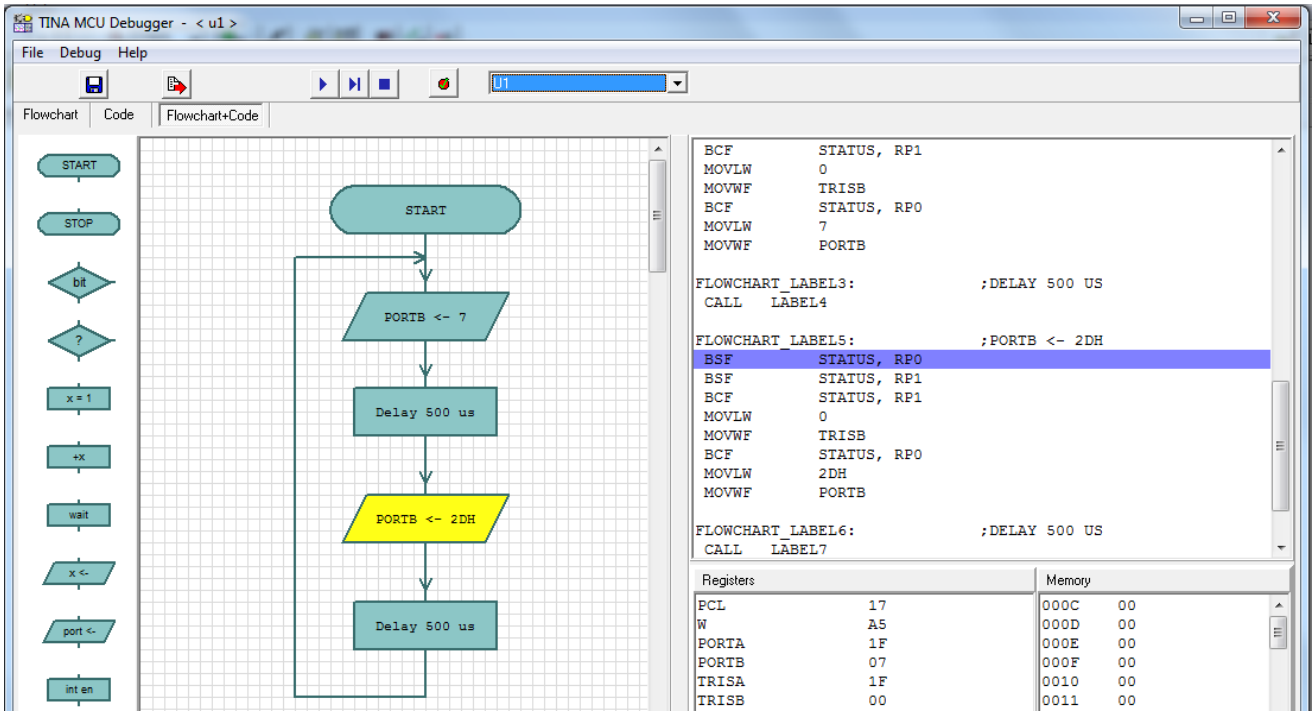


Рис.3.23. Окно отладчика *Flowchart+Code*

Работа в среде MPLAB IDE

11. Загрузите программу MPLAB IDE. В меню выбираем *Configure-Select device* и применяемый микроконтроллер PIC16F84A.

12. Создайте проект. Выбираем *Project- Project Wizard* и нажимаем *Next*. Убеждаемся, что в проекте используется PIC16F84A.

13. В окне установки инструментов языка выбираем *MPASM Assembler [mpasmwin.exe]v5.51* (рис.3.24)

14. Создаем проект и файл C:\MP-LAB1\LAB1.

15. Добавляем в проект файл *LAB1.asm* из папки *TI-LAB1*. Нажимаем три раза на букву перед именем файла на правой панели, чтобы появилось «С». Это позволит скопировать файл в директорию проекта. Нажимаем *Далее*.

Проверьте правильность созданного проекта и нажмите «Готово».

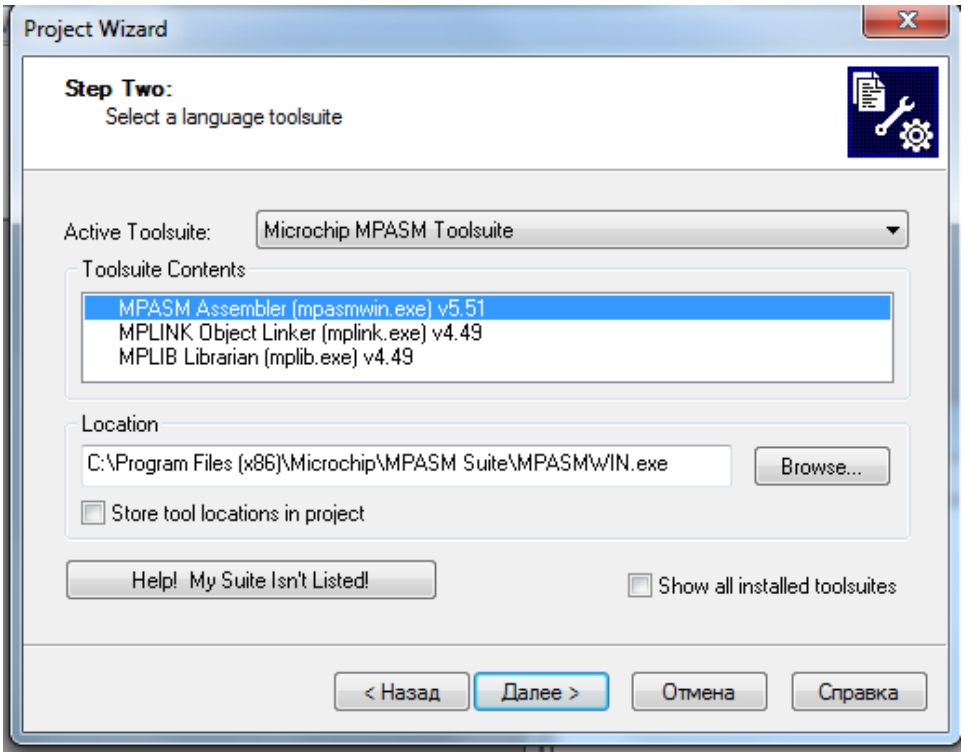


Рис.3.24. Выбор инструментов языка

16. Проведите сборку проекта. Выполните *Project – Build All* и добейтесь получения успешного результата.

17. Проведите тестирование полученного кода в отладчике. Для этого выбираем в главном меню *Debugger-Select Tool*. На вкладке выбираем *MPLAB SIM*. Появляются дополнительные

пункты в меню *Debugger*, *View* и дополнительная панель отладчика.

Запустите программу пошагово. Наблюдайте длительные циклы на метках `Lable8` и `Lable9`. Для выхода из цикла сделайте останов и выполните команду *Step out*. Зарегистрируйте фрагмент программы с циклом.

18. Проверка значения порта B. Остановим программу после `Lable8`. Выбираем *View-Watch* , далее устанавливаем *PortB* и нажимаем *Add SFR*. Видим на выходе порта B число 0x07 в формате hex, равное 7 (рис.3.25). Зарегистрируйте результат наблюдения.

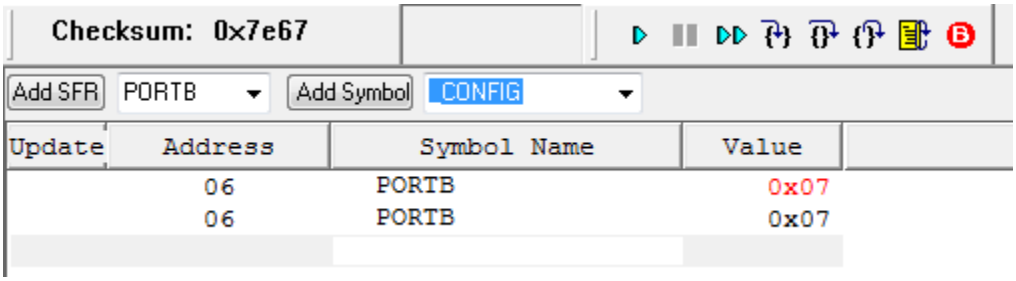


Рис.3.25. Просмотр портов в режиме *Watch*

Снова запустите программу в пошаговом режиме и остановите после `Lable9`. Выберите *Windows-Watch*. Теперь на выходе порта B будет число 0x4D, равное десятичному числу 77. Зарегистируйте результат наблюдения.

19. Сделайте редакцию программы. В команде `MOVLW 7` запишем число, равное сумме номера бригады и номера группы. В команде `MOVLW 4DH` запишите сумму номера бригады и даты выполнения лабораторной работы. Сохраните файл. Сделайте также *Save-Workspace*.

20. Выполните компиляцию отредактированного в ассемблере кода, сделав *Project-Build All*.

21. Посмотрите содержание папки C:\MP-LAB1 . Зарегистрируйте результат.

22. Определите время выполнения программы. Установите прерывание в конце программы. Выделите последнюю строку и на вкладке *Debugger* выберите *Set Breakpoint*.

Сделайте сброс: *Debugger-Reset-Processor Reset*. Выполните *Debugger-Stop Watch* и выберите *Windows-Stop watch.*

На тактовой частоте процессора 20 МГц определите время выполнения программы и зарегистрируйте результат.

Проверка исправленного кода в программе TINA

23. Загрузите программу TINA и откройте файл *C:\TI-LAB1\Lab1*. Для микроконтроллера откройте *Properties*, выберите *MCU Input File Selection.* Последовательно выбирайте и загрузите из папки C:\MP-LAB1 три новых файла: *LAB1.asm, LAB1.hex, LAB1.lst.*

24. Выберите интерактивный режим *TR* и нажмите *Start.* В меню отладчика нажмите *Run* и убедитесь, что на дисплее появляются новые числа. Значит программа отредактирована и скомпилирована правильно.

Обратите внимание, что после загрузки нового HEX – файла, в режиме *Flowchart* открывается пустое окно. В него можно загрузить сохраненный файл *FL2.tfc* из папки *C:\TI-LAB1*, но изменения в этот файл не переходят. Кроме того, после загрузки старого файла *FL2.tfc* стираются новые отредактированные файлы и требуется их повторная загрузка.

Домашнее задание

1. Распечатать листинг программы генератора чисел.

2. Используя документацию на микроконтроллер PIC16F84A, написать подробные комментарии к программе генератора чисел.

3. Составить для PIC16F84A программу, отображающую на дисплее последовательность чисел от 0 до 255.

4. Придумать, составить и отладить свою оригинальную программу для PIC16F84A. Составить подробное описание отладки в MPLAB IDE и TINA.

Контрольные вопросы

1. Назовите назначение и состав программной среду MPLAB IDE.

2. Перечислите особенности среды MPLAB IDE.

3. Назовите последовательность шагов при создании проекта в MPLAB IDE.

4. Как можно добавить готовый файл в проект и как его сохранить в проекте ?

5. Как установить биты конфигурации для микроконтроллера ?

6. Как выполнить компиляцию программы ?

7. Как найти и исправить ошибки в программе ?

8. Как выполнить проверку и отладку программы ?

9. Какие кнопки используют при отладке ?

10. Как установить и снять точки останова в программе ?

11. Как можно использовать окно наблюдения Watch ?

12. Как определить время выполнения программы и ее части ?

Глава 4. ПРАКТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ НА АССЕМБЛЕРЕ

В этой главе мы подробно изучим систему команд на ассемблере для микроконтроллера PIC16F84A, научимся программировать и моделировать устройства с внешними управляющими компонентами, выполняющие арифметические и логические операции, циклы задержки, прерывания, использующие таймеры и EEPROM.

4.1. Лабораторная работа №2

Изучение системы команд микроконтроллера PIC16F84A на языке ассемблера

Цель работы: изучение системы команд на ассемблере, запись текста программ и отладка в MPLAB IDE, создание модели микроконтроллера в программе TINA, загрузка программ в модель и проверка работы.

Для изучения системы команд микроконтроллера PIC16F84A в лабораторной работе используется программа «Тренажер-ASM», составленная с использованием методики из [15]. Блок-схема программы показана на рис.4.1.

Правила оформления программы

Текст программы традиционно пишется в три колонки от левого края. Каждая колонка занимает 12 знакомест.

В первый столбец записывают определяемые имена и метки.

Во второй столбец – команды и директивы.

В третий столбец – имена регистров и числа.

Комментарии записывают после символа "точка с запятой". Комментарии полезны для облегчения работы с программой.

Для лучшего зрительного восприятия блоки программы можно друг от друга отделять пустыми строками.

В шапке программы указывается:

- тип используемого микроконтроллера;

Блок-схема программы «Тренажер-ASM»

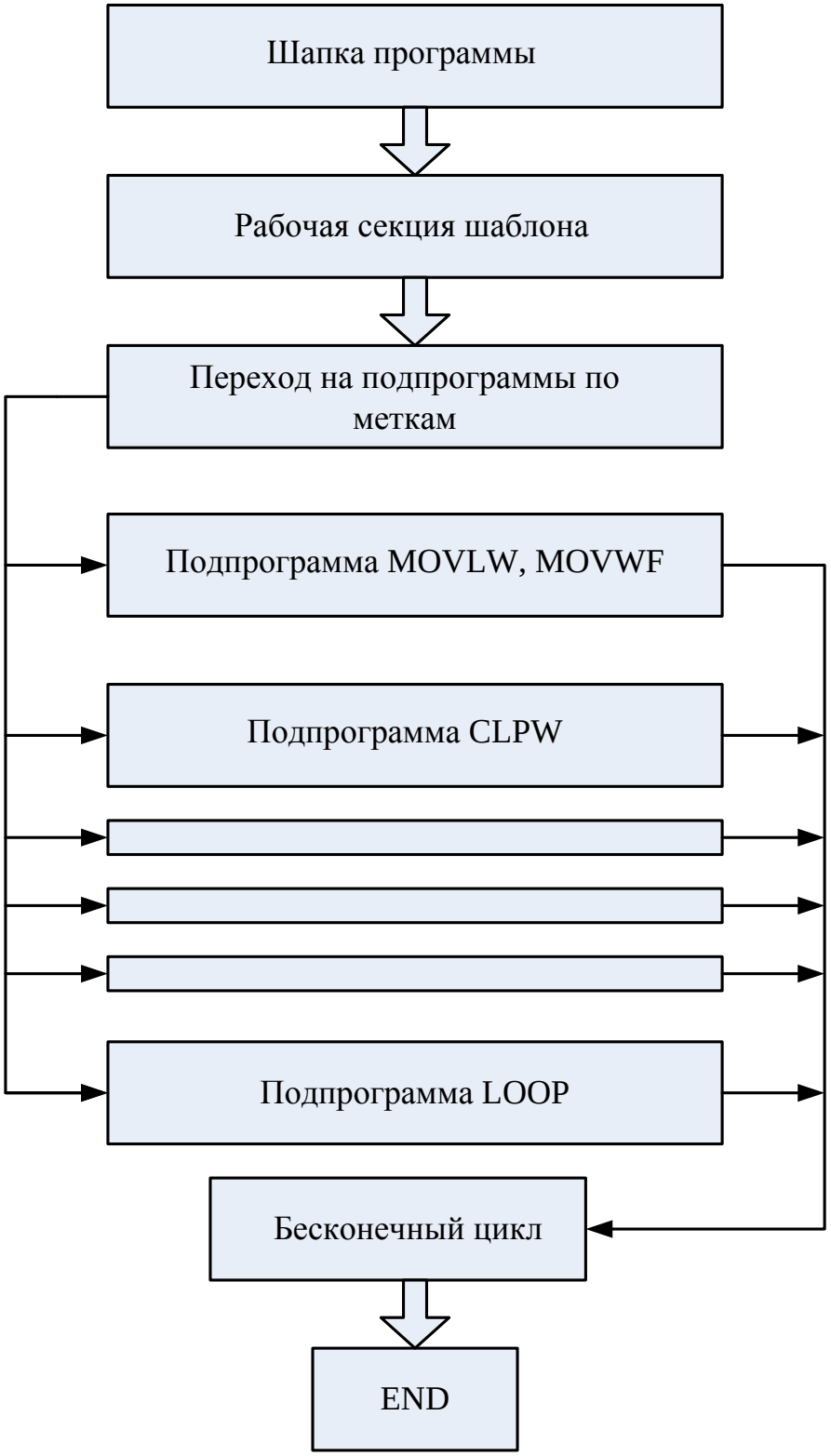


Рис.4.1. Блок-схема программы «Тренажер ASM»

- его конфигурация (директива __CONFIG);

-перечисляются имена регистров и их адреса, сопоставленные числам (директива EQU);

- ячейки ОЗУ;

- константы и т.п.

(В MPLAB IDE биты конфигурации устанавливаются на вкладке *Configure-Configuration Bits* и директиву _ CONFIG в программе можно не записывать).

Между шапкой и телом программы стоит директива ORG , определяющая начальный адрес программы в памяти программ.

Затем идёт тело программы, которое определяет логику работы микроконтроллера.

В самом конце идёт директива END , определяющая конец программы.

Порядок работы с программой «Тренажер-АСС»

1. Набрать в Notepad++ шаблонный файл *PICOD-TMP.asm,* представленный на листинге 4.1.

Листинг 4.1

Шаблонный файл *PICOD-TMP.asm*

```
LIST P=PIC16F84A
#include <p16F84a.inc>    ; подключение файла
                          ; описания микроконтроллера
                     ;описание операционных регистров
TMR0        EQU      01h
PC          EQU      02h
STATUS      EQU      03h
FSR         EQU      04h
                          ; регистры ввода-вывода
CNT         EQU      05h
DATAPORT    EQU      06h

                      ;ячейки озу
SCRATCH     EQU      0Ch
DIGIT       EQU      0Dh
                             ;биты регистра STATUS
C           EQU      0h
```

```
DC          EQU       1h
Z           EQU       2h
PD          EQU       3h
TO          EQU       4h
RP          EQU       5h
                            ; управляющие регистры
TRISA       EQU       85h
TRISB       EQU       86h
                        ; слова инициализации портов
INITA       EQU       B'00000000'
INITB       EQU       B'00000000'

            ; РАБОЧАЯ СЕКЦИЯ ШАБЛОНА

            ORG       0
            GOTO       BEGIN

            ORG        100h
BEGIN

                          ;инициаоизация порта A
            BCF        STATUS,RP   ;выбор банка 0
            CLRF       CNT      ;очистить регистр CNT
            MOVLW      INITA    ;загрузить
                               ;B'00000000' в региср W
            BSF        STATUS,RP     ;выбор банка 1
            MOVWF      TRISA        ;все разряды
                           ;порта A установить на выход

                          ;инициаоизация порта B
            BCF         STATUS,RP      ;выбор банка 0
            CLRF        DATAPORT      ;очистить
                                      ;регистр DATAPORT
            MOVLW       INITB     ;загрузить
                             ; B'00000000' в регистр W
            BSF         STATUS,RP     ;выбор банка 1
            MOVWF       TRISB    ;все разряды порта B
                                ; установить на выход
            BCF         STATUS,RP      ;выбор банка 0
Select1     GOTO        Label1    ;выбор
                                       ; подпрограмм
```

```
                ;БЛОК ПОДПРОГРАММ

Label1      MOVLW       B'01010101' ;загрузить
                         ;01010101 в регистр W
N1          NOP
            MOVWF       DATAPORT ;записать W в порт B
                                 ;(DATAPORT)
N2          NOP

            GOTO        Label40


Label40     GOTO    $   ;зациклиться навсегда


            END
```

2. Загрузить программу MPLAB IDE и создать проект с названием *PICODE* в папке C:\MP-LAB2\PICOD, пользуясь указаниями из главы 3.

3. Добавить в проект готовый шаблонный файл *PICOD-TMP.asm* из папки *PICLAB*. Не забудьте добавляя файл в проект в правом окне четвертого шага *Project Wizard* установить букву C.

3. Выполнить компиляцию командой *Project-Build All* . В случае успешной компиляции в папке C:\ MP-LAB2 будут файлы *PICOD-TMP* с расширениями *ASM, *HEX, *LST.

4. Изучить содержание программы и протестировать код в отладчике. Для этого выбираем в главном меню *Debugger-Select Tool*. На вкладке выбираем *MPLAB SIM*. Появляются дополнительные пункты в меню *Debugger*, *View* и дополнительная панель отладчика.

В шапке программы регистрам специального назначения сопоставляются ячейки памяти от 01h до 06h. При этом PORTA эквивалентен имени CNT, PORTB эквивалентен DATAPORT. Назначены две ячейки ОЗУ с адресами 0Ch и 0Dh. Определены биты регистра STATUS. Символу `C` присвоено значение 0h, поскольку `C` это нулевой бит слова состояния STATUS. Каждый

раз, когда надо будет проверить бит 0h, мы будем пользоваться предварительно определенным символом `C`. Каждый раз, когда мы захотим обратиться к биту 2h (ZERO), мы будем использовать символ `Z` вместо 02h.

Управляющие регистры TRISA и TRISB занимают ячейки 85h и 86h. Эти установки соответствуют описанию области оперативной памяти PIC16F84A.

Заданы слова инициализации портов INITA и INITB в виде восьмиразрядных нулей в двоичной системе.

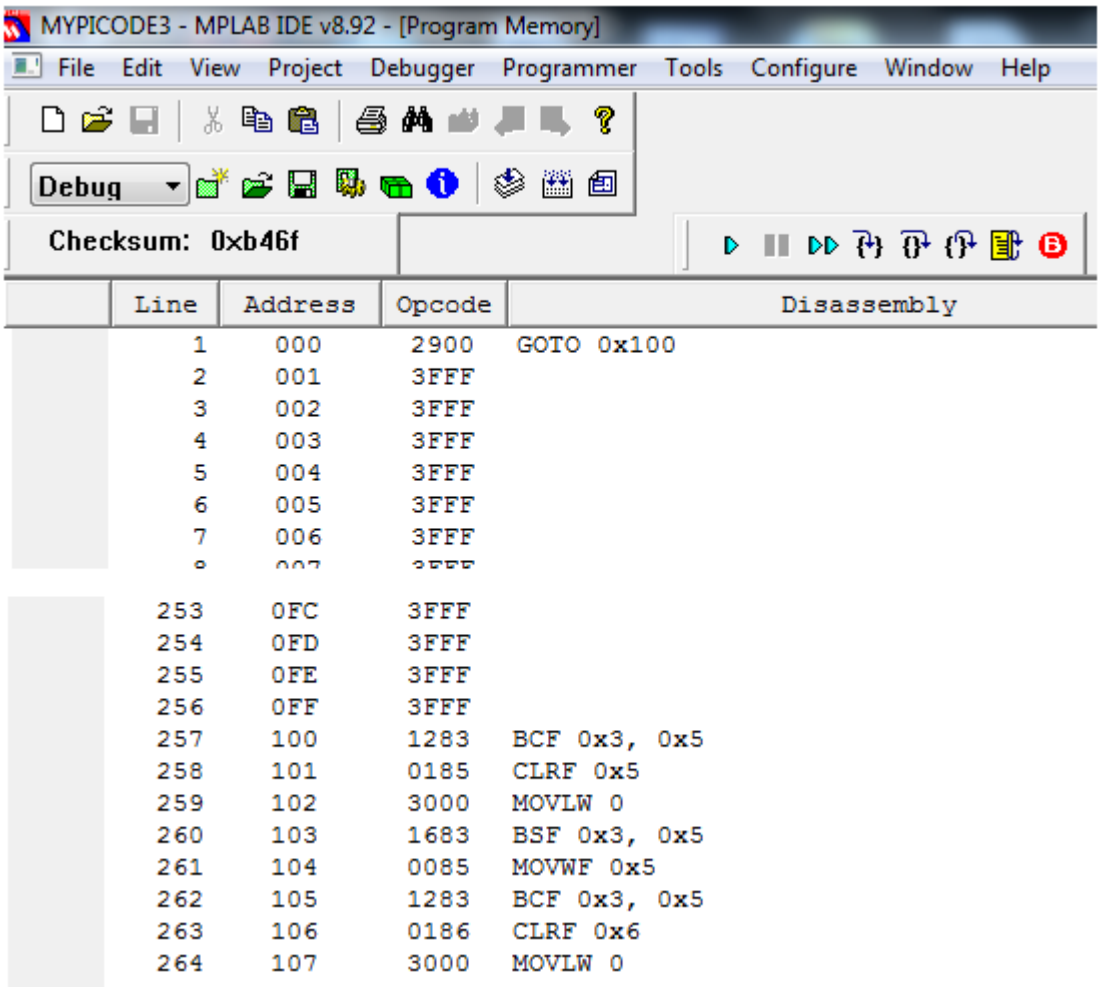


Рис.4.2. Фрагменты памяти программ в MPLAB

В рабочей секции шаблона выражение `ORG 0` задает начальный адрес кода программы в памяти программ. При включении микроконтроллера работа начинается с адреса 0h. Второе выражение `ORG 100` и команда `GOTO BEGIN`

определяет, что продолжение программы начнется с адреса 100h.

Для проверки этого в меню MPLAB IDE выберите *VIEW-Program memory*. Фрагменты программы показаны на рис.4.2.

Команды `BCF STATUS,RP` и `BSF STATUS,RP` нужны для переключения между банками памяти. Вся память данных микроконтроллера разбита на два банка. Банку 0 соответствуют адреса 00h..7F, банку 1-8F..FF. Выбор банка определяется состоянием бита 5 в регистре STATUS. Когда этот бит установлен в `1`, выбран банк 1, иначе - банк 0.

Так как PORTA и CONT находятся в банке 0, сначала командой `BCF STATUS RP` устанавливаем 0 в бит RP и выбираем банк 0. Командой `CLRF CONT` выполняем очистку регистра CONT.

Содержание регистров специального назначения можно проверить, выбрав *View-Special Function Registers* (Рис.4.3).

Address	SFR Name	Hex
	WREG	0x00
00	INDF	--
01	TMR0	0x00
02	PCL	0x3C
03	STATUS	0x1C
04	FSR	0x00
05	PORTA	0x00
06	PORTB	0x00
08	EEDATA	0x00
09	EEADR	0x00
0A	PCLATH	0x00
0B	INTCON	0x00
81	OPTION_REG	0xFF
85	TRISA	0x00
86	TRISB	0x00
88	EECON1	0x00
89	EECON2	0x00

Рис.4.3. Просмотр регистров специального назначения

Содержание статического ОЗУ отображается по команде *View-File Registers* (Рис.4.4).

Содержание памяти данных EEPROM отображается по команде *View-EEPROM* (Рис.4.5).

Командой `MOVLW INTA` загружаем B’00000000’ в регистр W.

Командой `BSF STATUS RP` устанавливаем `1` в бит RP и выбираем банк 1, в котором находится регистр TRISA.

Следующей командой `MOVLF TRISA` все разряды PORTA установлены на выход.

В результате получили, что в исходном состоянии порт А работает на выход и содержит на всех выходах нули.

Аналогично происходит инициализация порта В.

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	--	00	3C	1C	00	00	00	--	00	00	00	00	04	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
60	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
70	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
80	--	FF	3C	1C	00	00	00	--	00	00	00	00	04	00	00	00
90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
E0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
F0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Рис.4.4. Просмотр содержания ОЗУ

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
10	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
30	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....

Рис.4.5. Просмотр содержания EEPROM

Далее по команде `Select1 GOTO Label1` происходит переход на выполнение подпрограмм для изучения конкретной команды. Изменяя номер метки, можно переходить на любую подпрограмму. После выполнения подпрограммы происходит переход на метку Label40 и программа зацикливается по команде `GOTO $`.

5. Проверку выполнения команд мы будем делать на модели микроконтроллера со светодиодами в программе TINA.

Загрузить в компьютере программу TINA. Собрать схему (рис.4.6). Для наглядного размещения светодиодов в схеме использованы соединительные джамперы J0-J7.

В свойствах микроконтроллера установить частоту 4МГц.

В меню *Analysis-Options* установить *Enable MCU Code debbuger* и *Enable VHDL mixed mode.* В главном меню установить интерактивный режим *Transient.*

Используя редактор текста  обязательно подписать схему, указав группу, бригаду, исполнителей и дату.

Далее мы приступаем к изучению команд ассемблера для *PIC*-контроллеров.

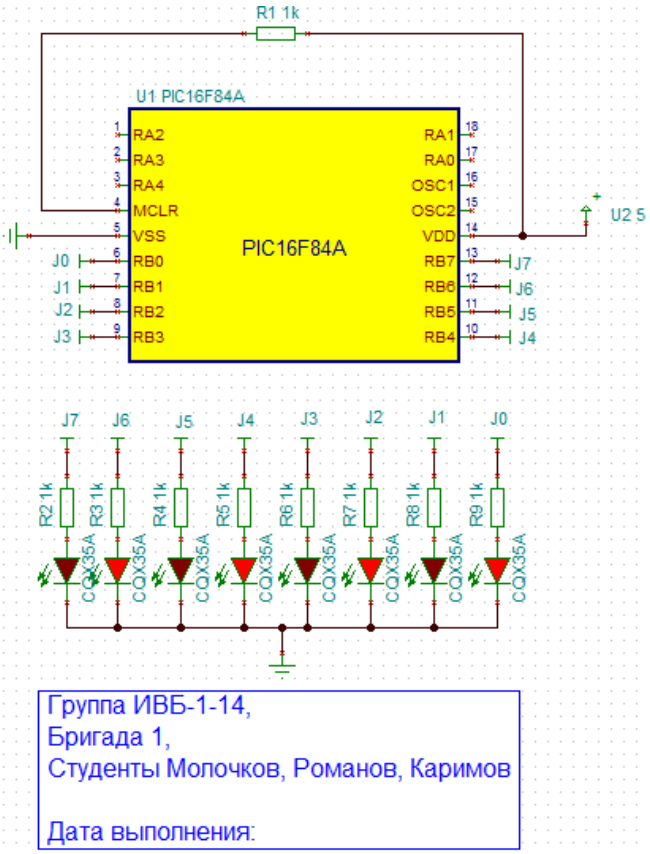


Рис.4.6. Схема модели со светодиодами

6. Команды MOVLW и MOVWF

Открыть проект в MPLAB-IDE. Выбрать *Debugger-Select Tool- MPLAB SIM.* В шаблонном файле (листинг 4.1) уже записана первая подпрограмма с меткой LABEL1 для изучения команд MOVLW и MOVWF.

Команда MOVLW пересылает константу B’01010101’ в регистр W.

Команда MOVWF пересылает содержимое регистра W в регистр F, которым в данной команде является DATAPORT.

Выполнить *RUN* и при необходимости повторно выполнить компиляцию (*Build All*). После этого можно исследовать работу программы.

В MPLAB-IDE на метках N1 и N2 сделать точки останова, выполнив *Set Breakpoints* .

Выполнить сброс процессора (*Debuger-Reset- Processor reset*) и затем *Run*.

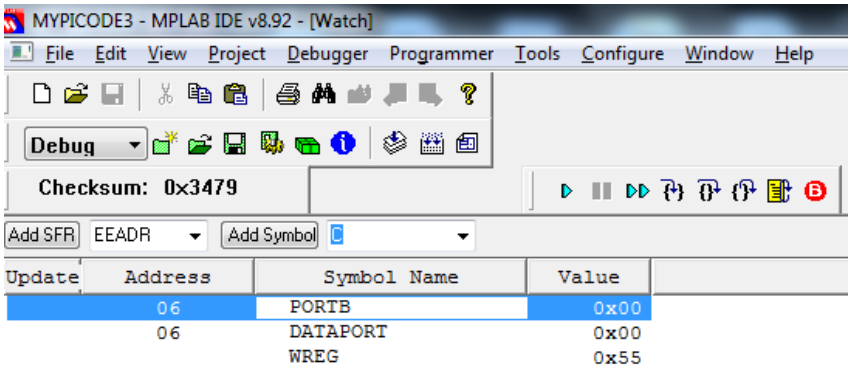


Рис.4.7. Просмотр содержания портов до загрузки

При остановке на метке N1 просмотреть содержание регистров. Для этого выполнить: *View- Watch.*

На рис. 4.7 PORTB и DATAPORT содержат нули, а регистр W содержит шестнадцатиричное число 0x55, соответствующее двоичному B’01010101’.

При остановке на метке N2 три регистра содержат число B’01010101’, которое выдается на выходы PORTB (рис.4.8).

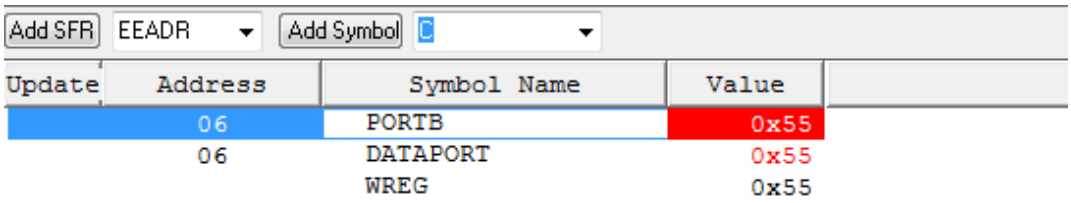


Рис.4.8. Содержание портов после загрузки

Зарегистрировать результаты наблюдения.

Выполнить проверку работы микроконтроллера с подпрограммой Lable1.

В модели программы TINA для микроконтроллера PIC16F84A открыть *Properties-MCU[HEX.LIST File name]*, выбрать *HEX/LIST file – Select HEX* и загрузить файлы *PICOD-TMP.hex* и *PICOD-TMP.asm* из папки C:/MP-LAB2.

Запустить программу в интерактивном режиме, нажав  и *Start*. В окне *Debugger* можно наблюдать выполнение программы. Сделать скриншот и зарегистрировать состояние диодов на выходе порта B (рис.4.9).

Рис.4.9. Моделирование команд MOVLW и MOVWF

Внимание !

В отчет надо включить:

- текст подпрограммы Lable1;
- описание сущности команд;
- содержание регистров и переменных (рис.4.7 и 4.8);
- состояние светодиодов (рис.4.9)

7.Команда NOP

Это команда «Нет операции». Используется в циклах временной задержки. Особой проверки для этой команды не требуется.

8. Команда CLRW

Эта команда очищает рабочий регистр W. Составим подпрограмму Label2 (листинг 4.2):

Листинг 4.2

```
Label2    MOVLW   B'01010101'   ;загрузить 01010101
                                 ; в регистр W
          CLRW                 ;очистить регистр W
          MOVWF   DATAPORT  ;записать W в порт B
                            ; (DATAPORT)
          GOTO    Label40
```

Каждую новую подпрограмму мы будем добавлять в шаблонный файл *PICOD-TMP.asm* и изменить команду `Select1 GOTO Label**XX**`, где XX – номер метки подпрограммы.

Выполнить компиляцию и проверку на модели контроллера. Зарегистрировать результаты.

9. Команда CLRF f.

Очистить содержимое регистра f и установить флаг Z=1. Добавим подпрограмму Label3 (листинг 4.3):

Листинг.4.3

```
Label3     MOVLW      B'01010101'     ; загрузить
                                    ;01010101 в регистр W
           MOVWF      DATAPORT  ;записать W в порт B;
           CLRF       DATAPORT  ;очистить порт B
N3         GOTO          Label40
```

Выполнить компиляцию и отладку в MPLAB IDE. При остановке по метке N3 выбрать *Watch* и проверить содержание регистров и переменных (рис.4.10). Убедиться в том, что регистр DATAPORT очищен, регистр STATUS содержит значение 0x1C или B’00011100’. Второй бит равен 1, следовательно, выставлен флаг Z=1.

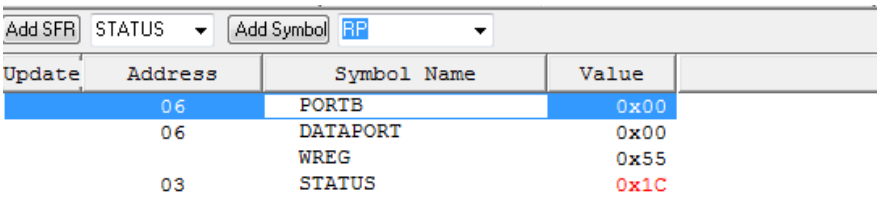


Рис.4.10. Проверка содержания регистров

10. Команда SUBWF f,d

Позволяет вычесть регистр W из любого регистра f . Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f. Добавить в шаблонный файл п/п Label4 (листинг 4.4):

Листинг.4.4

```
Label4      MOVLW    0FFh    ; загрузить 0FFh в
                               ; регистр W
            MOVWF    DATAPORT ; записать W в порт B
            MOVLW    01h   ; загрузить 01h в регистр W
            SUBWF    DATAPORT,1 ; выполнить вычитание
N4          GOTO     Label40
```

В регистр W загружается 0FFh, соответствующее B’11111111’. Это число записываем в DATAPORT. В регистр W загружаем 01h и вычитаем эту единицу из DATAPORT. Результат сохраняем в DATAPORT.

При остановке в точке N4 проверить содержание регистров (рис.4.11):

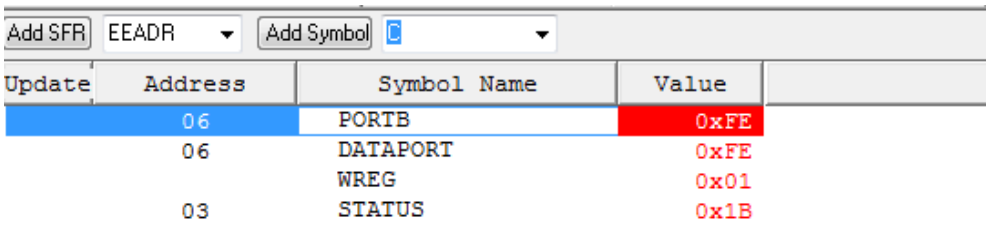


Рис.4.11. Содержание регистров для команды SUBWF f,d

Регистр DATAPORT содержит 0xFE или B’11111110’.

Выполнить моделирование и проверить включение светодиодов. Зарегистрировать результаты.

11. Команда ADDWF f,d

Прибавляет рабочий регистр W к любому регистру f и устанавливает флаги C, DC, Z в регистре STATUS. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.

Выполнить отладку п/п Label5 отладку в MPLAB IDE и моделирование в TINA.

Листинг.4.5

```
Label5      MOVLW   0h   ; загрузить 0h в регистр W
            MOVWF   DATAPORT; записать W в порт B
            MOVLW   01h  ; загрузить 01h в регистр W
            ADDWF   DATAPORT,1   ; выполнить сложение
N5          GOTO     Label40
```

Проверить и объяснить состояние регистров и светодиодов.

12. Команды SUBLW k, ADDLW k

Эти две команды работают аналогично вышеописанным, за тем исключением, что операция производится между рабочим регистром W и байтовой константой, заданной в команде. Команда SUBLW вычитает рабочий регистр W из константы k, а команда ADDLW добавляет рабочий регистр W к константе k. Эти команды также устанавливают признаки C, DC, Z.

Выполнить отладку и моделирование п/п Label6, зарегистрировать и объяснить результаты.

Листинг.4.6

```
Label6      MOVLW      05h   ; загрузить 05h в W
            MOVWF     DATAPORT  ; записать W в порт B
            SUBLW      0FFh   ; вычесть из 0FFh
                       ;содержимое рабочего регистра
            MOVWF     DATAPORT; загрузить новое
                              ; содержимое в DATAPORT
            GOTO      Label40
```

13. Команды DECF f,d, INCF f,d

Команда DECF уменьшает заданный регистр f на 1, а INCF увеличивает заданный регистр f на 1. Результат может быть помещен обратно в регистр f (при d=1), либо в рабочий регистр W (при d=0). В результате выполнения этих команд может установиться признак Z в регистре STATUS.

Записать в шаблонный файл п/п Label7 с командой DECF, выполнить отладку и моделирование, объяснить результаты.

Листинг 4.7

```
Label7      MOVLW     0FFh   ; загрузить 0FFh в
                             ; регистр W
            MOVWF     DATAPORT ; записать W в порт B
```

```
            DECF      DATAPORT,1    ; уменшить
                                    ; DATAPORT на 1
            GOTO      Label40
```

Записать п/п Label8 с командой INCF, выполнить отладку и моделирование, объяснить результаты.

Листинг.4.8

```
Label8      CLRF    DATAPORT      ; очистиь DATAPORT
            INCF    DATAPORT,1    ; увеличить DATAPORT
                                  ; на 1
            GOTO    Label40
```

14. Команды IORWF f,d, ANDWF f,d, XORWF f,d

Три команды выполняют логические действия ИЛИ, И и ИСКЛЮЧАЮЩЕЕ ИЛИ. Операцию логического сложения ИЛИ можно использовать для установки отдельных битов в регистрах. Сбрасываются эти биты затем операцией логического умножения И. Когда над одинаковыми битами выполняется операция ИСКЛЮЧАЮЩЕЕ ИЛИ, результат равен 0. Поэтому операцию ИСКЛЮЧАЮЩЕЕ ИЛИ можно использовать для проверки состояния (установлены или сброшены) определенных битов в регистре.

В этих командах, если d=0, результат сохраняется в W. Если d=1, результат сохраняется в f.

Записать, отладить и смоделировать п/п Label9 установки бит 1 в порте B при помощи команды IORWF (листинг 4.9):

Листинг 4.9

```
Label9      CLRF    DATAPORT      ; очистить порт B
            MOVLW   B'00000010'   ; установить маску
                                  ; в регистре W
            IORWF   DATAPORT,1    ; установить биты
                                ; в порте B по маске W
            GOTO    Label40
```

Следующая п/п Label10 (листинг 4.10) позволяет сбросить биты при помощи команды ANDWF. Записать, отладить и смоделировать эту п/п.

Листинг 4.10

```
Label10     MOVLW   B'11111111'   ; загрузить 0FFh
                                  ; в регистр W
```

```
            MOVWF   DATAPORT      ; установить все
                                  ;биты в порте B
            MOVLW   B'00000101'   ; установить маску
                                  ; в регистре W
            ANDWF   DATAPORT,1    ; очистить биты в
                                 ; порте B по маске W
            GOTO    Label40
```

Пусть требуется проверить равен ли регистр SCRATCH значению 04h. Используем команду XORWF (листинг 4.11):

Листинг. 4.11

```
Label11     MOVLW    04h  ; загрузить 04h в регистр W
            MOVWF    DATAPORT ; загрузить регистр W
                    ; в порт B(DATAPORT)
            MOVWF    SCRATCH  ; загрузить регистр W
                              ; в SCRATCH
            XORWF    SCRATCH,0    ; проверить
                              ; равенство W и SCRATCH
            MOVWF    DATAPORT   ; загрузить регистр W
                              ; в порт B(DATAPORT)
            GOTO      Label40
```

Записать, отладить и смоделировать эту п/п. Поскольку SCRATCH и W равны, результат выполнения операции XORWF равен нулю (все светодиоды не горят). В регистре STATUS установится бит Z, который реальная программа затем может проверить и обработать.

15. Команды IORLW k, ANDLW k, XORLW k

Эти три команды выполняют те же действия, что и их три предыдущие, за тем исключением, что операция производится между рабочим регистром W и константой – маской k, заданной в команде. Результат выполнения команды помещается в рабочий регистр W.

Команда IORLW k

Листинг 4.12

```
Label12      MOVLW    010h  ; загрузить 010h
                            ;в регистр W
             IORLW    09h   ; установить 0-й и
```

```
                    ;3-й биты
          MOVWF     DATAPORT  ; загрузить регистр W
                              ; в порт B (DATAPORT)
                     GOTO          Label40
```

Команда ANDLW k

Листинг 4.13

```
Label13     MOVLW     0FFh   ; загрузить 0FFh
                             ;в регистр W
            ANDLW     040h   ; оставить 6-й бит
            MOVWF     DATAPORT  ; загрузить регистр W
                              ; в порт B(DATAPORT)
            GOTO       Label40
```

Команда XORLW k

Листинг 4.14

```
Label14     MOVLW     B'00100000'; загрузить 20h
                               ;в регистр W
            XORLW     B'11111111'; проинвертировать W
            MOVWF     DATAPORT ; загрузить регистр W
                              ; в порт B(DATAPORT)
            GOTO      Label40
```

Записать, отладить и смоделировать три подпрограммы, зарегистрировать в отчете состояние регистров и светодиодов после выполнения п/п и дать объяснение.

16.Команда MOVF f ,d

Эта команда используется для пересылки регистра f в рабочий регистр W (d=0). Если установить d=1, то эта команда загрузит регистр сам в себя, но при этом бит Z в регистре STATUS установится в соответствии с содержимым регистра.

Например, мы хотим загрузить 0Fh в регистр SCRATCH, а потом загрузить регистр SCRATCH в рабочий регистр W.

Для этого выполняется п/п Label15 (листинг 4.15):

Листинг 4.15

```
Label15     MOVLW     0Fh  ; загрузить 0Fh в регистр W
            MOVWF     SCRATCH  ; загрузить регистр W
                               ; в SCRATCH
```

```
        CLRW              ; очистить W
        MOVF    SCRATCH,0 ; загрузить SCRATCH
                          ; в регистр W
        MOVWF   DATAPORT  ; записать W в порт B
                          ; (DATAPORT)
        GOTO    Label40
```

Записать, отладить и смоделировать п/п Label15.

Если в процессе выполнения программы требуется проверить регистр DATAPORT на ноль, мы можем выполнить следующую команду:

```
        MOVF    DATAPORT,1  ; записать W в порт B
```

Бит Z регистра STATUS будет установлен, если условие будет выполнено (DATAPORT = 0h).

18. Команда COMF f,d

Эта команда инвертирует все биты в регистре f. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.

Записать, отладить и смоделировать п/п Label16 (листинг 4.16). Дать пояснения.

Листинг 4.16

```
Label16 MOVLW   B'01010101'  ; загрузить
                     ; '01010101' в регистр W
        MOVWF   DATAPORT  ; загрузить регистр W
                          ; в DATAPORT
        COMF    DATAPORT    ; инвертировать
                            ; DATAPORT
        GOTO    Label40
```

19. Команды DECFSZ f,d, INCFSZ f,d

При d=1 команда DECFSZ уменьшает на единицу, а INCFZ увеличивает на единицу заданный регистр f и пропускает следующую команду, если регистр стал равным нулю. При d=0 результат записывается в регистр W и следующая команда пропускается, если рабочий регистр W стал равным нулю. Эти команды используются для формирования временных задержек, счетчиков, циклов и т.д.

Записать, отладить и смоделировать подпрограмму организации цикла Label17 (листинг 4.17) . Описать мигание светодиодов.

Листинг 4.17

```
Label17

START
          MOVLW    0FFh  ; загрузить 0FFh в регистр W
          MOVWF    SCRATCH ; загрузить регистр W
                           ; в SCRATCH
LOOP
          DECFSZ    SCRATCH,1; уменьшить SCRATCH на 1
          GOTO    LOOP ; и переходить обратно, пока
                         ; не станет =0
          MOVF    DIGIT,0; загрузить регистр DIGIT
                           ; в W
          MOVWF   DATAPORT  ; вывести на светодиоды
          DECF    DIGIT,1   ; уменьшить DIGIT на 1
          GOTO    START    ; перейти на начало
          GOTO    Label40
```

Команда DECFSZ здесь работает в цикле задержки, состоящем из двух команд : DECFSZ и GOTO LOOP. Поскольку мы предварительно загрузили в регистр SCRATCH значение 0FFh, этот цикл выполнится 255 раз, пока SCRATCH не станет равным нулю. При тактовой частоте 4 МГц это дает задержку 1 мксек/команду * 2 команды * 255 = 510 мксек.

Когда SCRATCH станет равным нулю, выполняются команды с регистром DIGIT. Его значение уменьшается на единицу и цикл повторяется от метки START.

В результате регистр DIGIT перебирает все значения за 256 циклов.

Проанализируйте работу подпрограммы в MPLAB-IDE. Установите *Breakpoint* на команде GOTO START (рис.4.12). Сделайте Reset, Run. При каждой остановке наблюдайте содержание регистров (рис.4.13).

Аналогичную п/п можно использовать и с командой INCFSZ, заменив загружаемое в регистр SCRATCH значение с

FFh на 0h. Светодиоды будут мигать точно так же и если заменить команду DECF на команду INCF.

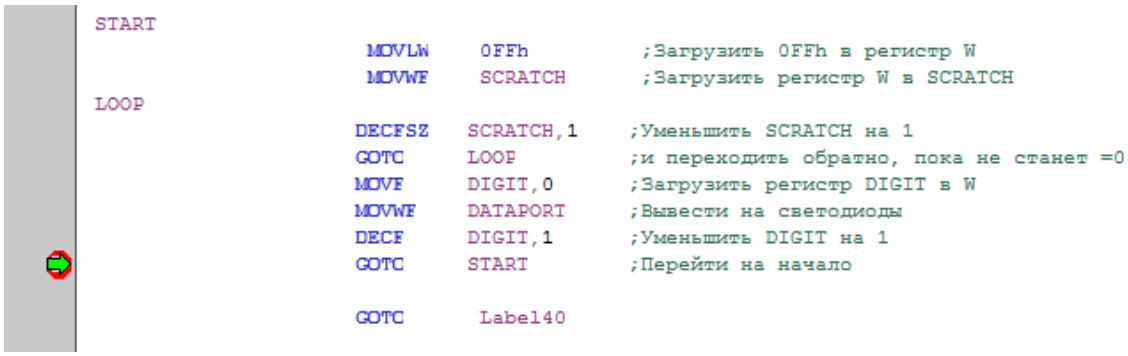


Рис.4.12

20. Команда SWAPF f,d

Команда меняет местами полубайты в любом регистре. При d=0 результат записывается в рабочий регистр W, а при d=1 остается в регистре.

Записать, отладить и смоделировать п/п Label18 (листинг 4.18), в которую включен фрагмент п/п Label17, создающий задержку перед выполнением команды SWAPF.

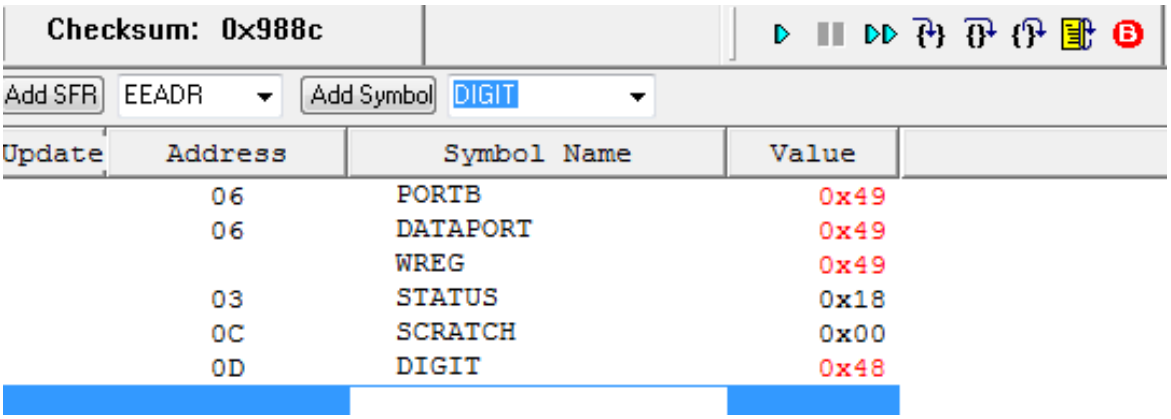


Рис.4.13

Листинг 4.18

```
Label18   MOVLW   B'00001111'  ; загрузить 00001111
                                ;в регистр W
          MOVWF   DATAPORT  ; загрузить регистр W
                            ; в DATAPORT
          ; подпрограмма задержки из Label17
```

```
          MOVLW    0FFh  ; загрузить 0FFh в
                         ; регистр W
          MOVWF    SCRATCH ; загрузить регистр W
                           ; в SCRATCH
LOOP1
          DECFSZ   SCRATCH,1; уменьшить SCRATCH на 1
          GOTO     LOOP1 ; и переходить обратно, пока
                         ; не станет =0
                   ; окончание подпрограммы задержки
          SWAPF    DATAPORT,1   ; поменять полубайты
          GOTO     Label40
```

Объяснить состояние светодиодов.

21. Команды RRF f,d и RLF f,d

Имеется две команды сдвига:

RRF f,d - сдвиг вправо содержимого любого регистра f через бит C регистра статус и

RLF f,d - сдвиг влево содержимого любого регистра f через бит C регистра статус.

При d=0 результат сдвига записывается в регистр W, а при d=1 остается в регистре.

Инструкции сдвига используются для выполнения операций умножения и деления, для последовательной передачи данных и для других целей. Во всех случаях бит, сдвигаемый из 8-битного регистра, записывается в бит C в регистре STATUS, а бит C записывается в другой конец регистра, в зависимости от направления сдвига. При сдвиге влево (RLF) бит C записывается в младший бит регистра, а при сдвиге вправо (RRF) C записывается в старший бит регистра.

Записать, отладить и смоделировать две подпрограммы Label19 (листинг 4.19) и Label20 (листинг 4.20). Проверить содержание регистров.

Листинг 4.19

```
Label19     BCF    STATUS,0 ; очистить бит 0(CARRY)
                            ; в регистре STATUS
            MOVLW  0FFh   ;загрузить 0FFh в регистр W
            MOVWF  DATAPORT  ; загрузить регистр W
                             ; в DATAPORT
```

```
            RRF     DATAPORT,1; сдвинуть вправо
            GOTO          Label40
```

Листинг 4.20

```
Label20     BCF     STATUS,0 ; очистить бит 0(CARRY)
                             ; в регистре STATUS
            MOVLW   0FFh ; загрузить 0FFh в регистр W
            MOVWF   DATAPORT  ; загрузить регистр W
                              ; в DATAPORT
            RLF     DATAPORT,1   ; сдвинуть влево
            GOTO       Label40
```

22. Команды BCF f,b, BSF f,b

Команды очистки бита BCF и установки бита BSF используются для работы с отдельными битами в регистрах. Параметр b означает номер бита, с которым производится операция, и может принимать значения от 0 до 7.

Записать, отладить и смоделировать п/п Label21 с командой BCF f,b (листинг 4.21).

Листинг 4.21

```
Label21     MOVLW   0FFh  ;Загрузить 0FFh в регистр W
            MOVWF   DATAPORT ;Загрузить регистр W
                             ; в DATAPORT
            BCF     DATAPORT,7   ;Очистить бит 7
                                 ; в порте B
            GOTO    Label40
```

23. Команды BTFSC f,b, BTFSS f,b

Команды условных переходов BTFSC и BTFSS проверяют состояние заданного бита b в любом регистре f и в зависимости от результата пропускают (не исполняют) или исполняют следующую команду. Команда BTFSC пропускает команду, если заданный бит сброшен, а команда BTFSS - если установлен.

Записать, отладить и смоделировать п/п Label22 (листинг 4.22):

Листинг 4.22

```
Label22     MOVLW   0FFh ; загрузить 0FFh в регистр W
            MOVWF   DATAPORT; включить светодиоды
```

```
            MOVLW   B'00000001'; загрузить 00000001
                                ;в регистр W
            MOVWF   CNT  ; загрузить регистр W в CNT
LOOP2
            BTFSS   CNT,0  ; проверить бит 0 в CNT
            GOTO    LOOP2  ; ждать пока бит 0
                           ; не установится
            BCF     DATAPORT,7 ; выключить светодиод
            GOTO    Label40
```

Ранее мы упоминали о возможности проверки битов состояния в регистре STATUS. Это также делается при помощи команд BTFSS и BTFSC.

Проверка бита C:

```
     ; Проверка бита C
     BTFSS   STATUS,C  ; если C установлен,
                       ; пропустить GOTO
     GOTO    WHERE_EVER   ;
```

Аналогично проверяется бит Z:

```
    ; Проверка бита Z
     BTFSS   STATUS,Z  ; если Z установлен,
                       ; пропустить GOTO
     GOTO    WHERE_EVER   ;
```

Эти команды часто используются в программах.

24. Команды CALL k, RETURN

Эти две команды предназначены для работы с подпрограммами. Команда CALL используется для перехода на подпрограмму по адресу, задаваемому в команде, а команда RETURN - для возврата из подпрограммы. Обе команды выполняются за 2 цикла. Адрес, на котором находилась команда CALL, запоминается в специально организованных регистрах, называемых стеком. Эти регистры недоступны для обращений и используются только при вызовах подпрограмм и возвратах. Глубина стека, т.е. число специальных регистров равно 8. Поэтому из основной программы можно сделать не более 8 вложенных вызовов подпрограмм. После возврата из подпрограммы выполнение продолжается со следующей после

CALL команды. Регистр W и регистр STATUS при вызове подпрограммы не сохраняются, поэтому, если необходимо, их можно сохранить в отдельных ячейках памяти.

Записать, отладить и смоделировать пример п/п Label23 (листинг 4.23):

Листинг 4.23

```
Label23
START2
          BSF    DATAPORT,7   ; включить светодиод
          CALL    PAUSE  ; вызвать подпрограмму
                         ; задержки
          BCF     DATAPORT,7   ; выключить светодиод
          CALL    PAUSE  ; вызвать подпрограмму
                         ; задержки
          GOTO    START2 ; перейти на начало
 ; подпрограмма задержки
PAUSE
          MOVLW    02h   ; загрузить 02h в регистр W
          MOVWF   SCRATCH  ; загрузить регистр W
                          ; в SCRATCH
          MOVLW    02h   ; загрузить 02h в регистр W
          MOVWF    DIGIT ; загрузить регистр W в
                          ;DIGIT
LOOP3
          DECFSZ   SCRATCH,1; уменьшить SCRATCH на 1
          GOTO     LOOP3; и переходить обратно, пока
                        ; не станет =0
          DECFSZ   DIGIT,1  ; уменьшить DIGIT на 1
          GOTO     LOOP3    ; и переходить на
                   ; метку LOOP3, пока не станет =0
          RETURN   ; вернуться из подпрограммы
          GOTO     Label40
```

В результате светодиод будет мигать. PAUSE - подпрограмма формирования паузы. Измените подпрограмму так, чтобы ускорить мигание светодиода примерно в два раза. Зарегистрируйте результаты.

25. Команды RETLW k, RETFIE

Существуют еще две команды, предназначенные для возврата из подпрограмм. Команда RETLW возвращает в рабочем регистре W константу, заданную в этой команде.

Записать, отладить и смоделировать п/п Label24 (листинг 4.24):

Листинг 4.24

```
Label24     CALL   SHOWSYM   ; вызвать подпрограмму
            MOVWF  DATAPORT  ; вывести элемент таблицы
                             ; в порт B
            GOTO    Label40  ; зациклиться навсегда
SHOWSYM
            RETLW    081h    ; записать 081h в W и
                       ; вернуться из подпрограммы
```

Команда RETFIE выполняет возврат из подпрограммы обработки прерываний. Содержимое вершины стека выгружается в счетчик команд PC. Осуществляется глобальное разрешение прерываний (бит GIE устанавливается в 1).

26. Специальные команды CLRWDT и SLEEP

Команда CLRWDT предназначена для сброса сторожевого таймера. Эта команда должна присутствовать в таких участках программы, где время выполнения программы между двумя соседними командами CLRWDT может превышать время срабатывания сторожевого таймера.

Команда SLEEP предназначена для перевода процессора в режим пониженного энергопотребления. После выполнения этой команды тактовый генератор процессора выключается и обратно в рабочий режим процессор можно перевести либо по входу сброса, либо по срабатыванию сторожевого таймера, либо по прерыванию.

Домашнее задание

1. Оформить отчет, который должен содержать полный , текст программы, скриншоты моделирования в TINA, обсуждение результатов и выводы.

2. Составить программу, в которой светодиоды включаются последовательно, начиная с нулевого, с циклом в модели 1 с.

3. Составить программу, в которой все светодиоды мигают одновременно с циклом 1с.

4. Составить программу «бегущего огня», в которой последовательно включаются светодиоды: 0-4, 1-5, 2-6, 3-7.

5. Составить программу, в которой циклически выполняются программы из предыдущих пунктов домашнего задания.

Контрольные вопросы

1. Назовите команды пересылки и объясните, как они выполняются.

2. Какая команда выполняет очистку рабочего регистра ?

3. Какая команда может очистить любой регистр ?

4. Назовите арифметические команды сложения и объясните их работу.

5. Назовите арифметические команды вычитания и объясните их работу.

6. Какие две команды выполняют операции декремента содержания регистра и в чем их различие ?

7. Какие команды выполняют операции инкремента содержания регистра и в чем их различие ?

8. Назовите команды, выполняющие операцию И и объясните их особенности.

9. Назовите команды, выполняющие операцию ИЛИ и объясните их особенности.

10. Назовите особенности команд, выполняющих операцию ИСКЛЮЧАЮЩИЕ ИЛИ.

11. Какая команда инвертирует все биты в регистре ?

12. Какая команда может поменять полубайты в регистре ?

13. Какие команды выполняют сдвиг содержимого регистра и как они работают ?

14. Какие команды выполняют очистку и установку бита в регистре ?

15. Какие команды выполняют проверку и пропуск следующей инструкции ?

16. Какие команды применяют для переходов и работы с подпрограммами ?

17. Назовите команды контроля и управления микроконтроллером.

4.2. Лабораторная работа №3

Программирование микроконтроллера с внешним управлением

Цель работы: изучение основ практического программирования микроконтроллеров с внешним управлением для выполнения простейших задач. С помощью ключей мы сможем управлять светодиодами, выводить на дисплеи числа и т.п.

Для этого создадим модель лабораторного макета (рис.4.14) и сохраним файл в папке TI-LAB3.

В макете использован микроконтроллер PIC16F84A. Частота генератора установлена равной 4 МГц. Линии RA0 – RA3 порта А работают на ввод. К ним подключены коммутирующие ключи SW-0 – SW-3, которые служат для реализации внешнего управления микроконтроллером. При замыкании ключа на вход подается сигнал высокого уровня. Линия RA4 порта А работает на вывод и подключена к светодиоду LED1. Линии RB0 – RB7 порта В работают на вывод к подключены к индикаторам (к двум HEX - дисплеям и LCD – дисплею). Линия RB4 ключом SW-4 может соединяться со светодиодом LED2.

Рассмотрим подробно программу модели лабораторного макета.

В шапке программы (листинг 4.25) задан тип микроконтроллера и система исчисления (R=HEX), используемая по умолчанию.

Директивами EQU регистрам специального назначения и константам, использованным в программе, сопоставлены их адреса в оперативной памяти, номера битов регистров, численные значения. Это позволит более компактно записывать текст программы.

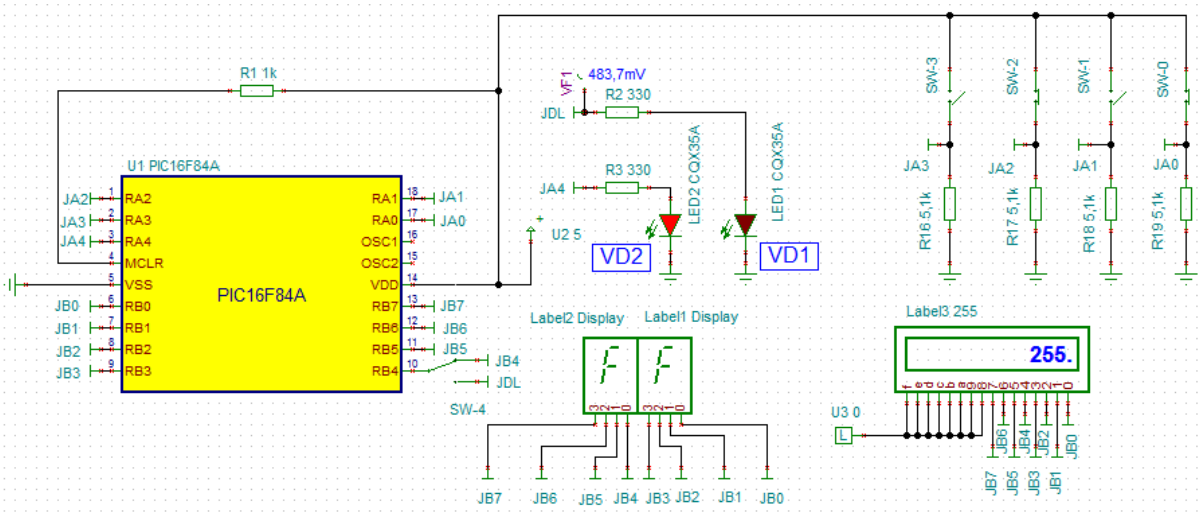


Рис.4.14. Модель лабораторного макета

Листинг 4.25

```
LIST  P=PIC16F84A,  R=HEX  ; директива, определяющая
    ;тип процессора и систему счисления по умолчанию
    ; описание используемых переменных и адресов
    ; ячеек для хранения переменных пользователя
#INCLUDE <pic16F84a.inc>
INTCON      EQU      0x0B
TMRO        EQU      0x01
INTF        EQU      1
TOIF        EQU      5
PCL         EQU      0x02
STATUS      EQU      0x03
OPTION_REG  EQU      0x81
RP0         EQU      5
IRP         EQU      7
PORTA       EQU      0x05
PORTB       EQU      0x06
TRISA       EQU      0x85
TRISB       EQU      0x86
W           EQU      0
F           EQU      1
TEMPA       EQU      0x0C
TEMPB       EQU      0x0D
COUNT1      EQU      0x0E
COUNT2      EQU      0x0F
COUNT3      EQU      0x10
```

Можно вместо описания регистров специального назначения микроконтроллера и управляющих битов этих регистров использовать шаблон программы для данного микроконтроллера.

Для этого последовательно открываем в MPLAB IDE или Notepad файл: C:\Program Files(x86)\Microchip\MPASM Suite\Template\Object\16F84ATMPO.

Этот файл содержит указание, что требуется файл описания микроконтроллера `Files required: p16F84A.INC` и директива #INCLUDE <pic16F84a.inc>, подключающая модуль описания микроконтроллера, описание конфигурации, начальные адреса основной программы и подпрограммы прерывания, шаблон подпрограммы прерываний Interrupt. Файл описания микроконтроллера <p16F84a.inc> содержит адреса регистров специального назначения и положение их управляющих битов.

Запишем в программе директиву #INCLUDE <pic16F84a.inc>, но для того, чтобы лучше запомнить регистры специального назначения мы будем каждый раз перечислять их в шапке программы.

Вернемся к продолжению программы, начатой в листинге 4.25.

Директивы #DEFINE определяют, какую строку записи заменяет та или иная введенная метка, привязанная к схеме модели (листинг 4.26). Так SW0 соответствует линии RA0 порта A.

Директива ORG 0x00 устанавливает начальный стартовый адрес программного кода равным нулю. Команда GOTO BEGIN вместе с директивой ORG 0x005 и меткой BEGIN выполняют переход на адрес памяти 0x005, с которого размещается основная программа.

Листинг 4.26

```
;         Определение меток текста
#DEFINE    Z      STATUS,2  ; бит нулевого результата
#DEFINE    VD1    PORTB,4     ; светодиод VD1
#DEFINE    VD2    PORTA,4     ; светодиод VD2
#DEFINE    SW0    PORTA,0     ; тумблер SW0
```

```
#DEFINE     SW1    PORTA,1      ; тумблер SW1
#DEFINE     SW2    PORTA,2      ; тумблер SW2
#DEFINE     SW3    PORTA,3      ; тумблер SW3


;исполняемая программа
            ORG    0x000
            GOTO   BEGIN
            ORG    0x005

BEGIN
            CALL   INIT_PORTS  ; вызов подпрограммы
                               ; инициализации портов
            GOTO   Label1
```

Далее командой CALL INIT_PORTS происходит переход в подпрограмму инициализации портов (листинг 4.27). Вначале подпрограмма устанавливает порты в единичное состояние, чтобы исключить возможную неопределенность. Затем команда BSF STATUS,5 в 5-й бит регистра STATUS записывает единицу и происходит переход в банк 1 оперативной памяти, где размещены регистры TRISA и TRISB. В регистр TRISA записываем константу 0x0F=B’00001111’. В результате линии RA0-RA3 порта A включаются на ввод. В регистр TRISB записываем 0x00. При этом все линии порта B будут работать на вывод.

Команда BCF STATUS, RP0 выполняет возврат в банк 0.

Листинг 4.27

```
     ; подпрограмма инициализации портов
INIT_PORTS
             MOVLW    0xFF   ; установка портов в 1
             MOVWF    PORTA
             MOVWF    PORTB
             BSF      STATUS,RP0   ; переход в банк 1
             MOVLW    0x0F    ; RA0, RA1, RA2, RA3 –
                              ; ввод, ост. вывод
             MOVWF    TRISA
             MOVLW    0x00
             MOVWF    TRISB    ; RB0-RB7 - вывод
             BCF      STATUS,RP0   ; возврат в банк 0
```

```
            RETURN      ; возврат из подпрограммы
Label1
           END
```

После выполнения инициализации портов командой GOTO Label1 (листинг 4.26) переходим к выполнению учебной программы 1 «Включение светодиода VD2 ключом SW0».

Лабораторное задание

1. Для отладки программы и моделирования работы микроконтроллера надо создать проект в MPLAB IDE. Сначала в редакторе *Notepad++* аккуратно наберите полный текст листингов 4.25, 4.26 и 4.27. Установите в Notepad++ «Кодировать в ANSI». Создайте папку MP-LAB3 и сохраните файл под названием *LAB3*, указав расширение *.asm.

Загрузите программу MPLAB IDE, для микроконтроллера PIC16F84A создайте проект MP-LAB3, добавьте в проект файл *LAB3.asm*. При этом не забудьте в правом окне у названия файла получить букву «С», чтобы файл был скопирован в проект.

После готовности проекта проведите сборку проекта *Project - Build All* и добейтесь успешного результата. Чтобы русский текст отображался правильно, установите в окне *Edit-Editor Properties-Text* кодировку 1251 (ANSI - Cyrillic).

На рабочем поле MPLAB IDE после метки Label1 выполните набор текста учебной программы (листинг 4.28):

Листинг 4.28

```
;* учебная программа  - включение светодиода VD2
;ключом SW-0

Label1
LOOP1       CLRWDT          ;сброс сторожевого таймера
            CALL     GET_RA ; вызов подпрограммы
                            ; GET_RA
            CALL     SB1_VD2  ; вызов подпрограммы
                              ; SB1_VD2
            GOTO    LOOP1  ; переход к метке LOOP1
                           ; для повторения процесса
GET_RA     ; подпрограмма чтения состояния порта A
            MOVF    PORTA,W  ; чтение состояния
```

```
                        ; порта A в W
            MOVWF     TEMPA  ; пересылка W в TEMPA
            RETURN       ; возврат из подпрограммы

SB1_VD2    ; подпрограмма вывода на светодиод VD2
           ; состояния кнопки SW0 (разряда 0)
            ; регистра TEMPA)
            BTFSS     TEMPA,0  ; пропустить команду,
                               ; если TEMPA,0=1
                               ; кнопка нажата
            GOTO       P0      ; перейти на P0
            BSF        VD2  ; зажечь светодиод VD2
P0
            BTFSC      TEMPA,0 ; пропустить команду,
                           ; если TEMPA,0=0
                           ; (кнопка не нажата)
            GOTO       P1   ; перейти на P1
            BCF        VD2   ; погасить светодиод
P1
            RETURN
```

Программа выполняет включение светодиода VD2 ключом SW0. Рассмотрим работу программы.

Основная программа содержит замкнутый цикл LOOP1-GOTO LOOP1 для периодического повторения опроса состояния ключа SW0 и управления индикатором. Команда CLRWDT очищает сторожевой таймер, чтобы программа не прерывалась по переполнению таймера. Две следующие команды вызывают подпрограммы CALL GET_RA и CALL SB1_VD2. Подпрограмма GET_RA выполняет считывание текущего состояния порта A и запись в регистр TEMPA.

Информация о состоянии ключа SW0 хранится в разряде 0 регистра TEMPA. Подпрограмма SB1_VD2 анализирует состояние разряда 0 этого регистра. Для этого используются команды BTFSS и BTFSC, пропускающие выполнение следующей за ними команды в зависимости от состояния TEMPA,0. При этом выполняются переходы на команду включения BSF VD2 или выключения BCF VD2 светодиода.

После набора текста листинга 4.28 выполните компиляцию программы и в случае успешного результата загрузите HEX-файл и ASM-файл из проекта в микроконтроллер модели (рис.4.14). В интерактивном режиме выполните *START* и *RUN*. Управляя ключом SW0, проверьте правильность работы учебной программы 1. Сделайте скриншот модели с включенным светодиодом.

2. В проекте MP-LAB3 наберите текст учебной программы 2 (листинг 4.29). Эта программа должна включать светодиод VD2 при состоянии тумблеров SW3=SW1=1, SW2=SW0=0. В тексте программы (листинг 4.26) сделайте переход GOTO Label2. Выполните *Project - Build All.* Если результат успешный, внимательно проанализируйте текст программы и ее работу.

Листинг 4.29

```
;* учебная программа  - включение светодиода VD2
; при состоянии тумблеров SW3=1, SW2=0, SW1=1, SW0=0
Label2
LOOP2
            CLRWDT  ; сброс сторожевого таймера
            CALL     GET_RA; вызов подпрпограммы
                           ; GET_RA
            CALL     ZAG_1010  ; вызов подпрограммы
                               ; ZAG_1010
            GOTO     LOOP2  ; переход к LOOP для
                            ; повторения процесса

ZAG_1010          ; зажигает светодиод VD2 при
                  ; условии SW3=SW1=1 SW2=SW0=0
            BTFSS    TEMPA,1    ; пропустить команду,
                                ; если  TEMPA,1=1

            GOTO      P2
            BTFSS     TEMPA,3   ; пропустить команду,
                                ; если TEMPA,3=1
            GOTO      P2        ; TEMPA,3=1
            BTFSC     TEMPA,2   ; пропустить команду,
                                ; если TEMPA,2=0
            GOTO      P2         ;
            BTFSC     TEMPA,0   ; пропустить команду,
```

```
                                    ; если TEMPA,0=0

            GOTO     P2  ;
            BSF      VD2; зажечь светодиод VD2
            GOTO     P3
P2
            BCF      VD2  ; погасить светодиод
P3
            RETURN
```

Загрузите HEX-файл и ASM-файл в модель микроконтроллера и проверьте работу программы.

Отредактируйте программу так, чтобы светодиод включался при наборе ключами двоичного числа, равного номеру Вашей бригады. Выполните компиляцию, проверьте работу программы в модели. Сделайте скриншот модели с включенным светодиодом.

3. В проекте MP-LAB3 наберите текст учебной программы 3 «Индикатор» (листинг 4.30).

Листинг 4.30

```
; учебная программа  – индикатор

Label3
            CLRWDT     ; сброс сторожевого таймера
            CLRF     PORTA    ; очистка PORTA
            CLRF     PORTB    ; очистка PORTB
            BSF      STATUS,5  ; переход в банк 1
            CLRF     TRISB    ; установка PORTB
                              ; на вывод
            BCF      STATUS,5   ; переход в банк 0

LOOP3
            CALL     GET_RA  ; программа чтения PORTA
            MOVF     TEMPA,0 ; пересылка TEMPA в W
            MOVWF    PORTB   ; пересылка W в PORTB
            GOTO     LOOP3
```

Эта программа отражает на двух индикаторах четырехбитное число, набранное ключами SW0-SW3. В первой части программы выполняется очистка портов A и B, порт B устанавливается на вывод. Циклическая программа по метке

LOOP3 выполняет чтение состояния ключей на входах порта A и пересылает эти данные в порт B для отражения на индикаторах. HEX-дисплей показывает шестнадцатиричное число, а LCD – дисплей – десятичное. В тексте программы (листинг 4.26) сделайте переход GOTO Label3.

Выполните компиляцию программы, загрузите программу в модель и проверьте правильность отображения чисел, набранных ключами.

Сделайте скриншот модели с отображением номера Вашей бригады.

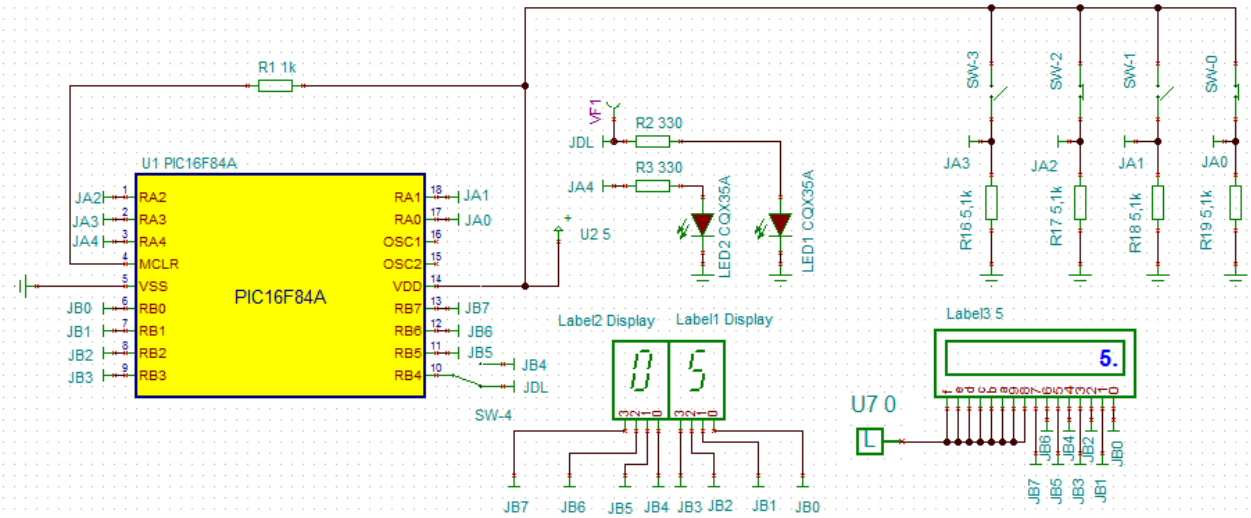


Рис.4.15. Отображение числа 5.

4. В проекте MP-LAB3 наберите текст учебной программы 4 «Мигание светодиода» (листинг 4.31). Светодиод должен мигать с определенной частотой при замыкании ключа SW0.

Листинг 4.31

```
   ; учебная программа - мигание светодиода VD1
Label4
LOOP4
            CLRWDT    ; сброс стороржевого таймера
            CALL      GET_RA  ; вызов подпрограммы
                              ; чтения PORTA
            CALL      SW_0    ; вызов подпрограммы SW_0
            GOTO      LOOP4   ; переход к метке LOOP4
                              ; для повторения процесса
```

```
SW_0        ; подпрограмма мигания светодиода VD1
            ; при нажатии кнопки SW-0
             BTFSS    TEMPA,0 ; пропустить команду,
                              ; если TEMPA,0=1
                              ; (кнопка нажата)
             GOTO     B0      ; перейти на B0
             BSF      PORTB,4  ; подача высокого
                               ; уровня на RB4
             MOVLW    0x3E     ; пересылка константы
                               ; .62 в W
             CALL     DELAY_A  ; вызов подпрограммы
                               ; задержки DELAY_A
             BCF      PORTB,4  ; подача низкого
                               ; уровня на RB4
             MOVLW    0x3E     ; пересылка константы
                               ; .62 в W
             CALL     DELAY_A  ; вызов подпрограммы
                               ; DELAY_A
B0
             RETURN

DELAY_A     ; подпрограмма формирования задержки
             MOVWF    COUNT1   ; загрузка W в регистр
                               ; COUNT1
LOOP5
             DECFSZ   COUNT1,F  ; декремент COUNT1
             GOTO     LOOP5  ; повторение цикла .62
                             ; раза
             RETURN   ; возврат из подпрограммы
             END
```

Программа выдает сигналы определенной длительности и частоты. Для этого используются подпрограммы формирования временной задержки.

В цикле LOOP4 выполняется подпрограмма чтения состояния порта A (GET_RA) и подпрограмма мигания светодиода SW_0. Если ключ SW0 замкнут (TEMPA,0=1), на вывод RB4 командой BSF PORTB,4 подается высокий уровень, в регистр W загружается константа (например, 0x3E), определяющая длительность включения светодиода, и

происходит вызов подпрограммы задержки DELAY_A. Если ключ SW0 разомкнут, командой BCF PORTB,4 подается низкий уровень на вывод RB4, и светодиод выключается на время задержки, определяемое константой 0x3E.

Подпрограмма формирования задержки DELAY_A работает следующим образом. Из регистра W константа 0x3E загружается в регистр COUNT1. В каждом цикле подпрограммы LOOP5 команда DECFSZ COUNT1,F (где ранее задано: F EQU 1) уменьшает на единицу содержимое регистра COUNT1 и сохраняет результат в этом регистре. Когда результат станет равным нулю, происходит возврат из подпрограммы LOOP5.

Для исполнения каждого внутреннего цикла требуется три машинных цикла микроконтроллера (1 цикл на исполнение команды DECSFZ при ненулевом результате и два цикла на исполнение команды GOTO). Выход из подпрограммы DELAY_A потребует 4-х циклов (2 цикла на исполнение команды DECSFZ при нулевом результате и 2 цикла на RETURN). К этому надо добавить еще 4 цикла, необходимых для загрузки константы в рабочий регистр, вызова подпрограммы и загрузки регистра COUNT1. Тогда общее время исполнения подпрограммы DELAY_A составит:

$T_D = 4 + 3 \cdot (L - 1) + 4 = 5 + 3 \cdot L$ циклов, где L – константа, переданная через рабочий регистр в подпрограмму DELAY_A.

При тактовой частоте микроконтроллера 4 МГц время цикла из 4-х тактов составляет 1 мкс. При загрузке L=0x3E формируемый интервал времени составит 5+3x62 = 191 мкс.

Для более точного расчета к времени задержки подпрограммы DELAY_A следует прибавить время исполнения команд внешнего цикла LOOP4.

В исполняемой программе (листинг 4.26) поставьте команду GOTO Label4. Выполните компиляцию программы и загрузите в модель макета. Ключ SW4 переключите в положение JDL. В интерактивном режиме выполните START и RUN. Наблюдайте мигание светодиода при замкнутом ключе SW0.

Остановите моделирование и выполните *Analysis-Transient* на интервале 0 – 1 мс. Определите длительность включения светодиода с помощью курсоров и сравните с расчетом (рис.4.16). Объясните разницу времени включения и выключения светодиода.

Отредактируйте программу так, чтобы период переключений светодиода был близок к 400 мкс, а отношение времени включения к времени выключения было близко к номеру Вашей бригады.

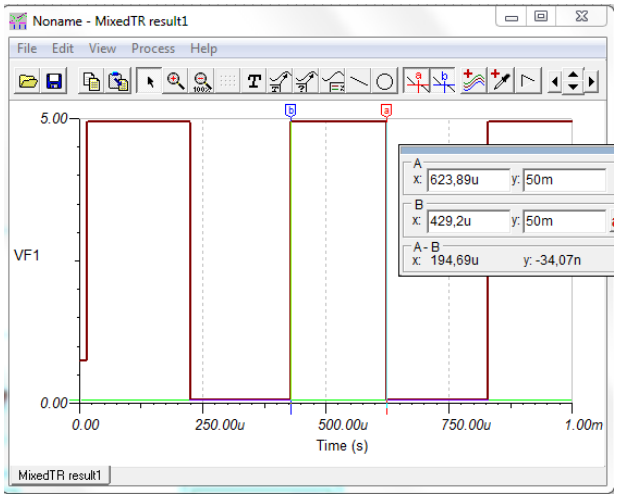


Рис.4.16. Измерение длительности включения светодиода

Домашнее задание

Оформите отчет по работе. Отчет должен содержать полный текст программы, скриншоты моделирования в TINA, обсуждение результатов и выводы.

Контрольные вопросы

1. Для чего применяют директиву EQU?
2. В каких файлах содержится описание микроконторрллера и как подключить эти файл к программе ?
3. Для чего применяют директиву #DEFINE ?
4. Для чего применяют директиву ORG ?
5. Как создать в программе замкнутый цикл ?
6. Как проводится анализ состояния кнопок и управление включением светодиода ?

7. Как работает программа «Индикатор», отображающая числа на дисплеях ?

8. Как работает подпрограмма формирования задержки ?

9. Как рассчитать длительность выполнения подпрограммы задержки ?

10. Как изменится длительность задержки, если тактовая частота микроконтроллера будет 1МГц ?

4.3. Лабораторная работа №4

Программирование арифметических и логических операций

Цель работы: изучение ввода чисел в микроконтроллер, выполнения арифметических и логических операций и отображения результатов.

Лабораторное задание

1. Собрать в программе TINA модель сумматора (рис.4.17) и сохранить схему в папке TI-LAB4 под именем LAB4. Сумматор содержит микроконтроллер, две HEX-клавиатуры и LCD – дисплей. Клавиатуры подключены к линиям порта B, работающим на ввод. Дисплей подключен к линиям RA0-RA4 порта A, работающим на вывод. Вывод RA4 с открытым коллектором подключен через R_2 к питанию.

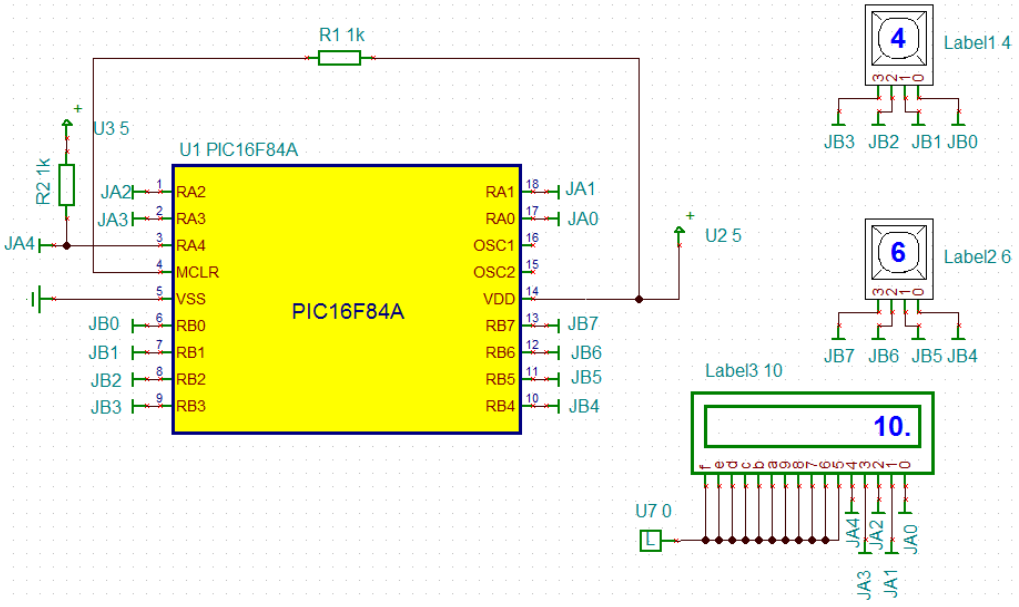


Рис.4.17. Модель сумматора

2. Составить программу суммирования двух чисел, установленных на HEX-клавиатурах. Для этого создать в MPLAB IDE в папке MP-LAB4 проект LAB4, составить программу, выполнить компиляцию и моделирование. В программе можно использовать фрагменты листингов 4.25, 4.26, 4.27 из лабораторной работы №3, редактируя их для новой задачи. В итоге шаблон программы может соответствовать листингу 4.32.

Листинг 4.32

```
LIST  P=PIC16F84A,  R=HEX; директива, определяющая
  ; тип процессора и систему счисления по умолчанию
#include <P16F84.inc>
; описание используемых переманных и адресов
; ячеек для хранения переменных пользователя

INTCON      EQU      0x0B
TMRO        EQU      0x01
INTF        EQU       1
TOIF        EQU       5
PCL         EQU      0x02
STATUS      EQU      0x03
RP0         EQU       5
IRP         EQU       7
PORTA       EQU      0x05
PORTB       EQU      0x06
TRISA       EQU      0x85
TRISB       EQU      0x86
W           EQU      0
F           EQU      1
TEMPA       EQU      0x0C
TEMPB       EQU      0x0D
COUNT1      EQU      0x0E
COUNT2      EQU      0x0F
COUNT3      EQU      0x10
OPTION_REG  EQU      0x81

;*        Определение меток текста

#DEFINE     Z      STATUS,2 ; бит нулевого результата

;исполняемая программа
```

```
            ORG     0x000
            GOTO    BEGIN
            ORG     0x005

BEGIN
            CALL    INIT_PORTS  ; вызов
                ; подпрограммы инициализации портов
            GOTO    Label1

               ; подпрограмма инициализации портов
INIT_PORTS
            MOVLW    0xFF  ; установка портов в 1
            MOVWF    PORTA
            MOVWF    PORTB
            bsf      STATUS,5   ; переход в банк 1
            MOVLW    0x00  ; RA0, RA1, RA2,
                           ; RA3, RA4 - вывод
            MOVWF    TRISA
            MOVLW    0xFF
            MOVWF    TRISB  ; RB0-RB7 - ввод
            BCF      STATUS,RP0 ; возврат в банк 0
            RETURN   ; возврат из подпрограммы
```

При моделировании сложить десятичные числа в соответствии с таблицей 4.1

Таблица 4.1

Номер бригады	1	2	3	4	5	6	7	8	9	10
Первое число	7	13	6	12	9	7	11	14	8	4
Второе число	12	7	14	5	13	10	9	3	12	15

Возможный вариант основной программы суммирования двух чисел показан на листинге 4.33.

Листинг 4.33

```
; Задание 1. Суммирование двух чисел
Label1
```

```
LOOP1       CLRWDT   ;сброс сторожевого таймера
            CALL     GET_RB  ; вызов подпрограммы
                             ; GET_RB
            MOVF     TEMPB,0 ; переслать TEMPB в W
            ANDLW    0x0F    ; побитно сложить W с
                             ; 0x0F=00001111
            MOVWF    COUNT1  ; первое число переслать
                             ; в COUNT1
            MOVF     TEMPB,0 ; переслать TEMPB в W
            ANDLW    0xF0    ;побитно сложить W с
                             ; 0xF0=11110000
            MOVWF    COUNT2  ; второй полубайт
                             ;второго числа в COUNT2
            SWAPF    COUNT2,0 ; поменять местами
            ; полубайты второго числа, результат в W
            ADDWF    COUNT1,0  ; сложить W с первым
                               ;числом, результат в W
            MOVWF    PORTA  ; переслать результат
                            ; в PORTA

            GOTO     LOOP1   ; переход к метке LOOP
                            ; для повторения процесса
GET_RB       ; подпрограмма чтения состояния порта B
            MOVF     PORTB,W  ; чтение состояния
                              ; порта B в W
            MOVWF    TEMPB  ; пересылка W в TEMPB
            RETURN          ; возврат из подпрограммы
            END
```

Записать программы из листингов 4.32 и 4.33 в MPLAB IDE, выполнить компиляцию и отладку, проверить на модели сумматора в TINA. Сделать скриншот результата моделирования.

3. Составить программу вычитания двух чисел. На дисплее должен отображаться модуль разности двух чисел.

Пример программы показан на листинге 4.34.

Листинг 4.34

```
;Задание 2.  Вычитание числа B из числа A

Label2
LOOP2       CLRWDT  ;сброс сторожевого таймера
```

```
CALL    GET_RB  ; вызов подпрограммы
                 ; GET_RB
MOVF    TEMPB,0 ; переслать TEMPB в W
ANDLW   0x0F    ; побитно сложить W с
                 ; 0x0F=00001111
MOVWF   COUNT1  ; первое число переслать
                 ; в COUNT1
MOVF    TEMPB,0 ; переслать TEMPB в W
ANDLW   0xF0    ;побитно сложить W с
                 ; 0xF0=11110000
MOVWF   COUNT2  ; второй полубайт второго
                 ; числа в COUNT2
SWAPF   COUNT2,1; поменять местами
                ; полубайты второго
               ; числа, результат в COUNT2
MOVF    COUNT2,0  ; переслать второе
                  ; число в W
SUBWF   COUNT1,0  ; вычесть из W первое
                  ; число
BTFSS   STATUS,0  ; проверка флага
                ; отрицательного результата
SUBLW    0x20  ; вычесть W из 1F
MOVWF    PORTA  ; переслать результат
                ; в PORTA
GOTO     LOOP2  ; переход к метке LOOP2
                ; для повторения процесса
```

На дисплее отобразить номер бригады. Записать программу из листинга 4.34 в отчет. Сделать скриншот моделирования сумматора в TINA.

4. Составить программу выполнения операции побитного «Исключающего ИЛИ» двух чисел (проверка на одинаковость). Пример программы показан на листинге 4.35.

Листинг 4.35

```
; Задание 3. Исключающее "ИЛИ"
Label3
LOOP3        CLRWDT   ;сброс сторожевого таймера
             CALL     GET_RB ; вызов подпрограммы
                             ; GET_RB
             MOVF     TEMPB,0 ; переслать TEMPB в W
             ANDLW    0x0F  ; побитно сложить W с
```

```
                    ; 0x0F=00001111
MOVWF     COUNT1 ; первое число переслать
                    ; в COUNT1
MOVF      TEMPB,0 ; переслать TEMPB в W
ANDLW     0xF0 ;побитно сложить W
                   ; с 0xF0=11110000
MOVWF     COUNT2 ; второй полубайт
                ; второго числа в COUNT2
SWAPF     COUNT2,1 ; поменять местами
                   ; полубайты второго числа,
                    ; результат в COUNT2
MOVF      COUNT2,0 ; переслать второе
                    ; число в W
XORWF     COUNT1,0 ; побитное исключающее
                    ; ИЛИ для W и COUNT1
MOVWF     PORTA    ; переслать результат
                    ; в PORTA

GOTO      LOOP3   ; переход к метке LOOP3
                  ; для повторения процесса
```

Записать программу из листинга 4.35 в отчет, сделать скриншот модели сумматора в TINA. Используя двоичную запись чисел, объяснить показания на клавиатуре и дисплее для двух неодинаковых чисел.

5. Составить программу выполнения циклического сдвига вправо и суммирования двух чисел. Одно из чисел соответствует номеру бригады. Используя двоичную запись чисел, объяснить показания на клавиатуре и дисплее.

Возможный вариант программы показан на листинге 4.36.

Листинг 4.36

```
; Задание 4. Циклический сдвиг вправо
Label4
LOOP4         CLRWDT   ;сброс сторожевого таймера
              CALL     GET_RB ; вызов подпрограммы
                        ; GET_RB
              MOVF     TEMPB,0  ; переслать TEMPB в W
              ANDLW    0x0F  ; побитно сложить W
                        ; с 0x0F=00001111
              MOVWF    COUNT1 ; первое число переслать
```

```
                ; в COUNT1
MOVF     TEMPB,0  ; переслать TEMPB в W
ANDLW    0xF0  ;побитно сложить W
               ; с 0xF0=11110000
MOVWF    COUNT2 ; второй полубайт
           ; второго числа в COUNT2
SWAPF    COUNT2,1 ; поменять местами
              ; полубайты второго числа,
                ; результат в COUNT2
RRF      COUNT1,1 ; сдвиг вправо числа
                  ; COUNT1
RLF      COUNT2,1 ; сдвиг влево числа
                  ; COUNT2
MOVF     COUNT2,0 ; переслать второе
                  ; число в W
ADDWF    COUNT1,0 ; сложить W и COUNT1
MOVWF    PORTA    ; переслать результат
                  ; в PORTA
GOTO     LOOP4 ; переход к метке LOOP4
                ; для повторения процесса
END
```

Домашнее задание

Оформите отчет по работе. Отчет должен содержать полную программу скриншоты моделирования в TINA, обсуждение результатов и выводы.

Контрольные вопросы

1. Назовите команды сложения из системы команд PIC16F84A и объясните их работу.

2. Назовите команды вычитания и объясните их работу.

3. Какие биты регистра STATUS используют для анализа результата выполнения команд сложения и вычитания ?

4. Назовите команды «Исключающее ИЛИ» и объясните их работу.

5. Для чего в изученных программах применяются операции логического И ?

6. Для чего в изученных программах применяется операция SWAPF ?

7. Назовите команды циклического сдвига и объясните их работу ?

8. Какое число будет на индикаторе после выполнения программы (листинг 4.35), если ввести первое число 2, а второе 7?

9. Какое число будет на индикаторе после выполнения программы (листинг 4.36), если ввести первое число 3, а второе 5?

4.4. Лабораторная работа №5

Применение циклов задержки

Цель работы: изучение способов создания задержки определенной величины с помощью циклов. Использования LCD – дисплея для счета импульсов мигания индикатора.

Лабораторное задание

1. Собрать в программе TINA модель мигалки и счетчика (рис.4.18) и сохранить файл в папке TI-LAB5.

В модели ключ SW2 выполняет сброс счетчика и дисплея в нулевое состояние. Ключ SW1 запускает и останавливает счет. Логический индикатор L1 мигает с заданной частотой. Опыт показывает, что в программе TINA дисплеи работают надежнее, если все компоненты схемы являются логическими (логический индикатор, цифровые источники высокого и низкого уровня и т.д.).

2. Составить программу для микроконтроллера. При составлении программы можно воспользоваться фрагментами подпрограммы «Мигание светодиода» (листинг 4.31).

Создайте проект LAB5 в папке MP-LAB5 и добавьте в проект файл LAB3.asm из проекта LAB3.

Выполните редактирование текста: удалите ненужные фрагменты, измените инициализацию портов и т.д.

Возможный вариант основной программы показан на листинге 4.37.

Основная программа содержит замкнутый цикл LOOP4 – GOTO LOOP4 и несколько подпрограмм.

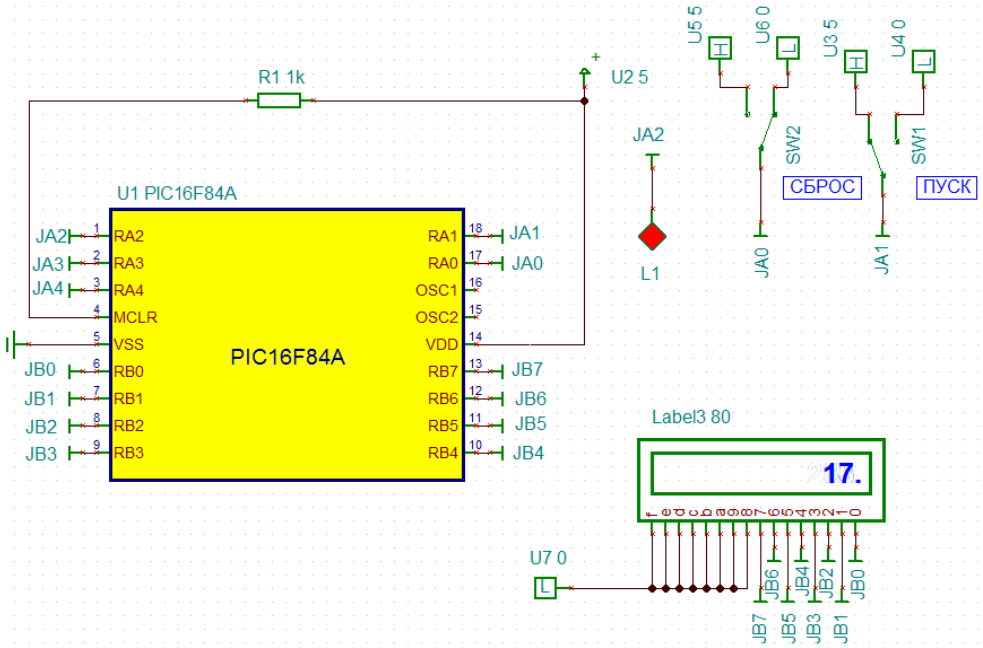


Рис.4.18. Схема модели мигалки и счетчика

Листинг 4.37

```
; учебная программа  -  счетчик с мигалкой
Label4
              MOVLW     0  ; записать 0 в регистр W
              MOVWF     COUNT2 ; записать 0 в регистр
                               ; счетчика
              MOVWF     PORTB
LOOP4         CLRWDT  ; сброс стороржевого таймера
              CALL      GET_RA  ; вызов подпрограммы
                                ; чтения PORTA
              NOP  ; пустая команда для отладки
              CALL      SW_0  ; вызов п.п. мигания
                              ; светодиода
              GOTO      LOOP4  ; переход к метке LOOP4
                        ; для повторения процесса
```

Подпрограмма GET_RA выполняет чтение состояния порта A (листинг 4.38).

Листинг 4.38

```
GET_RA    ; подпрограмма чтения состояния порта A
              MOVF      PORTA,W  ; чтение состояния
                                 ; порта A в W
              MOVWF     TEMPA  ; пересылка W в TEMPA
              RETURN     ; возврат из подпрограммы
```

Подпрограмма SW_0 анализирует состояние ключа SW, кнопки PB1 , управляет включением и выключением индикатора и счетчиком (листинг 4.39).

Листинг 4.39

```
SW_0        ; подпрограмма мигания светодиода VD1
            ; при нажатии кнопки SW-0
            BTFSS      TEMPA,1 ; пропустить команду,
                               ; если TEMPA,1=1
            GOTO       B0   ; перейти на B0
            BSF        PORTA,2  ; подача высокого
                               ; уровня на RA2
            CALL       DELAY_A
            BCF        PORTA,2   ; подача низкого
                                 ; уровня на RA2
            CALL       DELAY_A

            MOVLW      0x00 ; записать 0 в регистр W
            MOVF       COUNT2,W
            BTFSS      TEMPA,0  ; пропустить
                           ; команду, если TEMPA,0=1
            GOTO       B1
            MOVLW      0   ; записать 0 в регистр W
            MOVWF      COUNT2 ; записать 0 в регистр
                             ;  COUNT2
B1          CALL       SHET  ; вызов п/п счета

B0          RETURN
```

Подпрограмма DELAY_A (листинг 4.40) создает временную задержку и определяет частоту мигания индикатора. Для этого командой MOVLW 0x03 в регистр W записываем число, определяющие количество повторений цикла LOOP7 подпрограммы DELAY_A. Командой DECFSZ COUNT1,F в каждом цикле это число уменьшается на единицу, а результат сохраняется в регистре COUNT1. Когда результат станет равным нулю, произойдет выход из подпрограммы.

Листинг 4.40

```
DELAY_A     ; подпрограмма формирования задержки
              MOVLW     0x03
```

```
            MOVWF     COUNT1  ; загрузка W в регистр
                              ; COUNT1
LOOP7
            DECFSZ    COUNT1,F  ; декремент COUNT1
            GOTO      LOOP7  ; повторение цикла
                             ; заданное число раз
            RETURN     ; возврат из подпрограммы
```

Перед вызовом подпрограммы SHET команда BTFSS TEMPA,0 проверяет состояние кнопки PB1 и выполняет переход на обнуление счетчика, если кнопка замкнута. Подпрограмма SHET суммирует в регистре COUNT2 количество миганий индикатора и выводит результат на дисплей (листинг 4.41).

Листинг 4.41

```
SHET     ; подпрограмма счетчика
             BSF     STATUS,RP0    ; переход в банк 1
             MOVLW   0x00
             MOVWF   TRISB  ; RB0-RB7 - вывод
             BCF     TRISB,7
             BCF     STATUS,RP0   ; возврат в банк 0
             MOVF    COUNT2,W
             MOVWF   PORTB
             MOVLW   0x01
             ADDWF   COUNT2,F  ;x = x + 1

             MOVF     COUNT2,W
             RETURN
```

3. Отладить программу, загрузить в модель мигалки и счетчика и проверить функционирование модели.

4. Вычислить задержку, которую дает обращение к подпрограмме DELAY_A с загруженным числом 0x03. Сначала надо из описания микроконтроллера узнать, сколько циклов требуется для выполнения каждой команды в подпрограмме задержки:

```
CALL           DELAY_A     - 2 цикла;
MOVLW          0x03              - 1 цикл;
MOVWF          COUNT1         -  1 цикл;
```

```
DECFSF          COUNT1,F     -   1 цикл;
GOTO            LOOP7            -    2 цикла;
RETURN           -    2 цикла.
```

Общее число циклов микроконтроллера в подпрограмме DELAY_A составит:

2+1+1+3(1+2)+1=14 циклов.

Каждый цикл выполняется за 4 такта генератора. При частоте генератора 4 МГц время выполнения подпрограммы задержки составит 14 мкс.

Но для расчета частоты мигания индикатора надо подсчитать все команды, которые входят во внешний цикл LOOP4.

Делать это достаточно хлопотно. Но в программе MPLAB IDE легко определить время выполнения программы или отдельной ее части.

Для этого в основную программу вместо команды NOP вставим команду BSF TEMPA,1 , которая имитирует замкнутый ключ SW (листинг 4.42).

Листинг 4.42

```
LOOP4          CLRWDT    ; сброс стороржевого таймера
               CALL     GET_RA  ; вызов подпрограммы
                                ; чтения PORTA
               BSF     TEMPA,1  ; замыкание ключа SW
               CALL    SW_0
               GOTO    LOOP4  ; переход к метке LOOP4
                              ; для повторения процесса
```

Установим точки останова на команде CALL DELAY_A и следующей команде MOVLW 0x00 (рис.4.19).

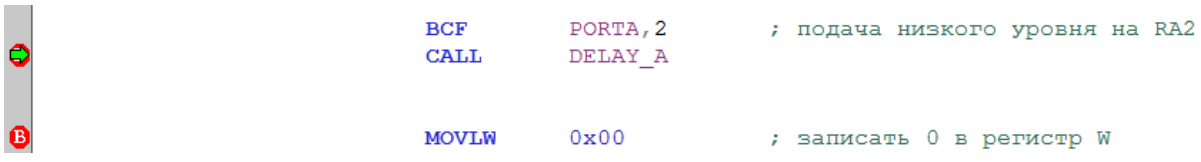


Рис.4.19. Точки останова в программе

Выполним *Deburger –Reset – Processor Reset*. Далее выполним *Deburger – Stop watch* и *Run*. В первой точке в окне *Windows – Stop watch* увидим число выполненных циклов

программы 49. Еще раз нажимаем Run и получаем число циклов после подпрограммы DELAY_A, равное 63 (рис.4.20). Следовательно, подпрограмма задержки выполняется за 14 циклов, что соответствует расчету.

Время выполнения подпрограммы на частоте процессора 20 МГц составляет 2,8 мкс. В модели частота процессора составляет 4 МГц. Следовательно, время выполнения подпрограммы задержки будет в 5 раз больше и составит 14 мкс.

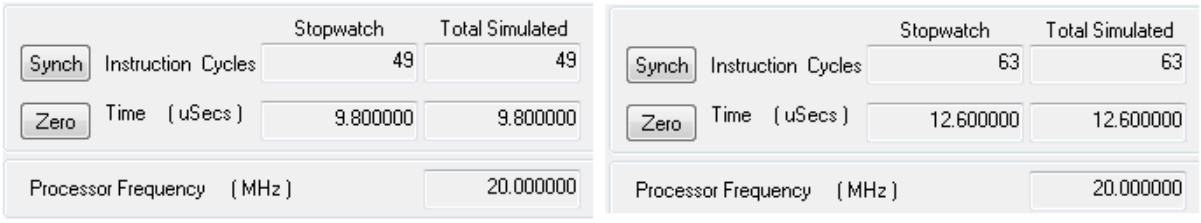


Рис.4.20. Окна времени выполнения подпрограммы задержки

Определим период мигания светодиода. Для этого установим одну точку останова на команде с меткой цикла LOOP4.

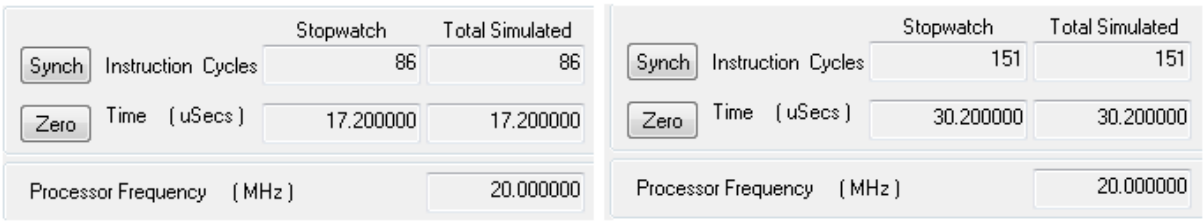


Рис.4.21. Окна времени выполнения всей программы при втором и третьем останове

Количество циклов между третьим и вторым остановом программы (рис. 4.21) составляет 65. Время исполнения для частоты процессора 20 МГц составляет 13 мкс, а на частоте 4МГц составит 65 мкс.

Выполним моделирование. К выходу RA2 порта А подключим *Voltage Pin VF1*. Выбираем *Analysis – Transient*. На временной диаграмме напряжения на светодиоде (рис.4.22) с помощью курсоров находим период включения индикатора 64,7 мкс.

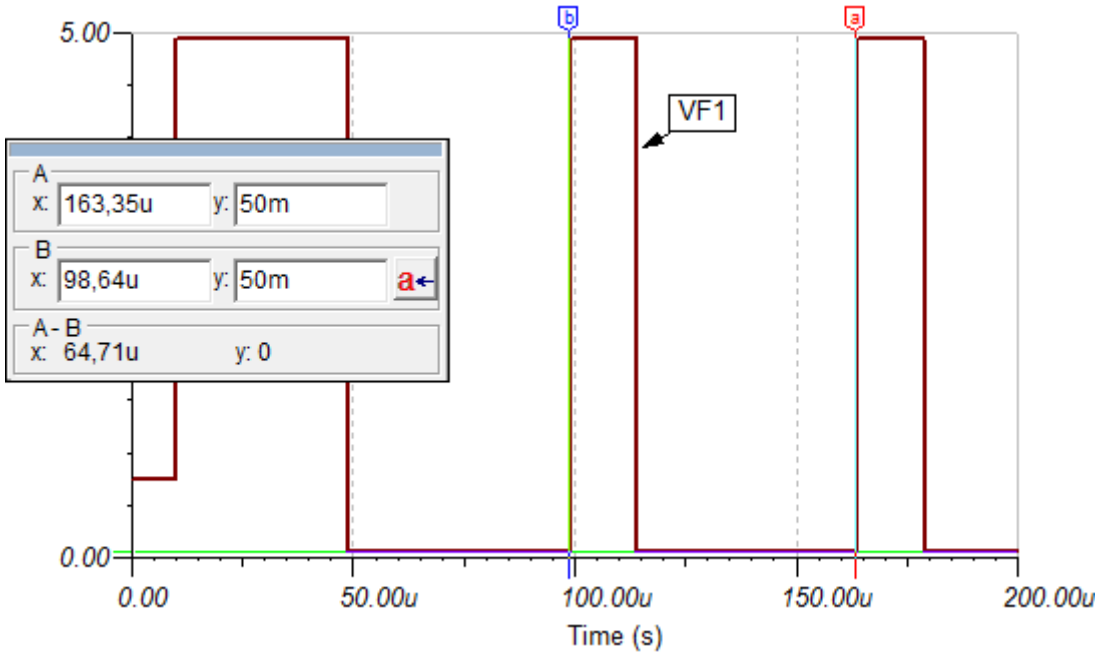


Рис.4.22. Диаграмма напряжения на индикаторе

Таким образом, мы получили совпадение результатов измерения периода мигания индикатора в программе MPLAB IDE и TINA.

2. Отредактируйте программу так, чтобы в подпрограмме DELAY_A в счетчик COUNT1 было записано число, Y=10+5N, где N – номер бригады. Выполните отладку программы, измерение периода мигания индикатора двумя методами. Получите диаграммы напряжения на индикаторе.

Домашнее задание

Оформите отчет по работе. Отчет должен содержать программу, скриншоты моделирования в TINA, обсуждение результатов и выводы.

Контрольные вопросы

1. Объясните работу подпрограммы формирования задержки.
2. Объясните работу подпрограммы счетчика.
3. Какое количество циклов требуется для выполнения различных команд микроконтроллера ?

4. Как подсчитать время выполнения фрагментов программы ?

5. Как измерить в MPLAB IDE время выполнения фрагментов программы ?

6. Как определить период мигания светодиода в модели программы TINA ?

4.5. Лабораторная работа №6

Применение прерываний программы

Цель работы: ознакомиться с тем, что такое прерывания, какие источники прерываний есть у микроконтроллера PIC16F84A, научиться программировать прерывания от изменения сигналов на линиях RB7-RB4 порта B и внешнее прерывание с вывода RB0/INT, составить программу для управления частотой мигания светодиода, используя прерывания.

Основные сведения о прерываниях

Напомним основные сведения о прерываниях из описания микроконтроллера PIC16F84A. Прерывания позволяют микроконтроллеру реагировать на некоторые события в момент их появления независимо от того, что выполняет в это время микроконтроллер. Это обеспечивает связь между микроконтроллером и окружающей средой. Каждое прерывание изменяет ход выполнения программы, прерывает ее, передает управление на вектор/адрес 0004 и после выполнения подпрограммы обслуживания прерываний продолжает основную программу с той же точки прерывания (рис.4.23).

Микроконтроллер PIC16F84A имеет 4 источника прерываний:

1. прерывание при завершении записи данных в EEPROM;
2. прерывание от переполнения счетчика/таймера TMR0;
3. прерывание от изменения сигналов на линиях RB4, RB5, RB6 и RB7 порта B;
4. внешнее прерывание с вывода RB0/INT.

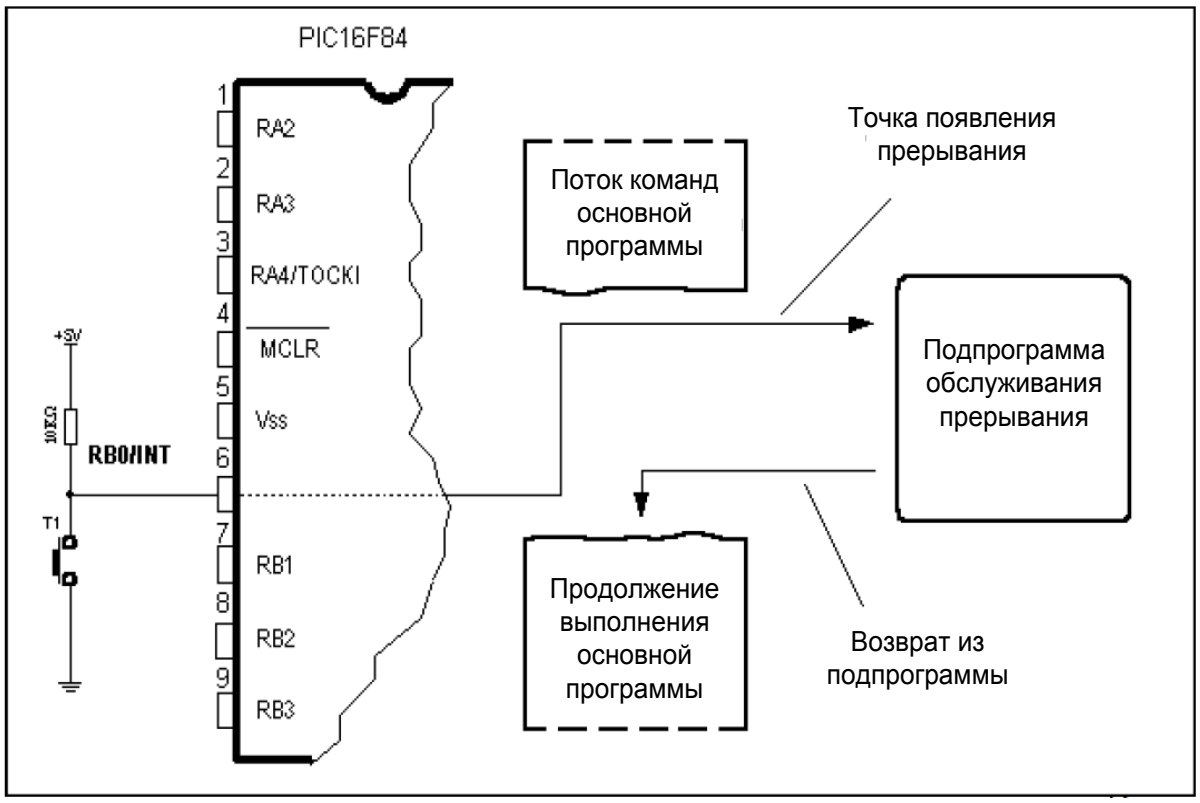


Рис.4.23. Схема выполнения прерывания

В данной лабораторной работе изучается прерывание от изменения сигналов на линиях порта B и внешнее прерывание. Для того, чтобы реагировать на сигналы прерываний, выводы RB<7:4> и RB0/INT должны быть включены на ввод. В этом случае текущее состояние на выводе сравнивается с предыдущим сохраненным состоянием и в случае их отличия генерируется прерывание. Время реакции на прерывания составляет приблизительно 5 циклов.

Разрешения прерываний и флаги прерываний устанавливаются в регистре прерываний INTCON.

Лабораторное задание

1. Собрать схему макета с прерываниями программы (рис.4.24). Сохранить файл с папке TI-LAB6.

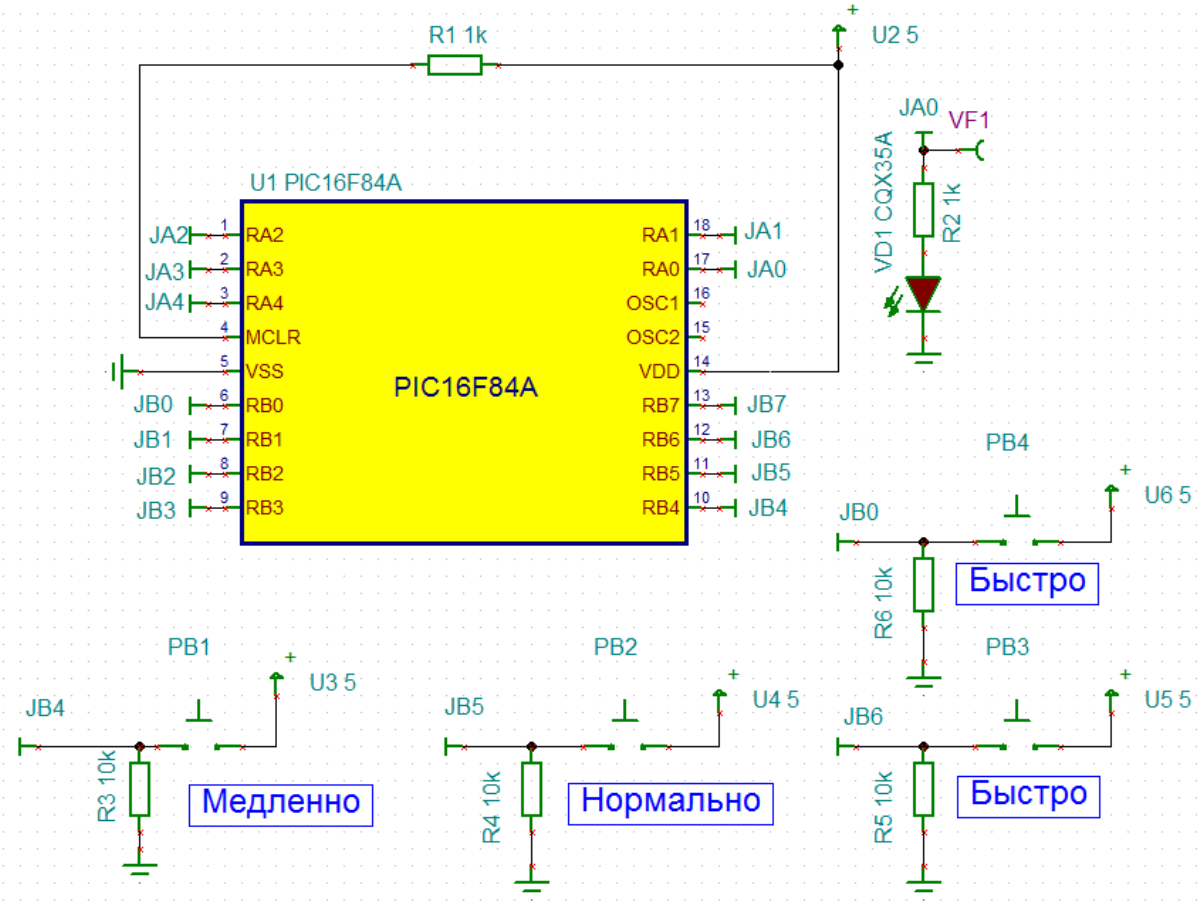


Рис.4.24. Схема макета с прерываниями программы

Светодиод - мигалка подключен к выходу RA0 порта A.

На входах RB<6:4> и RB0/INT порта B кнопками PB1-PB4 можно изменять текущее состояние и генерировать сигналы прерываний.

2. В MPLAB IDE создать проект LAB6 в папке MP-LAB6. Составить программу, изменяющую режим частоты мигания («медленно», «нормально», «быстро»).

Возможный вариант программы представлен ниже на листингах.

Шапка программы показана на листинге 4.43.

Листинг 4.43

```
LIST   P=PIC16F84A, R=HEX   ;   директива, определяющая
; тип процессора и систему счисления по умолчанию
#include <P16F84.inc>
; описание используемых переменных и адресов
; ячеек для хранения переменных пользователя
```

```
INTCON        EQU       0x0B
TMRO          EQU       0x01
INTF          EQU        1
TOIF          EQU        5
PCL           EQU       0x02
STATUS        EQU       0x03
EECON1        EQU       0x88
OPTION_REG    EQU       0x81
RP0           EQU        5
RBIF          EQU        0
RBIE          EQU        3
INTE          EQU        4
GIE           EQU        7
IRP           EQU        7
PORTA         EQU       0x05
PORTB         EQU       0x06
TRISA         EQU       0x85
TRISB         EQU       0x86
D             EQU       0
F             EQU       1
TEMPA         EQU       0x0C
TEMPB         QU        0x0D
COUNT1        EQU       0x0E
COUNT2        EQU       0x0F
STATUS_TEMP   EQU       0x11
W_TEMP        EQU       0x12
```

Для регистра прерываний INTCON задано положение битов разрешений прерываний GIE, RBIE, INTE и флагов прерываний RBIF, INTF, TOIF.

Основная программа (листинг 4.44) начинается с адреса 0x005. Прерывание передает управление на адрес 0x004 и выполняет переход на подпрограмму обслуживания прерываний INT_SERV.

Листинг 4.44

```
;  основная исполняемая  программа
            ORG     0x000
```

```
            GOTO    MAIN
            ORG     0x005

            ORG     0x004
            GOTO    INT_SERV
MAIN
            CALL    INIT_PORTREG ; вызов
               ; подпрограммы инициализации портов
            GOTO    Label4
```

В подпрограмме инициализации портов и регистров (листинг 4.45) порт А установлен на вывод, линии RB<6:4> и RB0/INT установлены на ввод. В регистре INTCON установлены разрешения на прерывания.

Листинг 4.45

```
INIT_PORTREG
            MOVLW    0xFF  ; установка портов в 1
            MOVWF    PORTA
            MOVWF    PORTB
            BSF      STATUS,RP0   ; переход в банк 1
            MOVLW    0x00
            MOVWF    TRISA; все RA на вывод

            MOVLW    0x71  ; 71H=01110001
            MOVWF    TRISB ; RB4-RB6 - ввод,
                           ; ост.  вывод
            MOVLW    0x00
            BCF     OPTION_REG,7  ; подтягивающие
                                ; резисторы включены
            BSF     OPTION_REG,6  ;прервыание по
                              ; переднему фронту INT
            BCF     STATUS,RP0   ; возврат в банк 0
            BCF     INTCON,RBIF ; очистка флага
                                ; прерываний
            BSF     INTCON,RBIE  ; разрещение
                 ; прерываний по изменению RB5-RB7
            BSF     INTCON,GIE   ; глобальное
                           ; разрешение прерываний
            BSF     INTCON,INTE  ; разрешено
                           ; прерывание по RB0/INT
```

```
            RETURN          ; возврат из подпрограммы
```

При переходе на метку Label4 в регистр COUNT1 записываем число 0x3E, которое будет соответствовать нормальной частоте мигания (листинг 4.46). В подпрограмме задержки DELAY_A выполняется декремент содержимого регистра COUNT2, в который предварительно загружаем число из регистра COUNT1.

Листинг 4.46

```
; учебная программа -  прерывания
Label4
            MOVLW   0x1E
            MOVWF   COUNT1

LOOP4       CLRWDT  ; сброс стороржевого таймера
            CALL    SW_0
            GOTO    LOOP4 ; переход к метке LOOP4
                          ; для повторени¤ процесса
SW_0    ; подпрограмма мигания светодиода VD1

            BSF     PORTA,0 ; подача высокого уровня
                            ; на RA2
            CALL    DELAY_A
            BCF     PORTA,0 ; подача низкого уровня
                            ; на RA2
            CALL    DELAY_A
            RETURN

DELAY_A   ; подпрограмма формирования задержки
            MOVF    COUNT1,0; загрузить COUNT1 в W
            MOVWF   COUNT2  ; загрузка W в регистр
                             ; COUNT2
LOOP7
            DECFSZ  COUNT2,F ; декремент COUNT2
            GOTO    LOOP7  ; повторение цикла
            RETURN   ; возврат из подпрограммы
```

Подпрограмма обслуживания прерываний INT_SERV (листинг 4.47) командами BTFSC PORTB,X анализирует на

каком входе сработала кнопка PB и выполняет загрузку разных чисел 0x03, 0x3E, 0x6E в регистр COUNT1. Первые три команды сохраняют текущее содержание регистров STATUS и W, так как в подпрограмме прерываний могут быть их изменения. Последние четыре команды восстанавливают содержание этих регистров перед возвратом в основную программу. Возврат в основную программу происходит по команде RETFIE.

Листинг 4.47

```
INT_SERV
             MOVWF   W_TEMP ; сохранить текущее
                             ; состояние W
             MOVF    STATUS,D  ; переслать STATUS в W
             MOVWF   STATUS_TEMP  ; сохранить STATUS

             BTFSC    PORTB,0
             GOTO     B0
             BTFSC    PORTB,4
             GOTO     B1
             BTFSC    PORTB,5
             GOTO     B2
             BTFSC    PORTB,6
             GOTO     B0
             GOTO     B3
B0           MOVLW    0x03
             MOVWF    COUNT1
             GOTO     B3
B1           MOVLW    0x6E
             MOVWF    COUNT1
             GOTO     B3
B2           MOVLW    0x3E
             MOVWF    COUNT1
B3           MOVF     STATUS_TEMP,D ; восстановить
                                    ; копию STATUS
             MOVWF    STATUS  ; восстановить Status
             SWAPF    W_TEMP,F
             SWAPF    W_TEMP,D  ; восстановить
                                ; содержание W
             RETFIE
```

3. Полный текст программы записать в Notepad++ и добавить в проект LAB6. Выполнить компиляцию и отладку программы.

4. Отлаженную программу загрузить в микроконтроллер модели макета с прерываниями и проверить функционирование.

Моделирование проводите в смешанном режиме. Для этого в меню программы TINA выберите *Analysis-Options* и установите дополнительно *Enable VHDL mixed mode.*

Домашнее задание

Оформите отчет по работе. Отчет должен содержать полный текст программы, скриншоты моделирования, обсуждение результатов и выводы.

Контрольные вопросы

1. Для чего применяют прерывания программы в микроконтроллерах ?

2. Объясните последовательность выполнения прерывания.

3. От каких источников могут выполняться прерывания в микроконтроллере PIC16F84A ?

4. Какой регистр используют для установки условий прерываний ?

5. Объясните назначение бита GIE регистра условий прерываний.

6. Объясните назначение битов EEIE и TOIE регистра условий прерываний.

7. Объясните назначение битов INTE и RBIE регистра условий прерываний.

8. Какие флаги используются в регистре условий прерываний и что они означают ?

9. В каком регистре можно выбрать фронт внешнего прерывания INT?

10. Объясните структуру и работу подпрограммы обслуживания прерываний.

11. В каком режиме программы TINA следует выполнять моделирование схем, содержащих микроконтроллеры и аналоговые элементы ?

4.6. Лабораторная работа №7

Применение таймеров TIMER0 и WDT

Цель работы: ознакомиться с таймером TIMER0 и его функциями; узнать, как сделать счетчик с использованием таймера TIMER0.

Основные сведения о таймерах TIMER0 и WDT

Структурная схема и функционирование таймера TIMER0 была рассмотрена в описании микроконтроллера PIC16F84A. Напомним кратко, что таймер TIMER0 содержит восьмиразрядный регистр с адресом 0x01 с возможностью записи и чтения, программно-управляемый предварительный делитель (предделитель), мультиплексор для выбора внутреннего или внешнего тактового входного сигнала, схему выбора фронта внешнего тактового сигнала, формирователь запроса прерывания по переполнению регистра TMR0 с FFh до 00h.

Режим работы таймера и предделителя устанавливается битами <0:5> регистра OPTION_REG. Разрешение прерывания по переполнению TMR0 задается битом T0IE регистра INTCON. Флаг прерывания по переполнению TMR0 выставляется битом TOIF регистра INTCON.

В подпрограмме обработки прерывания бит запроса прерывания T0IF должен быть сброшен. Если при каждом прерывании засылать в регистр TMR0 константу, то инкрементирование начнется с этого числа и цикл счета таймера уменьшится.

Лабораторное задание

1. Собрать схему счетчика на таймере (рис.4.25). Сохранить файл в папке TI-LAB7.

2. Составить и отладить в MPLAB-IDE программу так, чтобы счетчик увеличивал свое значение на единицу после каждых двух переключений SW на входе RA4/TOCK.

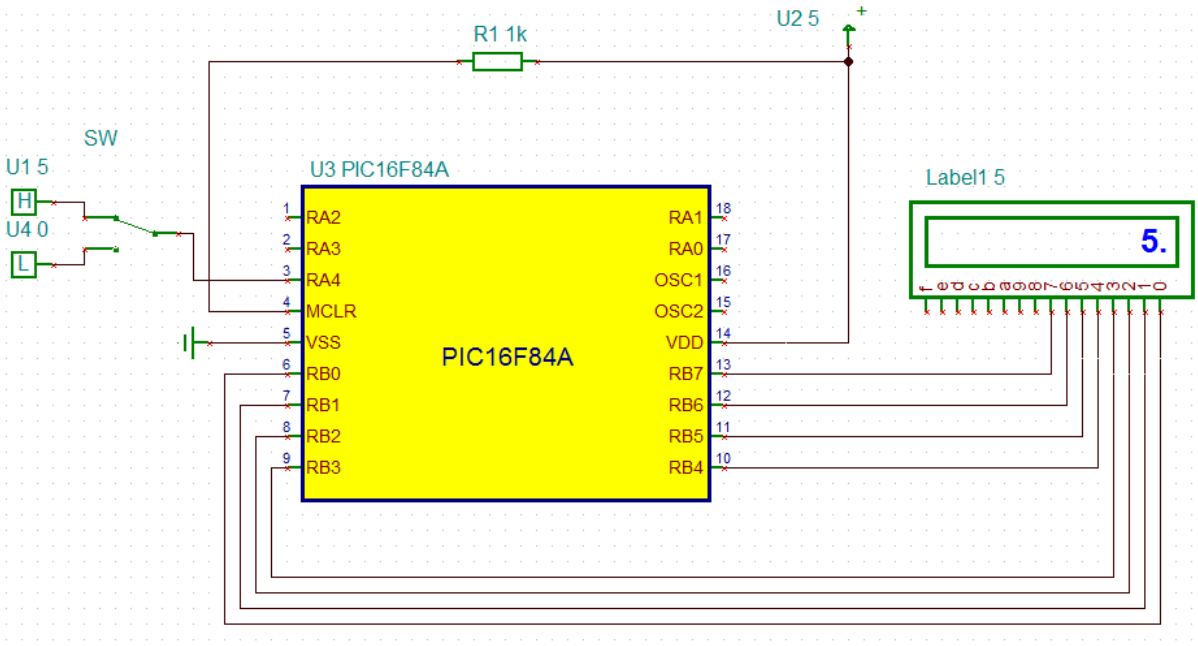


Рис.4.25. Модель счетчика на таймере

Возможный вариант текста программы показан на листингах 4.48-4.50.

```
LIST  P=PIC16F84A, R=HEX;   директива, определяющая
; тип процессора и систему счисления по умолчанию
; описание используемых переменных и адресов
; ячеек для хранения переменных пользователя
```

Листинг 4.48

```
#include <P16F84.inc>
INTCON       EQU      0x0B
TMR0         EQU      0x01
T0IE         EQU       5
T0IF         EQU       2
PCL          EQU      0x02
STATUS       EQU      0x03
OPTION_REG   EQU      0x81
RP0          EQU       5
GIE          EQU       7
PORTA        EQU      0x05
PORTB        EQU      0x06
TRISA        EQU      0x85
```

```
TRISB       EQU     0x86
```

В шапке программы (листинг 4.48) заданы адреса регистров INTCON, TMR0, PCL, STATUS, OPTION_REG, PORTA, PORTB, TRISA, TRISB, управляющих битов и переменных.

В основной программе (листинг 4.49) выполняется инициализация портов и регистров. Линия RA4/TOCK1 установлена на ввод, остальные линии порта А установлены на вывод. Линии порта В установлены на ввод.

Листинг 4.49

```
MAIN
            CALL    INIT_PORTREG ; вызов
                ; подпрограммы инициализации портов
            GOTO    Label4
               ; подпрограмма инициализации портов

INIT_PORTREG
            MOVLW   0xFF  ; установка портов в 1
            MOVWF   PORTA
            MOVWF   PORTB
            BSF     STATUS,RP0   ; переход в банк 1
            MOVLW   0xFF  ; RA4 ввод, ост. вывод
            MOVWF   TRISA
            MOVLW   0x00
            MOVWF   TRISB ; все RB - вывод
            MOVLW   0xE0  ; B'1110000'=E0 в W
            MOVWF   OPTION_REG   ; установка битов
                                 ; в OPTION_REG
            BCF     STATUS,RP0   ; возврат в банк 0
            MOVLW   0xA0  ; B'10100000'=A0 записать
                          ; в W
            MOVWF   INTCON       ; установка битов
                                 ; в INTCON

            RETURN   ; возврат из подпрограммы
```

В регистр OPTION_REG загружено число B’1110000’. При этом подтягивающие резисторы отключены, внешнее прерывание INT будет происходить по переднему фронту, таймер TIMER0 тактируется внешним тактом с вывода RA4/TOCKI, приращение

TMR0 происходит при перепаде от 0 к 1, предделитель включен перед TIMER0, коэффициент деления предделителя 1:2.

В регистр INTCON загружено число B’10100000’. При этом разрешены все немаскированные прерывания, запрещено прерывание по окончании записи в EEPROM, разрешено прерывание по переполнению TMR0, запрещены прерывания по входу RB0/INT, по изменению уровней сигналов на выводах RB4 – RB7. Сброшены все флаги прерываний.

Далее от метки Label4 после загрузки в таймер начального нулевого значения выполняется программа с циклом LOOP4 (листинг 4.50). Внешний тактовый сигнал с вывода RA4/TOCKI предделителем делится 1:2 и после каждых двух тактов увеличивает на единицу число в регистре таймера. Когда это число становится равным 255, происходит прерывание по переполнению таймера. В каждом цикле содержимое таймера отображается через порт B на дисплее.

Подпрограмма обслуживания прерываний INT_SERV формирует разрешения новых прерываний, сбрасывает флаг переполнения и возобновляет счет таймера с нуля.

Листинг 4.50

```
; учебная  программа -  счетчик на таймере
Label4
              MOVLW     0x00
              MOVWF     TMR0
LOOP4         CLRWDT   ; сброс стороржевого таймера
              MOVF      TMR0,0
              MOVWF     PORTB
              GOTO      LOOP4; переход к метке LOOP4
                            ; для повторения  процесса
INT_SERV

              BSF       INTCON,T0IE   ; разрешено
                          ; прерывание по переполнению
              BSF       INTCON,GIE
              BCF       INTCON,T0IF   ; сброс флага
                                      ; переполнения
              RETFIE
```

```
END
```

3. Смоделировать работу счетчика в среде TINA и проверить правильность функционирования.

4. Отредактировать программу так, чтобы таймер работал от внутреннего такта CLKOUT и коэффициенты деления предделителя были 1:2, 1:8, 1:64. Отладить программу в MPLAB IDE и выполнить моделирование в программе TINA.

5. Отредактировать программу так, чтобы таймер по п.4 начинал счет с числа 64.

Домашнее задание

Оформите отчет по работе. Отчет должен содержать полный текст программы в ассемблере, скриншоты моделирования в TINA, обсуждение результатов и выводы.

Контрольные вопросы

1. Какую структуру имеет таймер TIMER0 ?
2. В каких режимах может работать таймер TIMER0 и как устанавливают способ тактирования таймера ?
3. Как выбирают и устанавливают фронт приращения таймера при внешнем такте ?
4. Как выбирают способ включения предделителя ?
5. Как устанавливают коэффициент деления предделителя ?
6. Как устанавливают разрешение прерываний по переполнению регистра TMR0 ?
7. Как установливают начальное значение в таймере ?

4.7. Лабораторная работа №8

Применение памяти EEPROM и косвенной адресации

Цель работы: изучение косвенной адресации, записи данных в энергонезависимую память EEPROM и чтения данных из EEPROM.

Основные сведения о косвенной адресации

Прямая 9 – битная адресация использует младшие 7 бит как прямой адрес из кода операции, а два бита указателя страниц (RP1, PR0) из регистра статуса.

Признаком косвенной адресации является обращение к регистру INDF (адрес 00h). Команда, которая использует регистр INDF, фактически обращается к указателю, который хранится в регистре FRS (адрес 04h). Необходимый 9 – битный адрес формируется объединением 8 – битного числа FRS регистра и бита IRP из регистра статуса. Содержимое регистра FRS можно инкременировать, чтобы выполнять запись массива данных.

Основные сведения о применении энергонезависимой памяти EEPROM

В микроконтроллере PIC16F84A оперативное запоминающее устройство на регистрах называют ОЗУ, память программ называют Flash/ПЗУ. Электрически перепрограммируемую и энергонезависимую память данных EEPROM называют РПЗУ (репрограммируемое постоянное запоминающее устройство регистрового типа). В микроконтроллере PIC16F84A ее объем 64 х 8 бит, число циклов стирания/записи 1000000, срок хранения данных 40 лет.

Напомним, что доступ к памяти EEPROM осуществляется через два регистра: EEDATA (08h), который содержит в себе восьмибитовые данные для чтения/записи, и EEADR (09h), который содержит в себе адрес ячейки, к которой идет обращение. Дополнительно имеются два управляющих регистра: EECON1 (88h) и EECON2 (89h).

Процедура записи в EEPROM

Для записи в EEPROM надо выполнить следующие операции:

1. Установить адрес, в который надо сделать запись в EEADR.

2. Установить данные, которые требуется записать в регистр EEDATA.

3. Запретить все прерывания, установив GIE=0.

4. Установить в единицу бит WREN в регистре EECON1.

5. Выполнить последовательность команд:

```
movlw   55h
movwf   EECON2
movlw   AAh
movwf   EECON2
bsf     EECON1,WR.
```

6. Установить бит WR в EECON1.

7. Ждать, пока операция записи не закончится.

8. Разрешить все прерывания.

Время записи составляет около 10 мс. В конце записи бит WR автоматически обнуляется, а флаг завершения записи EEIF (он же запрос на прерывание) устанавливается.

Для предотвращения случайных записей в память данных служит бит WREN в регистре EECON1. Его рекомендуется держать выключенным, когда запись в память не требуется.

Процедура чтения из EEPROM

Для чтения данных из EEPROM надо выполнить следующие операции:

1. Установить адрес, с которого требуется считать данные, в EEADR.

2. Установить бит RD в EECON1.

3. Переслать читаемые данные из EEDATA в регистр W.

Эти операции будут повторяться при каждом случае записи в EEPROM или чтения. Изменяться будут только адрес и данные.

Лабораторное задание

1. Собрать схему модели EEPROM (рис.4.26). Сохранить файл LAB8 в папке TI-LAB8. На клавиатуре надо набирать последовательность заданных для бригады трех чисел и вводить эти числа в память ОЗУ по положительному фронту сигнала с ключа SW1 на линии RA4 порта A. Окончание записи в ОЗУ должно регистрироваться зажиганием индикатора L1. После окончания записи в ОЗУ числа требуется записать в EEPROM, последовательно считывать и выводить на дисплей с задержкой

для удобства чтения. Ключ SW2 «Сброс» позволяет повторить процесс ввода новых чисел и их отображение на дисплее.

2. Создать проект LAB8 в MPLAB IDE, разместить проект в папке MP-LAB8, составить и отладить программу, выполняющую заданные операции.

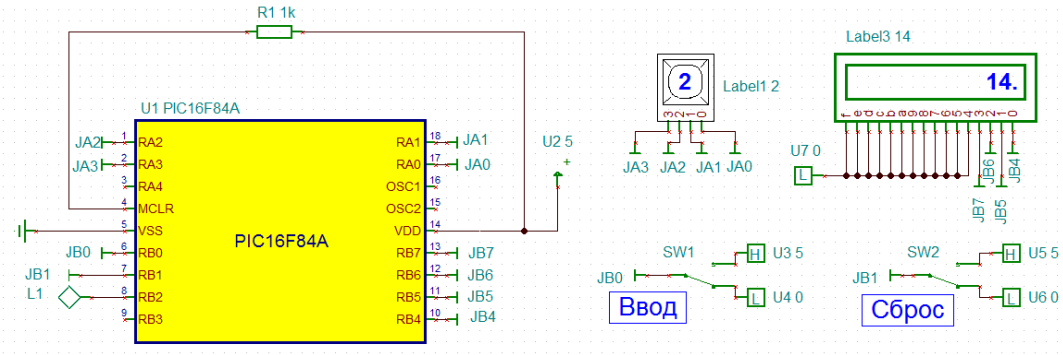


Рис.4.26. Схема модели РПЗУ

Возможный вариант программы представлен ниже на листингах.

В шапке программы (листинг 4.51) указаны регистры, применяемые для косвенной адресации (INDF, FSR) и работы с EEPROM (EECON1, EECON2, EEADR, EEDATA), управляющие биты, регистры данных и пр..

Листинг 4.51

```
LIST  P=PIC16F84A,  R=HEX; директива, определяющая
; тип процессора и систему счисления по умолчанию
; описание используемых регистров, переменных и
; адресов ячеек для хранения переменных пользователя
#include <P16F84.inc>
INTCON       EQU      0x0B
EECON1       EQU      0x88
EECON2       EQU      0x89
EEADR        EQU      0x09
EEDATA       EQU      0x08
OPTION_REG   EQU      0x81
STATUS       EQU      0x03
INDF         EQU      0x00
FSR          EQU      0x04
EEIF         EQU       4
```

```
WRERR       EQU      3
WREN        EQU      2
WR          EQU      1
RD          EQU      0
INTF        EQU      1
TOIF        EQU      5
GIE         EQU      7
EEIE        EQU      6
INTE        EQU      4
INTEDG      EQU      6
PCL         EQU     0x02
RP0         EQU      5
IRP         EQU      7
PORTA       EQU     0x05
PORTB       EQU     0x06
TRISA       EQU     0x85
TRISB       EQU     0x86
D           EQU      0
F           EQU      1
TEMPA       EQU     0x0C
TEMPB       EQU     0x0D
COUNT1      EQU     0x0E    ;  регистры данных
COUNT2      EQU     0x0F
COUNT3      EQU     0x10
INCOUNT     EQU     0x21
```

Исполняемая программа (листинг 4.52) содержит подпрограмму инициализации портов и регистров, блок ввода чисел в ОЗУ (листинг 4.53), блок записи в EEPROM (листинг 4.54), блок чтения из EEPROM и отображения чисел на дисплее (листинг 4.55).

Листинг 4.52

Исполняемая программа

```
;исполняемая программа
            ORG      0x000
            GOTO     BEGIN
            ORG      0x005
            ORG      0x004
            GOTO     INT_SERV
```

```
BEGIN
             CALL      INIT_PORTREG ;вызов
              ; подпрограммы инициализации портов
             GOTO      Label1
; подпрограмма инициализации портов и регистров
INIT_PORTREG
             MOVLW     0xFF ; установка портов в 1
             MOVWF     PORTA ; загрузка портов A и B
             MOVWF     PORTB
             BSF       STATUS,5 ; переход в банк 1
             MOVLW     0x1F ; RA0, RA1, RA2, RA3,
             MOVWF     TRISA ; RA4 - ввод
             MOVLW     0x03
             MOVWF     TRISB ; RB0,RB1-ввод, RB2-RB7
                       ; - вывод
             MOVLW     0xC0  ; B'11000000' в
                       ; OPTION_REG
             MOVWF     OPTION_REG ;  прерывание INT
                                  ; по перепаду 0-1
             BCF       STATUS,RP0 ; возврат в банк 0
             MOVLW     0x90  ; D0=B'10010000' - биты
              ; установки регистра INTCON
             MOVWF     INTCON ; разрешения прерываний
             RETURN    ; возврат из подпрограммы
```

Листинг 4.53

Ввод чисел в ОЗУ

```
Label1  ; ввод чисел в ОЗУ с косвенной адресацией

             MOVLW      0x00
             MOVWF      PORTB ; заслать 0 в порт B
             MOVLW      0x03  ; заслать 3 в W
             MOVWF      COUNT1 ; заслать 3 в счетчик
                              ; COUNT1
             BCF        PORTB,2 ; выключить индикатор
             MOVLW      INCOUNT   ; запись адреса в
             MOVWF      FSR  ;  регистр FSR
             GOTO       LOOP1

LOOP1        CLRWDT     ; сброс сторожевого таймера
```

```
            MOVLW     0x00 ; заслать 0 в W
            SUBWF     COUNT1,W ; вычесть W из
                               ; COUNT1
            BTFSC     STATUS,2 ; проверить нуль
                               ; по флагу Z
            GOTO      WR_EEPR  ; вызов пропрограммы
                          ; записи в EEPROM
            CALL      GET_RA   ; вызов подпрограммы
                               ; GET_RA
            GOTO      LOOP1
```

Листинг 4.54

Запись в EEPROM

```
WR_EEPR     BSF       PORTB,2  ; запись данных
                               ; в EEPROM
            MOVLW     0x03 ; заслать 3 в W
            MOVWF     COUNT1  ; заслать 3 в счетчик
                               ; COUNT1
            MOVLW     0x01 ; заслать начальный адрес
                               ; 0x01 в в EEADR
            MOVWF     EEADR ;
            MOVLW     INCOUNT ; запись адреса 0x21 в
                               ; регистр FSR
            MOVWF     FSR
LOOP2       MOVF      INDF,W ; переписать число с
                             ; адресом FSR в W
            MOVWF     EEDATA  ; и загрузить в EEDATA
            BSF       STATUS,RP0  ; переход в первый
                                  ; банк.
            BSF       EECON1,2 ; разрешить запись.
            MOVLW     0x55          ; обязательная
            MOVWF     EECON2        ; процедура
            MOVLW     0xAA          ; при записи.
            MOVWF     EECON2       ; ----"----
            BSF       EECON1,1     ; ----"----
B1          BTFSS     EECON1,4 ; проверка окончания
                             ; записи по флагу EEIF
            GOTO      B1  ; ожидание конца записи
            BCF       STATUS,RP0  ; переход в
                                  ; нулевой банк.
```

```
            INCF      FSR,F ; увеличить адрес ОЗУ
            INCF      EEADR,F ; увеличить адрес ПЗУ
            BSF       STATUS,RP0 ; переход в первый
                                 ; банк.
            BCF       EECON1,4 ; сбросить флаг
                           ; прерывания по окончании
            BCF       STATUS,RP0  ; переход в
                                  ; нулевой банк.
            DECFSZ    COUNT1,F ; уменьшить счетчик
            GOTO      LOOP2
            GOTO      RD_EEPR
```

В блоке записи в EEPROM на метке B1 организован цикл ожидания окончания записи по флагу EEIF.

Листинг 4.55

Чтение из EEPROM и отображения на дисплее

```
RD_EEPR      ; чтение из EEPROM
            MOVLW     0x03  ; загрузить 3 в COUNT1
            MOVWF     COUNT1
            MOVLW     0x01 ; загрузить начальный
                           ; адрес в COUNT2
            MOVWF     COUNT2

LOOP3
            BCF       STATUS,RP0  ; переход в
                                  ; нулевой банк.
            MOVF      COUNT2,D    ; записать адрес
            MOVWF     EEADR       ; в EEADR
            BSF       STATUS,RP0  ; переход в первый
                                  ; банк.
            BSF       EECON1,0 ; инициализировать
                               ; чтение.
            BCF       STATUS,RP0 ; переход в нулевой
                                 ; банк.
            MOVF      EEDATA,W ; скопировать в W из
                               ; EEPROM
            MOVWF     TEMPB    ; скопировать из W в
                               ; регистр TEMPB
            SWAPF     TEMPB,F  ; поменять полубайты
```

```
            MOVLW     0xF0  ; F0=B'11110000'
            ANDWF     TEMPB,0   ; очистить младшие
                                ; биты TEMPB
            MOVWF     PORTB  ; загрузить число в
                                ; PORTB
            CALL      DELAY_A ; задержка в дисплее
            INCF      COUNT2,F  ; увеличить адрес
                                 ; в ПЗУ
            DECFSZ    COUNT1,F ; уменьшить счетчик
                                ; COUNT1
            GOTO      LOOP3
            BTFSS     PORTB,1 ; проверка сброса
            GOTO      RD_EEPR
            GOTO      BEGIN
```

Подпрограмма GET_RA (листинг 4.56) выполняет чтение состояния порта A, к которому подключена HEX-клавиатура.

Листинг 4.56

```
GET_RA  ; подпрограмма чтения состояния порта A
            MOVF      PORTA,D  ; чтение состояния
                               ; порта A в W
            MOVWF     TEMPA ; пересылка W в TEMPA
            MOVLW     0x0F ; B'00001111' в W
            ANDWF     TEMPA,1  ; очистить старший
                               ; полубайт TEMPA
            RETURN     ; возврат из подпрограммы
```

Подпрограмма DELAY_A (листинг 4.57) устанавливает цикл вывода чисел на дисплей.

Листинг 4.57

```
DELAY_A ; подпрограмма формирования задержки
            MOVLW      0x10
            MOVWF      COUNT3 ; загрузка W в регистр
                              ; COUNT3
LOOP4
            DECFSZ     COUNT3,F ; декремент COUNT3
            GOTO       LOOP4 ; повторение цикла 10
                             ;раз
            RETURN     ; возврат из подпрограммы
```

Подпрограмма прерываний INT_SERV (листинг 4.58) выполняет прерывания по положительному фронту сигнала с ключа SW1, увеличивает адрес в регистре косвенной адресации и уменьшает число в счетчике COUNT1.

Листинг 4.58

```
INT_SERV
            MOVF        TEMPA,0
            MOVWF       INDF
            INCF        FSR,F ; увеличить адрес на 1
            DECF        COUNT1,F ; уменьшить COUNT1
            RETFIE      ; возврат из пп прерываний
```

Отладку программы можно сделать в MPLAB IDE. Для этого надо организовать непрерывное выполнение программы с записью постоянного числа 0x07, добавив отладочные команды (XXXXX) в листинги 4.53, 4.56 и 4.58

Отладка. Листинг 4.59

```
LOOP1       CLRWDT   ; сброс сторожевого таймера
            MOVLW     0x00 ; заслать 0 в W
            SUBWF     COUNT1,W ; вычесть W из COUNT1
            BTFSC      STATUS,2 ; проверить нуль по
                                ; флагу Z
            GOTO      WR_EEPR ; вызов пропрограммы
                              ; записи в EEPROM
            CALL      GET_RA  ; вызов подпрограммы
                              ; GET_RA
            CALL      INT_SERV ;XXXXXXXXXXXXXXXXXX
            GOTO      LOOP1
```

Отладка. Листинг 4.60

```
GET_RA  ; подпрограмма чтения состояния порта A
            MOVF      PORTA,D ; чтение состояния
                              ;порта A в W
            MOVLW     0x07 ;XXXXXXXXXXXXXXXXXX
            MOVWF     TEMPA  ; пересылка W в TEMPA
            MOVLW     0x0F   ; B'00001111' в W
            ANDWF     TEMPA,1 ; очистить старший
байт TEMPA
            RETURN  ; возврат из подпрограммы
```

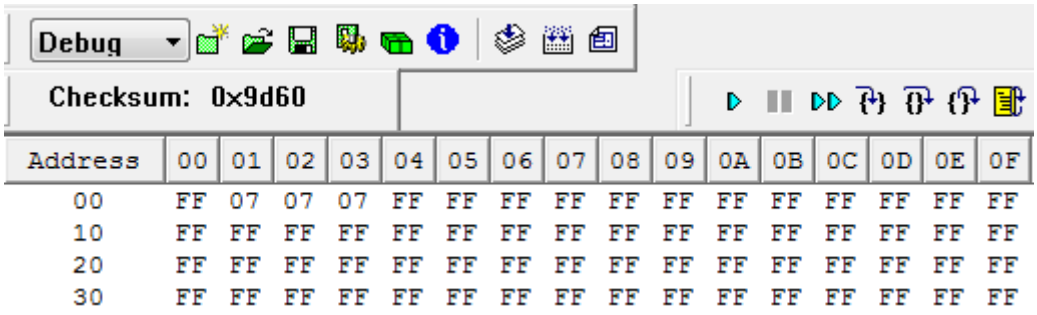
Отладка. Листинг 4.61

```
INT_SERV
                MOVF        TEMPA,0
                MOVWF       INDF
                INCF        FSR,F ; увеличить адрес на 1
                DECF        COUNT1,F ; уменьшить COUNT1
                RETURN       ; XXXXXXXXXXXXXX
                RETFIE  ; возврат из пп прерываний
```

Если программа работает правильно, число 07 будет записано в ОЗУ (рис.4.27) и в EEPROM (рис.4.28).

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	--	57	5C	1B	24	00	70	--	07	02	00	04	07	70	02	02
10	06	F0	1B	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	07	07	07	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Рис.4.27. Содержание ОЗУ при отладке



Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	FF	07	07	07	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
10	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
30	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Рис.4.28. Содержание EEPROM при отладке

Порт B должен содержать число 0x70, так на дисплей подается старший байт.

3. Убрать отладочные команды, загрузить программу в схему модели РПЗУ и проверить функционирование.

Записать в оперативную память три заданных для бригады числа, набирая их на HEX – клавиатуре. Переписать эти числа в EEPROM. Последовательно считывать эти числа с задержкой и выводить их на дисплей.

Убедиться в правильном функционировании макета РПЗУ.

Домашнее задание

1. Отчет должен содержать полный текст программы, скриншоты, подтверждающие правильное функционирование модели РПЗУ.

2. Внимательно изучить подпрограммы с косвенной адресацией, запись в EEPROM и чтение из EEPROM и уметь объяснить работу этих подпрограмм.

Контрольные вопросы

1. Что является признаком косвенной адресации ?

2. Какой регистр используют для формирования адреса при косвенной адресации ?

3. Для чего применяют память EEPROM и в чем ее особенности ?

4. Какой объем памяти данных EEPROM в кристалле PIC16F84A ?

5. Какие регистры используют для доступа к памяти EEPROM ?

6. Какие регистры используют для управления записью и чтением данных в EEPROM ?

7. Назовите биты регистров, управляющих записью и чтением в EEPROM, и объясните их назначение.

8. Объясните процедуру записи в EEPROM.

9. Объясните процедуру чтения из EEPROM.

10. Для чего требуется проверка окончания записи в EEPROM и как она выполняется ?

Глава 5. ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ НА ЯЗЫКЕ СИ

5.1. Язык программирования Си

В предыдущих главах мы программировали микроконтроллер на языке ассемблера и мысленно представляли, что происходит внутри микроконтроллера (в регистрах, в памяти и т.п.). Программа ассемблера в значительной степени привязана к микроконтроллеру или к группе, в которой он находится. Поэтому язык ассемблера называют машинно-ориентированным.

Для повышения эффективности и качества программирования надо использовать язык высокого уровня, который не зависит от архитектуры микроконтроллера и имеет синтаксис, ориентированный на решение различных задач.

Популярным языком у разработчиков микропроцессорных устройств стал язык Си. Язык Си (в иностранных книгах C) был изобретен и реализован Деннисом Ритчи (Dennis Ritchie) для компьютера DEC PDP-11.

Летом 1983 года был образован комитет по созданию для языка Си стандарта ANSI (*American National Standards Institute — Национальный институт стандартизации США*).

Стандарт ANSI был окончательно принят в декабре 1989 года. Версию Си, определенную стандартом 1989 года, обычно называют C89. В 1999 году был выпущен стандарт на язык Си++, который называют C99. Версия C89 является наиболее распространенной, служит основой для версии C99 и используется всеми компиляторами. Поэтому мы будем изучать примеры программирования микроконтроллеров на Си.

Программированию на языке Си посвящено много книг. В разных изданиях язык Си называют языком высокого или среднего уровня. Полное изучение языка потребуется тем, кто занимается разработкой больших и разнообразных по назначение программ для универсальных ЭВМ. Мы рассмотрим основные понятия языка Си, необходимые для программирования микроконтроллеров.

Процесс программирования микроконтроллера на языке Си с использованием программ MPLAB IDE, mikroC, TINA и Proteus можно разбить на следующие этапы:

1. Постановка задачи и разбиение ее на модули, в которых четко определены операции с известным набором входных и выходных данных.

2. Продумывание реализации каждого модуля.

3. Составление в редакторе исходного файла с текстом программы на языке Си.

4. Компиляция исходного файла в его эквивалент на языке ассемблера, отладка программы, устранение ошибок в орфографии и синтаксисе.

5. Ассемблирование и компоновка промежуточного файла для получения файла в HEX-кодах.

6. Загрузка итогового HEX-кода в память модели микроконтроллера в среде TINA или Proteus.

7. Запуск программы, ее тестирование и повторная отладка.

Составление программы на Си в MPLAB IDE с компилятором XC8 или в mikroC имеет некоторые особенности, о которых Вы узнаете позже. Но знание основ классического языка Си является необходимым для успешного программирования.

Далее мы приводим краткие справочные сведения о языке Си [16, 17].

5.2. Структура программы на языке Си

Любая программа на Си состоит из одной или нескольких функций. Обязательно должна быть определена единственная главная функция `main()`, именно с нее всегда начинается выполнение программы. В исходном тексте программы главная функция всегда содержит операторы, отражающие сущность решаемой задачи, чаще всего это вызовы функций. Структура программы Си содержит функции `f1()` – `fN()`, написанные программистом.

Первые строки заголовка программы (например, `#include <PIC.h>`, `#include <xc.h>`, `#include <stdio.h>`)

сообщают компилятору, что он должен включить информацию о стандартной библиотеке для PIC-микроконтроллеров, компиляторе, программах ввода-вывода. Эти строки встречается в начале многих исходных файлов Си –программ.

```
#include <stdio.h>
Объявление глобальных переменных
int main(список параметров)
{
  последовательность операторов
}
тип_возвращаемого_значения f1(список параметров)
{
  последовательность операторов
}
тип_возвращаемого_значения f2(список параметров)
{
  последовательность операторов
}
.
тип_возвращаемого_значения fN(список параметров)
{
  последовательность  операторов
}
```

5.3. Типы, операторы, выражения и директивы в языке Си

Ключевые слова. В табл. 5.1 перечислены 32 ключевых слова, определенные стандартом С89. Они же являются ключевыми словами языка Си как подмножества Си++. Набор ключевых слов вместе с формальным синтаксисом Си составляет язык программирования Си.

Кроме стандартных ключевых слов, многие компиляторы для лучшего функционирования в среде программирования разрешают дополнительно использовать некоторые нестандартные ключевые слова. Для наиболее эффективного использования возможностей конкретного компилятора программист обязательно должен ознакомиться с набором дополнительных ключевых слов.

В языке Си различаются верхний и нижний регистры символов: `else` — ключевое слово, а `ELSE` — нет. В программе ключевое слово может быть использовано только как ключевое слово, то есть никогда не допускается его использование в качестве переменной или имени функции.

Таблица 5.1

Ключевые слова языка Си

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizof	volatile
do	if	static	while

Комментарий – это поясняющий текст, который не учитывается при компиляции. Многострочные комментарии начинаются с символов /* и заканчиваются символом */. Однострочные комментарии начинаются с символов //.

Идентификатор – это последовательность букв, цифр и символов подчеркивания, которая используется для именования различных переменных, констант, функций, типов и т.д. Идентификатор не должен начинаться с цифры. Регистр букв имеет значение.

Литерал – постоянное значение некоторого типа, которое используется в выражениях. Например: 0xA – число 10 в шестнадцатиричной системе.

Оператор – это символ, указывающий компилятору, какие действия выполняются над операндами. Операторы, соединяющие операнды, образуют *выражения*. Выражения могут быть заключены в круглые скобки и отделяются друг от друга точкой с запятой (“;”). Операторы языка Си показаны в таблице 5.2.

Таблица 5.2

Операторы языка Си

Знак оператора	Назначение оператора
( )	Вызов функции
[ ]	Выделение элемента массива
.	Выделение элемента записи
->	Выделение элемента записи
!	Логическое отрицание
~	Поразрядное отрицание
-	Изменение знака
++	Увеличение на единицу
--	Уменьшение на единицу
&	Взятие адреса
*адрес	Обращение по адресу
(тип)	Преобразование типа (т.е. (float) a)

sizeof( )	Определение размера в байтах
*	Умножение
/	Деление
%	Определение остатка от деления
+	Сложение
-	Вычитание
<<	Сдвиг влево
>>	Сдвиг вправо
<	Меньше, чем
<=	Меньше или равно
>	Больше, чем
>=	Больше или равно
= =	Равно
!=	Не равно
&	Поразрядное логическое "И"
^	Поразрядное исключающее "ИЛИ"
|	Поразрядное логическое "ИЛИ"

&&	Логическое "И"
||	Логическое "ИЛИ"
?:	Условная (тернарная) операция
=	Присваивание
+=, - =, *=, /=, %=, <<=, >>=, &=, |=, ^=	Бинарные операции (например, a *= b (т.е. a = a * b) и т.д.)
,	Операция запятая

Таблица 5.3

Приоритеты операций в языке Си

	( ) [ ] -> .
	! ~ ++ -- - (type) * & sizeof * / % + - << >> < <= > >= == != & ^ | && || ?: = += -= *= /= , и т.д.

В табл. 5.3 приведены приоритеты всех операций, определенных в Си. Все операторы, кроме унарных и "?", связывают (присоединяют, ассоциируют) свои операнды слева направо. Унарные операторы (*, &, -) и "?" связывают (присоединяют, ассоциируют) свои операнды справа налево.

Последовательности выражений заключаются в фигурные скобки, которые обозначают границы функций и блоки выражений в циклических и условных конструкциях.

Типы и размеры данных

Стандарт С89 определяет пять фундаментальных типов данных: `char` — символьные данные, `int` — целые, `float` — с плавающей точкой, `double` — двойной точности, `void` — без значения.

Имеются также квалификаторы, которые можно использовать с этими типами данных, формируя другие типы (таблице 5.4.).

Таблица 5.4

Все типы данных, определенные стандартом Си

Тип	***Типичный размер в битах***	***Минимально допустимый диапазон значений***
`char`	8	от -127 до 127
`unsigned char`	8	от 0 до 255
`signed char`	8	от -127 до 127
`int`	16 или 32	от -32767 до 32767
`unsigned int`	16 или 32	от 0 до 65535
`signed int`	16 или 32	то же, что int
`short int`	16	от -32767 до 32767
`unsigned short int`	16	от 0 до 65535
`signed short int`	16	то же, что short int
`long int`	32	от -2 147 483 647 до 2

		147 483 647
long long int	64	от -(2^{63}-1) до (2^{63}-1), добавлен стандартом C99
signed long int	32	то же, что long int
unsigned long int	32	от 0 до 4 294 967 295
unsigned long long int	64	от 0 до (2^{64}-1), добавлен в C99
float	32	от 1E-37 до 1E+37, с точностью не менее 6 значащих десятичных цифр
double	64	от 1E-37 до 1E+37, с точностью не менее 10 значащих десятичных цифр
long double	80	от 1E-37 до 1E+37, с точностью не менее 10 значащих десятичных цифр

Тип `Char` занимает один байт, получил название от слова “Character” и часто используется в программах для хранения символов (кодов ASCII).

Данные могут иметь или не иметь (`unsigned`) знак, могут быть длинными (`long`) и двойной длины (`long long`).

Тип `void` служит для объявления функции, не возвращающей значения, или для создания универсального указателя.

Переменные

Переменная — это именованный участок памяти, в котором хранится значение, которое может быть изменено программой. Все переменные перед их использованием должны быть объявлены. Общая форма *объявления* имеет такой вид:

тип список_переменных;

Здесь *тип* означает один из базовых или объявленных программистом типов, а *список_переменных* состоит из одного или более идентификаторов, разделенных запятыми. Например:

```
int i,j,l;
short int si;
unsigned int ui;
```

В языке Си имена переменных, функций, меток и т.п. называются *идентификаторами*. Длина идентификатора (количество символов, из которых состоит идентификатор) является натуральным числом, обычно идентификатор представляет собой последовательность из одного или нескольких символов. Первый символ должен быть буквой или символом подчеркивания, последующие символы должны быть буквами, цифрами или символами подчеркивания. Например:

```
count, test23;
```

Верхний и нижний регистры рассматриваются как различные.

Объявление переменных может быть расположено в трех местах: внутри функции, в определении параметров функции и вне всех функций. Это - места объявлений, соответственно, локальных переменных, формальных параметров функций и глобальных переменных.

Переменные, объявленные внутри функций, называются локальными переменными. Локальную переменную можно использовать только внутри блока, в котором она объявлена. Иными словами, локальная переменная невидима за пределами своего блока.

Локальные переменные существуют только во время выполнения программного блока, в котором они объявлены, создаются они при входе в блок, а разрушаются — при выходе из него. Переменная, объявленная в одном блоке, не имеет никакого отношения к переменной с тем же именем, объявленной в другом блоке. Например, переменная `int x, j,` объявленная в блоке первой функции не мешает переменной с тем же именем в блоке второй функции.

```
void func1(void)
{
  int x;
```

```
  x = 10;
}

void func2(void)
{
  int x;

  x = -199;
}
```

Как правило, все локальные переменные функции чаще всего объявляются в самом начале функции, сразу после открывающейся фигурной скобки.

Если имена переменных, объявленных во внутреннем и внешнем блоках совпадают, то переменная внутреннего блока "прячет" переменную внешнего блока.

В стандарте С89 все локальные переменные должны быть объявлены в начале блока, до любого выполнимого оператора.

По умолчанию локальные переменные хранятся в стеке. Стек — динамически изменяющаяся область памяти.

Если функция имеет аргументы, значит должны быть объявлены переменные, которые примут их значения. Эти переменные называются *формальными параметрами* функции. Внутри функции они фигурируют как обычные локальные переменные и объявляются после имени функции внутри круглых скобок:

```
int is_in(char *s, char c)
{
}
```

Формальные параметры тоже являются динамическими переменными и разрушаются при выходе из функции.

В отличие от локальных, *глобальные переменные* видимы и могут использоваться в любом месте программы. Они сохраняют свое значение на протяжении всей работы программы. Чтобы создать глобальную переменную, ее необходимо объявить за пределами функции. Глобальная переменная может быть

использована в любом выражении, независимо от того, в каком блоке это выражение используется.

В следующем примере переменная `count` объявлена вне каких бы то ни было функций. Ее объявление расположено перед `main()`, однако, оно может находиться в любом месте перед первым использованием этой переменной, но только не внутри функции. Объявлять глобальные переменные рекомендуется в верхней части программы.

```
#include <stdio.h>
int count;  /* глобальная переменная count */

void func1(void);
void func2(void);

int main(void)
{
}
```

Если локальная и глобальная переменные имеют одно и то же имя, то при обращении к ней внутри блока, в котором объявлена локальная переменная, происходит ссылка на локальную переменную, а на глобальную переменную это никак не влияет.

Глобальные переменные используются в тех случаях, когда разные функции программы используют одни и те же данные.

Области видимости

Кроме терминов глобальная переменная и локальная переменная в языке Си предусмотрено более тонкое подразделение этих двух широких категорий. Стандарт Си определяет четыре типа областей видимости идентификаторов:

- файл (имя, объявленное вне всех блоков и классов, можно использовать в транслируемом файле, содержащем это объявление; такие имена называются глобальными (global)) (начинается в начале файла (*единица трансляции*) и кончается в конце файла);

- блок (начинается открывающейся фигурной скобкой "{" блока и кончается с его закрытием скобкой "}");
- прототип функции (идентификаторы, объявленные в прототипе функции, видимы внутри прототипа);
- функция (имена, объявленные в функции, могут быть использованы только в теле функции).

Квалификатор типа

В языке Си определяются квалификаторы типа, указывающие на доступность и модифицируемость переменной. Стандарт С89 определяет два квалификатора: `const` и `volatile`.

Переменная, к которой в объявлении (декларации) применен квалификатор `const`, не может изменять свое значение.

Например, в объявлении:

```
const int a=10;
```

создается переменная с именем `a`, причем ей присваивается начальное значение 10, которое в дальнейшем в программе изменить никак нельзя.

Квалификатор `volatile` указывает компилятору на то, что значение переменной может измениться независимо от программы, т.е. вследствие воздействия еще чего-либо, не являющегося оператором программы. Например, адрес глобальной переменной можно передать в подпрограмму операционной системы, следящей за временем, и тогда эта переменная будет содержать системное время. В этом случае значение переменной будет изменяться без участия какого-либо оператора программы.

Спецификаторы класса памяти

Стандарт Си поддерживает четыре спецификатора класса памяти:

```
extern, static, register, auto.
```

Эти спецификаторы сообщают компилятору, как он должен разместить соответствующие переменные в памяти. Общая форма объявления переменных при этом такова:

спецификатор_класса_памяти тип имя переменой;

Спецификатор класса памяти в объявлении всегда должен стоять первым.

Если нужно сослаться на переменную, определенную в другой части программы, необходимо объявить ее как внешнюю (extern). Например:

```
#include <stdio.h>

int main(void)
{
  extern int first, last; /* используются глобальные
переменные */
```

Переменные, объявленные со спецификатором static, хранятся постоянно внутри своей функции или файла.

В первых версиях компиляторов Си спецификатор register сообщал компилятору, что переменная должна храниться в регистре процессора, а не в оперативной памяти, как все остальные переменные. Спецификатор register можно применить только к локальным переменным и формальным параметрам функций.

Ниже приведен пример использования переменной, в объявлении которой применен спецификатор register; эта переменная используется в функции возведения целого числа m в степень. (Степень — натуральное число — представлена идентификатором e.)

```
int int_pwr(register int m,  register int e)
{
  register int temp;

  temp = 1;

  for(; e; e--) temp = temp * m;
  return temp;
}
```

Инициализация переменных

При объявлении переменной она может быть инициализирована. Для этого нужно после ее объявления

поставить знак равенства и константу, т.е. общая форма инициализации имеет следующий вид:

тип имя_переменной = константа;

Примеры инициализации переменных:

```
char ch = 'a';
int first = 0;
double count = 177.57;
```

Оператор присваивания

Оператор присваивания может присутствовать в любом выражении языка Си. Общая форма оператора присваивания:

```
имя_переменной=выражение;
```

Если в операции встречаются переменные разных типов, происходит *преобразование типов*. В операторе присваивания действует простое правило: значение выражения в правой части преобразуется к типу объекта в левой части.

```
int x;
char ch;
float  f;

void func(void)
{
  ch = x;     /* 1-я строка */
  x = f;      /* 2-я строка */
  f = ch;     /* 3-я строка */
  f = x;      /* 4-я строка */
```

В 1-й строке примера старшие двоичные разряды целой переменной `x` отбрасываются, а в `ch` заносятся младшие 8 бит. Если значение `x` лежит в интервале от 0 до 255, то `ch` и `x` будут идентичны и потери информации не произойдет. В противном случае в `ch` будут занесены только младшие разряды переменной `x`. Во 2-й строке в `x` будет записана целая часть числа `f`. В 3-й строке произойдет преобразование целого 8-разрядного числа, хранящегося в `ch`, в число в плавающем формате. В 4-й строке произойдет то же самое, только с 16-разрядным целым.

Оператор последовательного вычисления: оператор "запятая"

Выражение, стоящее справа после оператора "запятая", является значением всего разделенного запятыми выражения.

Оператор:

```
x = (y=3, y+1);
```

сначала присваивает `y` значение 3, а затем присваивает `x` значение 4. Скобки здесь обязательны, потому что приоритет оператора "запятая" меньший, чем оператора присваивания.

Выражения

Выражения состоят из операторов, констант, функций и переменных. В языке Си выражением является любая правильная последовательность этих элементов. Большинство выражений в языке Си по форме очень похожи на алгебраические, часто их и пишут, руководствуясь правилами алгебры. Необходимо быть внимательным и учитывать специфику выражений в языке Си.

Если в выражении встречаются переменные и константы разных типов, они преобразуются к одному типу. Компилятор преобразует "меньший" тип в "больший". Этот процесс называется *продвижением типов* (*type promotion*). После выполнения приведенных выше преобразований оба операнда относятся к одному и тому же типу, к этому типу относится и результат операции.

Можно принудительно преобразовать значение выражения к нужному ему типу, используя операцию приведения типов. Общая форма оператора явного приведения типа: например, запись `(float) x/2` преобразует значение `x/2` к типу `float`.

Одномерные массивы

Общая форма объявления одномерного массива имеет следующий вид:

```
тип имя_переменной [размер];
```

Здесь *тип* обозначает базовый тип массива, являющийся типом каждого элемента. *Размер* задает количество элементов

массива. Например, следующий оператор объявляет массив из 100 элементов типа `double` под именем `balance`:

```
double balance[100];
```

Размер массива должен быть указан явно с помощью выражения-константы, определяется во время компиляции и впоследствии остается неизменным.

Доступ к элементу массива осуществляется с помощью имени массива и индекса. Индекс элемента массива помещается в квадратных скобках после имени. Например, оператор

```
balance[3] = 12.23;
```

присваивает 3-му элементу массива `balance` значение 12.23.

Индекс первого элемента любого массива в языке Си равен нулю. Поэтому оператор

```
char p[10];
```

объявляет массив символов из 10 элементов — от `p[0]` до `p[9]`.

Логические значения ИСТИНА (True) и ЛОЖЬ (False) в языке Си

При выполнении многих операторов языка Си вычисляются значения условных выражений и в зависимости от полученного значения выбирается та или иная ветвь вычислительного процесса. Условное выражение может принимать одно из двух значений: ИСТИНА или ЛОЖЬ. В языке Си значение ИСТИНА представлено любым ненулевым значением, включая отрицательные числа. Значение ЛОЖЬ всегда представлено нулем. Такое представление логических значений ИСТИНА и ЛОЖЬ позволяет весьма эффективно программировать многие процедуры.

Условные операторы

В языке Си существуют два условных оператора: `if` и `switch`.

Общая форма оператора `if` следующая:

```
if (выражение) оператор;
```

```
else оператор;
```

Здесь оператор может быть только одним оператором, блоком операторов или отсутствовать (пустой оператор). Фраза `else` может вообще отсутствовать.

Если *выражение* истинно, то выполняется оператор или блок операторов, следующий за `if`. В противном случае выполняется оператор (или блок операторов), следующий за `else` (если эта фраза присутствует).

Условное выражение, входящее в `if`, должно иметь скалярный результат. Это значит, что результатом должно быть целое число, символ, указатель или число с плавающей точкой, но им не может быть массив или структура.

Вложенные условные операторы `if`

Оператор `if` является вложенным, если он вложен, т.е. находится внутри другого оператора `if` или `else`. Во вложенном условном операторе фраза `else` всегда ассоциирована с ближайшим `if` в том же блоке, если этот `if` не ассоциирован с другой фразой `else`.

Лестница `if-else-if`

В программах часто используется конструкция, которую называют *лестницей* `if-else-if`. Общая форма лестницы имеет вид

```
if (выражение) оператор;
else
  if (выражение) оператор;
  else
    if (выражение) оператор;
    .
    .
    .
    else оператор;
```

Работает эта конструкция следующим образом. Условные выражения операторов `if` вычисляются сверху вниз. После выполнения некоторого условия, т.е. когда встретится выражение, принимающее значение ИСТИНА, выполняется

ассоциированный с этим выражением оператор, а оставшаяся часть лестницы пропускается. Если все условия ложны, то выполняется оператор в последней фразе `else`, а если последняя фраза `else` отсутствует, то в этом случае не выполняется ни один оператор.

Оператор выбора – `switch`

Оператор выбора `switch` предназначен для выбора ветви вычислительного процесса исходя из значения управляющего выражения. При этом значение управляющего выражения сравнивается со значениями в списке целых или символьных констант. Если будет найдено совпадение, то выполнится ассоциированный с совпавшей константой оператор. Общая форма оператора `switch` следующая:

```
switch (выражение) {
  case постоянная1:
    последовательность операторов
    break;
  case постоянная2:
    последовательность операторов
    break;
  case постоянная3:
    последовательность операторов
    break;
  default:
    последовательность операторов;
}
```

Значение *выражения* оператора `switch` должно быть таким, чтобы его можно было выразить целым числом. Это означает, что в управляющем выражении можно использовать переменные целого или символьного типа, но только не с плавающей точкой. Значение управляющего *выражения* по очереди сравнивается с *постоянными* в операторах `case`. Если значение управляющего *выражения* совпадет с какой-то из *постоянных*, управление передается на соответствующую метку `case` и выполняется *последовательность операторов* до оператора `break`. Если оператор `break` отсутствует, выполнение

последовательности операторов продолжается до тех пор, пока не встретится `break` (в другой метке) или не кончится тело оператора `switch` (т.е. блок, следующий за `switch`). Оператор `default` выполняется в том случае, когда значение управляющего выражения не совпало ни с одной постоянной. Оператор `default` также может отсутствовать. В этом случае при отсутствии совпадений не выполняется ни один оператор.

Операторы цикла

В языке Си, как и в других языках программирования, операторы цикла служат для многократного выполнения последовательности операторов до тех пор, пока выполняется некоторое условие. Условие может быть установленным заранее (как в операторе `for`) или меняться при выполнении тела цикла (как в `while` или `do-while`).

Цикл `for`

Общая форма оператора `for` следующая:

```
for (инициализация; условие; приращение) оператор;
```

Цикл `for` может иметь большое количество вариаций. В наиболее общем виде принцип его работы следующий. *Инициализация* — это присваивание начального значения переменной, которая называется параметром цикла. *Условие* представляет собой условное выражение, определяющее, следует ли выполнять *оператор* цикла (часто его называют *телом цикла*) в очередной раз. Оператор *приращение* осуществляет изменение параметра цикла при каждой итерации. Эти три оператора (они называются также *секциями* оператора `for`) обязательно разделяются точкой с запятой. Цикл `for` выполняется, если выражение *условие* принимает значение ИСТИНА. Если оно хотя бы один раз примет значение ЛОЖЬ, то программа выходит из цикла и выполняется оператор, следующий за телом цикла `for`.

В следующем примере в цикле `for` выводятся на экран числа от 1 до 100:

```
#include <stdio.h>
```

```
int main(void)
{
  int x;

  for(x=1; x <= 100; x++) printf("%d ", x);

  return 0;
}
```

В этом примере параметр цикла `x` инициализирован числом 1, а затем при каждой итерации сравнивается с числом 100. Пока переменная `x` меньше 100, вызывается функция `printf()` и цикл повторяется. При этом x увеличивается на 1 и опять проверяется условие цикла `x <= 100`. Процесс повторяется, пока переменная `x` не станет больше 100. После этого процесс выходит из цикла, а управление передается оператору, следующему за ним. В этом примере параметром цикла является переменная `x`, при каждой итерации она изменяется и проверяется в секции условия цикла.

Бесконечный цикл

Для создания бесконечного цикла можно использовать любой оператор цикла, но чаще всего для этого выбирают оператор `for`. Так как в операторе `for` может отсутствовать любая секция, бесконечный цикл проще всего сделать, оставив пустыми все секции. Это хорошо показано в следующем примере:

```
for( ; ; ) printf("Этот цикл крутится
бесконечно.\n");
```

Если условие цикла `for` отсутствует, то предполагается, что его значение — ИСТИНА. В оператор `for` можно добавить выражения инициализации и приращения, хотя обычно для создания бесконечного цикла используют конструкцию `for( ; ; )`.

Цикл `while`

Обшая форма цикла `while` имеет следующий вид:

```
while (условие) оператор;
```

Здесь *оператор* (тело цикла) может быть пустым оператором, единственным оператором или блоком. *Условие* (управляющее выражение) может быть любым допустимым в языке выражением. *Условие* считается истинным, если значение выражения не равно нулю, а *оператор* выполняется, если условие принимает значение ИСТИНА. Если условие принимает значение ЛОЖЬ, программа выходит из цикла и выполняется следующий за циклом оператор.

Цикл `do-while`

В отличие от циклов `for` и `while`, которые проверяют свое условие перед итерацией, `do-while` делает это после нее. Поэтому цикл `do-while` всегда выполняется как минимум один раз. Общая форма цикла `do-while` следующая:

```
do {
оператор;
} while (условие);
```

Если оператор не является блоком, фигурные скобки не обязательны, но их почти всегда ставят, чтобы оператор достаточно наглядно отделялся от условия. Итерации оператора `do-while` выполняются, пока условие не примет значение ЛОЖЬ.

Операторы перехода

В языке Си определены четыре оператора перехода: `return`, `goto`, `break` и `continue`. Операторы `return` и `goto` можно использовать в любом месте внутри функции. Операторы `break` и `continue` можно использовать в любом из операторов цикла, причем `break` можно также использовать в операторе `switch`.

Оператор `return`

Оператор `return` используется для выхода из функции. Он заставляет программу перейти в точку вызова функции. Оператор `return` может иметь ассоциированное с ним значение, тогда при выполнении данного оператора это значение возвращается в

качестве значения функции. В функциях типа `void` используется оператор `return` без значения.

Общая форма оператора `return` следующая:

```
return выражение;
```

Выражение присутствует только в том случае, если функция возвращает значение. Это значение *выражения* становится возвращаемым значением функции.

Оператор `goto`

Кроме `goto`, в языке Си есть другие операторы управления (например `break`, `continue`), поэтому необходимости в применении `goto` практически нет.

Для оператора `goto` всегда необходима метка. *Метка* — это идентификатор с последующим двоеточием. Метка должна находится в той же функции, что и `goto`, переход в другую функцию невозможен. Общая форма оператора `goto` следующая:

```
goto метка;
.
метка:
```

Метка может находиться как до, так и после оператора `goto`. Например, используя оператор `goto`, можно выполнить цикл от 1 до 100:

```
x = 1;
loop1:
  x++;
  if(x<=100) goto loop1;
```

Оператор `break`

Оператор `break` применяется в двух случаях. В операторе `switch` с его помощью прерывается выполнение последовательности `case`. В этом случае оператор `break` не передает управление за пределы блока. Во-вторых, оператор `break` используется для немедленного прекращения выполнения цикла без проверки его условия, в этом случае оператор `break` передает управление оператору, следующему после оператора цикла.

Например, программа

```
#include <stdio.h>

int main(void)
{
  int t;

  for(t=0; t<100; t++) {
    printf("%d ", t);
    if(t==10) break;
  }

  return 0;
}
```

выводит на экран числа от 0 до 10. После этого выполнение цикла прекращается оператором `break`, условие `t < 100` при этом игнорируется.

Функция `exit()`

Эта функция вызывает немедленное прекращение работы всей программы и передает управление операционной системе.

Общая форма функции `exit()` следующая:

```
void exit (int код_возврата);
```

Оператор `continue`

Оператор `continue` немного похож на `break`. Оператор `break` вызывает прерывание цикла, а `continue` — прерывание текущей итерации цикла и осуществляет переход к следующей итерации. При этом все операторы до конца тела цикла пропускаются. В цикле `for` оператор `continue` вызывает выполнение операторов приращения и проверки условия цикла. В циклах `while` и `do-while` оператор `continue` передает управление операторам проверки условий цикла.

Оператор-выражение

Любое выражение, которое заканчивается точкой с запятой, является оператором. Например:

```
func();  /* вызов функции */
```

```
a = b+c; /* оператор присваивания */
```

Директивы препроцессора

В исходный код программы на языке Си можно вставлять различные инструкции компилятору. Они называются *директивами препроцессора* и расширяют возможности среды программирования.

Имеются следующие директивы препроцессора:

```
#define     #endif     #ifdef      #line
#elif       #error     #ifndef     #pragma
#else       #if        #include    #undef
```

Все они начинаются со знака `#`. Кроме того, каждая директива препроцессора должна занимать отдельную строку.

Директива `#define` определяет идентификатор и последовательность символов, которая будет подставляться вместо идентификатора каждый раз, когда он встретится в исходном файле. Идентификатор называется *именем макроса*, а сам процесс замены — *макрозаменой*. В общем виде директива выглядит таким образом:

```
#define имя_макроса последовательность_символов
```

В этом выражении нет точки с запятой. Между идентификатором и последовательностью символов *последовательность_символов* может быть любое количество пробелов. Признаком конца последовательности символов может быть только разделитель строк.

Предположим, например, что вместо значения 1 нужно использовать слово `LEFT` (левый), а вместо значения 0 — слово `RIGHT` (правый). Тогда можно сделать следующие объявления с помощью директивы `#define`:

```
#define LEFT 1
#define RIGHT 0
```

В результате компилятор будет подставлять 1 или 0 каждый раз, когда в вашем файле исходного кода встречается идентификатор соответственно `LEFT` или `RIGHT`.

Директива `#include`

Директива `#include` дает указание компилятору читать еще один исходный файл — в дополнение к тому файлу, в котором находится сама эта директива. Имя исходного файла должно быть заключено в двойные кавычки или в угловые скобки. Например, обе директивы

```
#include "stdio.h"
#include <stdio.h>
```

дают компилятору указание читать и компилировать заголовок для библиотечных функций системы ввода/вывода.

Файлы, имена которых находятся в директивах `#include`, могут в свою очередь содержать другие директивы `#include`. Они называются *вложенными директивами* `#include`.

Способ поиска файла зависит от того, заключено ли его имя в двойные кавычки или же в угловые скобки. Если имя заключено в угловые скобки, то поиск файла проводится тем способом, который определен в компиляторе. Если имя заключено в кавычки, то поиск файла проводится другим способом. Во многих компиляторах это означает поиск файла в текущем рабочем каталоге. Если же файл не найден, то поиск повторяется уже так, как будто имя файла заключено в угловые скобки.

Директивы условной компиляции

Имеется несколько директив, которые дают возможность выборочно компилировать части исходного кода вашей программы. Этот процесс называется *условной компиляцией* и широко используется фирмами, живущими за счет коммерческого программного обеспечения — теми, которые поставляют и поддерживают многие специальные версии одной программы.

Директивы `#if`, `#else`, `#elif` и `#endif`

Директивами условной компиляции являются `#if`, `#else`, `#elif` и `#endif`. Они дают возможность в зависимости от значения константного выражения включать или исключать те или иные части кода.

В общем виде директива #if выглядит таким образом:

```
#if константное выражение
    последовательность операторов
#endif
```

Если находящееся за #if константное выражение истинно, то компилируется код, который находится между этим выражением и #endif. В противном случае этот промежуточный код пропускается. Директива #endif обозначает конец блока #if.

Директива #else задает альтернативу на тот случай, если не выполнено условие #if.

Например можно записать:

```
/* Простой пример #if/#else. */
#include <stdio.h>

#define MAX 10

int main(void)
{
#if MAX>99
  printf("Компилирует для массива, размер которого
больше 99.\n");
#else
  printf("Компилирует для небольшого массива.\n");
#endif

  return 0;
}
```

В этом случае выясняется, что MAX меньше 99, поэтому часть кода, относящаяся к #if, не компилируется. Однако компилируется альтернативный код, относящийся к #else, и откомпилированная программа будет отображать сообщение «Компилируется для небольшого массива».

Директива #elif означает "else if" и устанавливает для множества вариантов компиляции цепочку if-else-if. После #elif находится константное выражение. Если это выражение истинно, то компилируется находящийся за ним блок кода, и

больше не проверяются никакие другие выражения `#elif`. В противном же случае проверяется следующий блок этой последовательности. В общем виде `#elif` выглядит таким образом:

```
#if выражение
    последовательность операторов
#elif выражение 1
    последовательность операторов
#elif выражение 2
    последовательность операторов
#elif выражение 3
    последовательность операторов
#elif выражение 4
.
.
#elif выражение N
    последовательность операторов
#endif
```

Директивы `#ifdef` и `#ifndef`

Другой способ условной компиляции — это использование директив `#ifdef` и `#ifndef`, которые соответственно означают "`if defined`" (если определено) и "`if not defined`" (если не определено). В общем виде `#ifdef` выглядит таким образом:

```
#ifdef имя_макроса
       последовательность операторов
#endif
```

Блок кода будет компилироваться, если *имя макроса* было определено ранее в операторе `#define`.

В общем виде оператор `#ifndef` выглядит таким образом:

```
#ifndef имя_макроса
        последовательность операторов
#endif
```

Блок кода будет компилироваться, если имя макроса еще не определено в операторе `#define`.

И в `#ifdef`, и в `#ifndef` можно использовать оператор `#else` или `#elif`.

Директива #undef

Директива #undef удаляет ранее заданное определение имени макроса, то есть "аннулирует" его определение; само имя макроса должно находиться после директивы. В общем виде директива #undef выглядит таким образом:

```
#undef имя_макроса
```

Контрольные вопросы

1. Какую структуру имеют программы, написанные на языке Си ?
2. Назовите несколько ключевых слов языка Си.
3. Как надо выделять комментарии в программе на Си ?
4. Что такое идентификатор ?
5. Что такое оператор ? Назовите несколько операторов языка Си .
6. Какие типы данных применяют в языке Си ?
7. Как объявляют переменные в программе на Си ?
8. Что такое локальные и глобальные переменные ?
9. Как оператор присваивания преобразует типы переменных ?
10. Назовите условные операторы и объясните как их применяют.
11. Назовите операторы цикла и объясните их применение.
12. Назовите операторы перехода и объясните их применение.
13. Что такое директивы препроцессора ?
14. Для чего применяют директиву #include ?

5.4. Компилятор MPLAB XC8 C и первая программа на языке Си

Составим программу на языке Си для мигания светодиода с управлением частотой (рис.5.1). ё

Как и в лабораторной работе №7 требуется управлять частотой мигания светодиода VD2, используя прерывания по входу INT и по изменению уровня сигнала на входе RB6.

Установка компилятора MPLAB XC8 C

Перед составлением программы на языке Си мы должны выбрать компилятор, который будет работать с программой MPLAB IDE, и ознакомиться хотя бы с его кратким описанием. Это необходимо потому, что компиляторы устанавливают дополнительные требования к структуре программы, описанию регистров специального назначения, битов конфигурации, реализации основного цикла, использования прерываний и т.п.

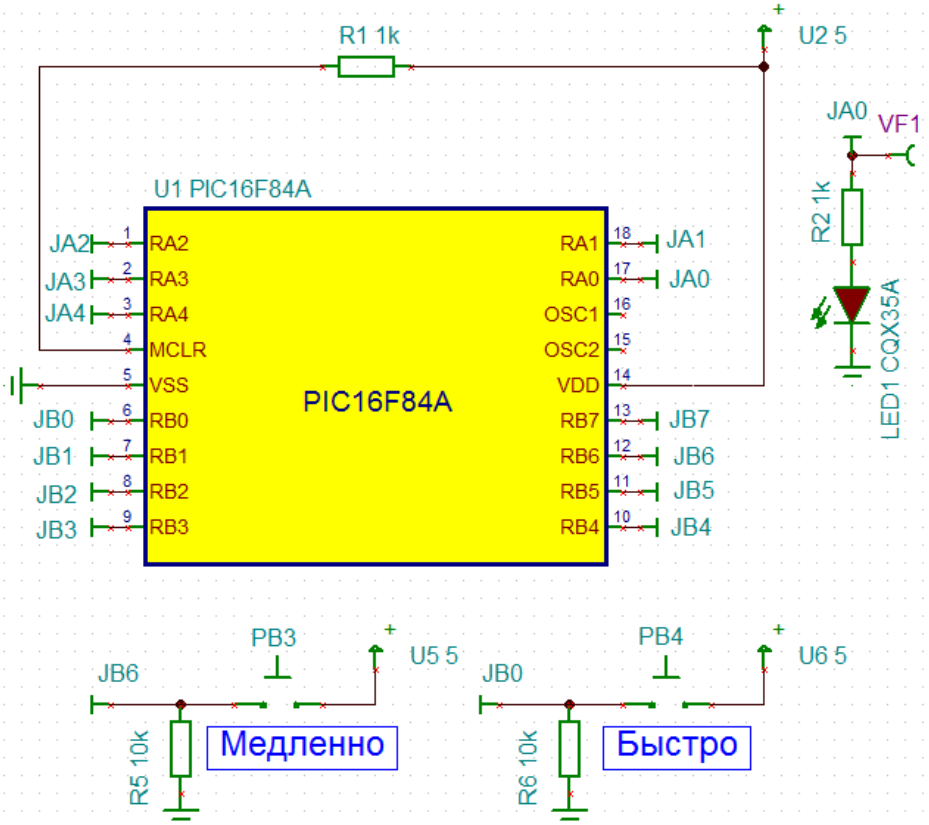


Рис.5.1. Схема модели макета с прерываниями

Мы выбрали бесплатный компилятор MPLAB XC8 C компании Microchip. Полное описание этого компилятора составляет более 500 листов [4]. Более краткое описание дано в [5]. Этот компилятор используют для программирования 8-битных микроконтроллеров.

Программу компилятора надо установить на компьютер, в котором уже работает MPLAB IDE. В папке Microchip программных файлов находится описание компилятора.

После установки компилятора создаем проект LAB9 в MPLAB IDE, разместив его в папке MP-LAB9. Особенность проявляется в том, что на втором шаге надо выбрать в языковых инструментах Microchip XC8 (рис. 5.2).

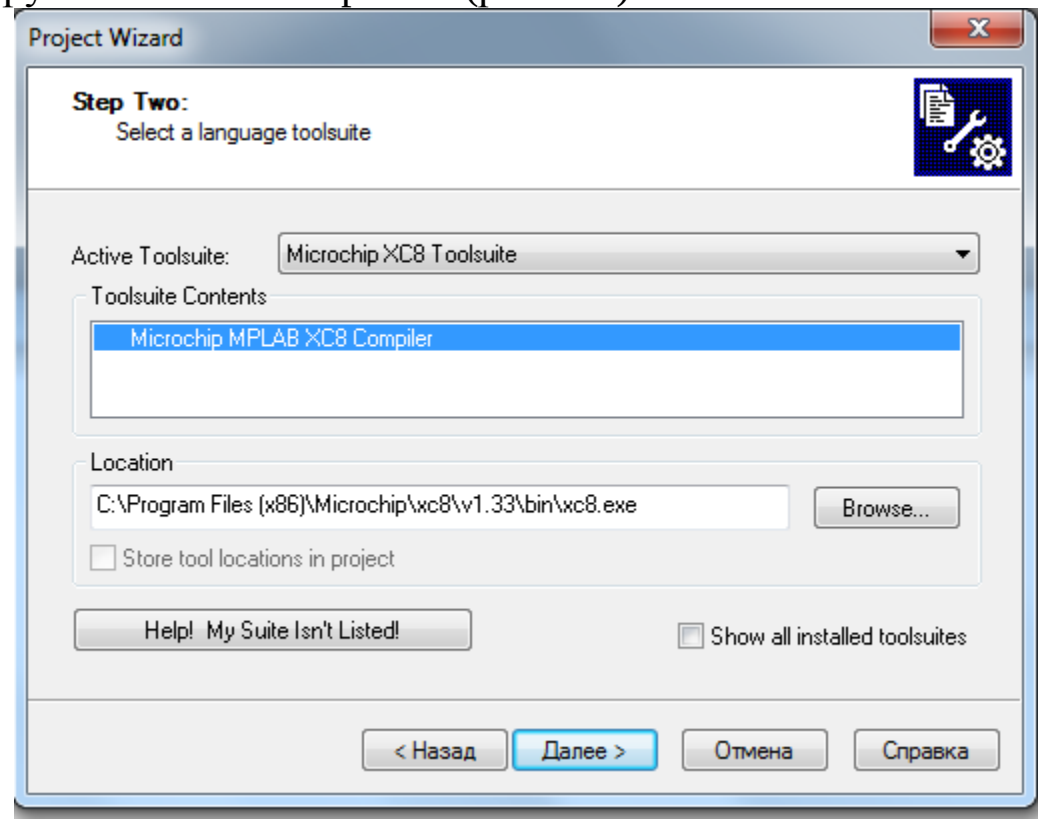


Рис. 5.2. Выбор в проекте компилятора Microchip XC8

В результате этого в меню Debugger появятся две дополнительные кнопки Build with MPLAB XC8 C Compiler V1.33 и Rebuild with MPLAB XC8 C Compiler V1.33 (рис. 5.3).

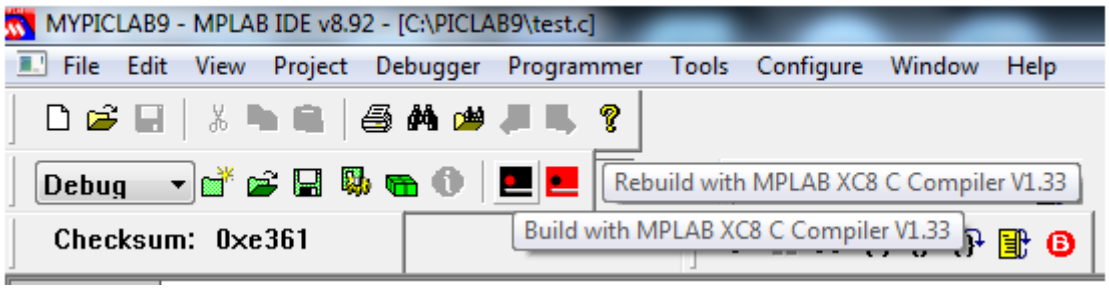


Рис. 5.3. Меню отладчика после установки компилятора XC8

Основные правила при работе с компилятором XC8

Все компиляторы MPLAB XC C используют Общий интерфейс Си (CCI), чтобы улучшить переносимость кода между разными микроконтроллерами. CCI предполагает, что наш исходный код уже соответствует стандарту ANSI. CCI добавляется к стандарту ANSI и облегчает программистам возможность использования программы на всех микроконтроллерах Microchip при использовании компиляторов MPLAB XC C.

При составлении программы на языке Си для компилятора MPLAB XC C в заголовке надо записать две директивы:

```
#include <PIC.h>
#include <xc.h>
```

Первая директива подключает к программе файлы описания микроконтроллера, который как устройство выбран при запуске MPLAB IDE. Вторая директива дает программе доступ к компилятору и его библиотекам. Эта директива должна быть во всех программах Си.

Наша первая программа для управления миганием светодиода с использованием прерываний на языке Си может быть записана так (Листинг 5.1):

Листинг 5.1

```
#include <PIC.h> // включение информации о библиотеке PIC
#include <xc.h>  //включение библиотеки компилятора
#pragma config WDTE=OFF, FOSC=XT, CP=OFF /*установка
битов конфигурацииции*/
//-------------------------------------------------
 // Мигалка с  прерываниями   на  СИ

int count, n, x;   //объявление глобальных переменных
void main(void)  //определение главной функции main

{
 count=2;
 TRISA=0x00; //Порт А - все выход
 TRISB=0xFF; // Порт В - RB0, RB4-RB6 -все входы
```

```
 INTCONbits.RBIE=1; /* разрещены прерывания
                     RB0,RB4-RB7*/
 INTCONbits.INTE=1; /* прерывания разрешены
                    по входу INT*/
 INTCONbits.GIE=1;
 RA0=0;
 PORTB=0;
   while(1)
     {
      n=count;
   while(n)
     {
   n--;
    }
      RA0^=1;  // инвертируем вывод RA0
    }
   }

void interrupt myIsr(void) /*программа обслуживания
                            прерываний*/
{extern int count;
    if(INTCONbits.INTF)  /*проверка прерывания
                          по входу INT*/
      count=5;
    else
      count=20;

      INTCONbits.INTF = 0;/*сброс флага прерываний
                           по входу INT*/
      INTCONbits.RBIF=0;  /*сброс флага прерываний
                           по входам RB4-RB7*/
      INTCONbits.RBIE=1; /* разрещены прерывания
                           RB0,RB4-RB7*/
      INTCONbits.INTE=1; /* разрешено прерывание
                          по входу INT*/
      PORTB=0;           //очистить порт B
}
```

Поясним особенности программы, обусловленные применением компилятора MPLAB XC8 и изучением его описания.

Биты конфигурации могут быть установлены в проекте MPLAB IDE на вкладке *Configure-Configuration Bits*. Для микроконтроллера PIC16F84A биты конфигурации устанавливают в регистр в адресом 0x2007.

Если надо записать биты конфигурации непосредственно в программу Си, используют директиву `#pragma config`.

Например:

```
#pragma config WDTE=OFF, FOSC=XT, CP=OFF
```

В программе используются целые переменные `(int count, n;)`. Инициализацию портов выполняют команды `TRISA=0x00` и `TRISB=0xFF`. Причем не требуется изменять бит RP0 регистра STATUS. Компилятор сделает это сам.

Для установки или чтения бита в регистре INTCON использованы команды `INTCONbits.INTE=1` и `INTCONbits.INTF`.

Для обслуживания прерываний применяется функция `void interrupt myIsr(void)`, где спецификатор прерываний `interrupt` преобразует функцию `myIsr ( )` в функцию обработки прерываний.

Приведем еще некоторые полезные примеры директив компилятора XC8.

Использование памяти EEPROM

Если данные должны быть записаны в EEPROM, используют спецификатор `__eeprom`.

Например:

```
__eeprom int serialNos[4];
```

Переменные, квалифицированные как `eeprom`, сначала помещаются в HEX-файл и «прожигаются» в EEPROM во время прошивки микроконтроллера. Поэтому, во время исполнения программы изменение данных в EEPROM требует выполнения функции записи в EEPROM и чтения из EEPROM.

Массивы в EEPROM определяют таким образом:

```
eeprom char regNumber[10] = "A93213";
eeprom int lastValues[3];
```

В обоих случаях начальные значения будут помещены в определенную секцию программы в виде HEX-файла. В качестве начальных значений номера элемента массива используются нули.

Не следует применять переменные `eeprom` в сложных выражениях, так как код программы получится значительно длиннее.

Для тех устройств, которые поддерживают внешнее программирование их области данных EEPROM, можно использовать макрос `__ EEPROM_DATA ()`, чтобы поместить начальные значения в готовый HEX-файл.

Макрос используется следующим образом:

```
#include <xc.h>
__ EEPROM_DATA (0, 1, 2, 3, 4, 5, 6, 7);
```

У макроса есть восемь параметров, которые представляют восемь значений данных. Каждое значение должно быть размером в байт. Неиспользованные значения должны быть определены нулем. Макрос можно использовать многократно, но во время выполнения программы загружать новые данные в макрос нельзя.

Функции доступа к EEPROM

Для того, чтобы во время выполнения программы читать и записывать данные в EEPROM используют библиотечные функции `eeprom_read ()` и `eeprom_write ()`.

Примеры записи этих функций:

```
#include <xc.h>
unsigned char eeprom_read(unsigned char address);
void eeprom_write(unsigned char address, unsigned
char value);
```

Функция `eeprom_write ()` будет инициировать процесс записи в EEPROM. Длительность процесса записи может быть

достаточно продолжительной. Завершение записи индицируется битом WR=0 в регистре EECON.

Многие микроконтроллеры PIC могут поддерживать макросы доступа к EEPROM. Записывают эти макросы следующим образом:

```
EEPROM_READ(address)
EEPROM_WRITE(address, value)
```

Чтобы гарантировать, что все записи в EEPROM завершились до выполнения EEPROM_READ (), надо опросить соответствующий флаг.

Например так:

```
xc.h // wait for end-of-write before EEPROM_READ
while(WR)
continue; // read from EEPROM at address
value = EEPROM_READ(address);
```

Команды ассемблера в программе Си

Инструкции ассемблера можно непосредственно встроить в код Си, используя директивы `#asm, #endasm` или оператор `asm ();`.

Директивы `#asm и #endasm` используют вначале и в конце блока инструкций ассемблера.

Оператор `asm ();` используют, чтобы встроить ассемблерные инструкции в программу с кодом Си. Эта форма выглядит и ведет себя как оператор Си. Обычно одна инструкция помещается в строку, но можно записать больше инструкций, разделяя их \n символом.

Наример:

```
asm ("MOVLW 55\n MOVWF _x") ;
```

Следующий пример показывает применение обеих методов:

```
void main(void)
{
var = 1;
#asm // like this...
BCF 0,3
BANKSEL(_var)

RLF (_var)&07fh
```

```
RLF (_var+1)&07fh
#endasm
// do it again the other way...
asm("BCF 0,3");
asm("BANKSEL _var");
asm("RLF (_var)&07fh");
asm("RLF (_var+1)&07fh");
}
```

5.5. Лабораторная работа №9

Компиляция и отладка программы с прерываниями на языке Си с использованием стимулов

Цель работы: изучение компиляции и отладки программ с прерываниями на языке Си в MPLAB IDE с компилятором MPLAB XC8 C с использованием стимулов для контроля прерываний.

Лабораторное задание

1. Записать программу из листинга 5.1 в текстовом редакторе *Notepad++* и сохранить в папке MP-LAB9 с расширением «.c» как LAB9.c.

2. Создать в MPLAB IDE проект LAB9 для микроконтроллера PIC16F84A и разместить его в папке MP-LAB9. На втором шаге надо выбрать в языковых инструментах Microchip XC8 Toolsuite. Добавить в проект файл LAB9.c. и скопировать этот файл в проект, установив «C» в окне выбранного файла.

3. В созданном проекте выбрать *Debbuger-Select Tool-MPLAB SIM*. Открыть файл программы и выполнить компиляцию, нажав кнопку  .

4. Если компилятор обнаружит ошибки, сделать исправления и добиться успешного результата.

5. Установить в программе две точки останова (рис. 5.4). Сделать сброс и запустить программу, выполнив *Run*. Убедиться, что программа находится в циклах с операторами `while(1)`, `while(n)`.

6. В программе MPLAB IDE открыть *Debugger-Stimulus-New Workbook*. В окне *Stimulus* (рис. 5.5) выбрать асинхронные стимулы *Asynch* и установить импульсные воздействия на выводах RB0 и RB6.

```
    }
    RA0^=1;   // инвертируем вывод

    INTCONbits.INTF = 0;//сброс
    INTCONbits.RBIF=0;  //сброс
```

Рис. 5.4. Точки останова в программе с прерываниями

7. Включить *Fire* для RB0. Убедиться, что начинает выполняться подпрограмма обслуживания прерываний. После этого, используя *Debugger-StopWatch*, определить период быстрых миганий светодиода для тактовой частоты микроконтроллера 20 МГц (рис. 5.5). Период мигания соответствует времени двукратного прохода через первую точку останова.

8. Включить *Fire* для RB6. После выполнения программы обслуживания прерываний повторно определить период медленных миганий светодиода.

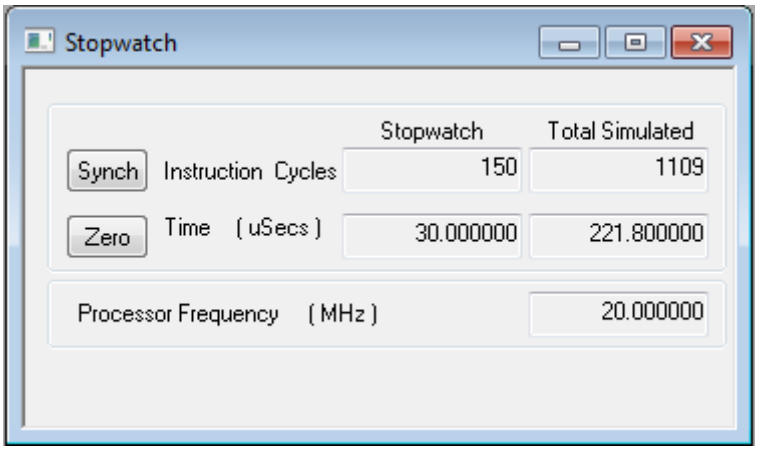


Рис. 5.5. Определение периода мигания светодиода

9. В окне установки стимулов выбрать вкладку *Pin/Register Actions* и установить действия на вводах RB0 и RB6 на заданных циклах выполнения программы с повторением после 500 циклов

(Рис.5.7). Оставить в программе одну точку останова на операторе `INTCONbits.INTF=0`. В меню *Debugger* открыть *StopWatch*, сделать сброс программы. Нажимая *Run* и *Step Over*, наблюдать и записать циклы остановок в подпрограмме обслуживания прерываний.

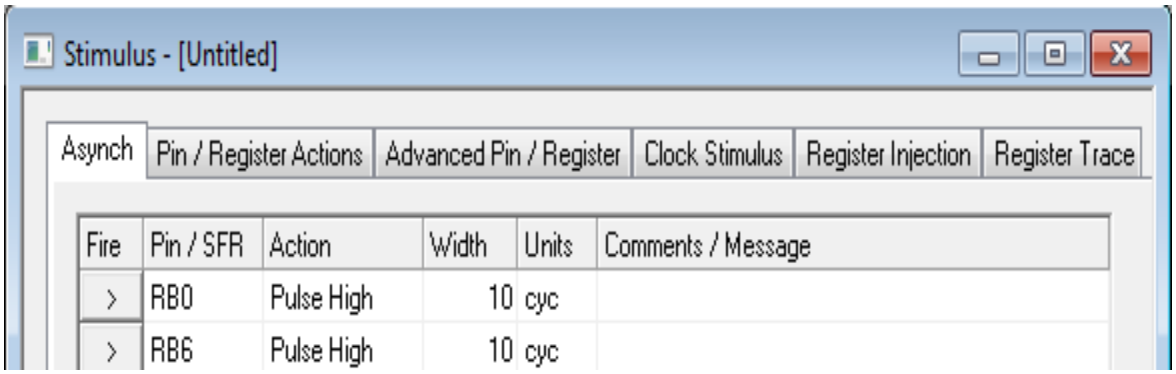


Рис. 5.6. Окно установки стимулов для прерываний по RB0 и RB6

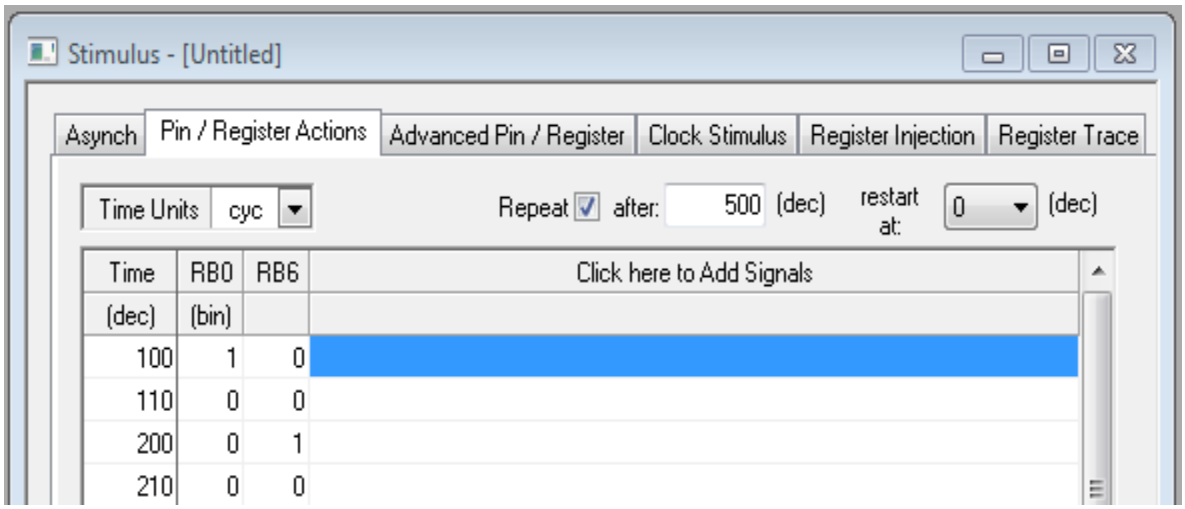


Рис. 5.7. Вкладка установки действий на выводах регистров

Установленные стимулы можно сохранить в виде файла в рабочей книге *Workbook* и использовать этот файл в дальнейшей работе.

Более сложные воздействия на программу можно задавать, используя остальные вкладки окна *Stimulus*.

10. В первом операторе программы установить: `count=5;`.

Выполнить компиляцию и загрузить исполняемый файл в модель макета.

Выполнить *Analysis-Transient* и измерить период мигания для значения `count=5` (рис. 5.8). Убедиться в совпадении результатов с учетом значения тактовой частоты макета.

Домашнее задание

Оформите отчет по работе. Отчет должен содержать листинг программы, скриншоты моделирования в TINA, обсуждение результатов и выводы.

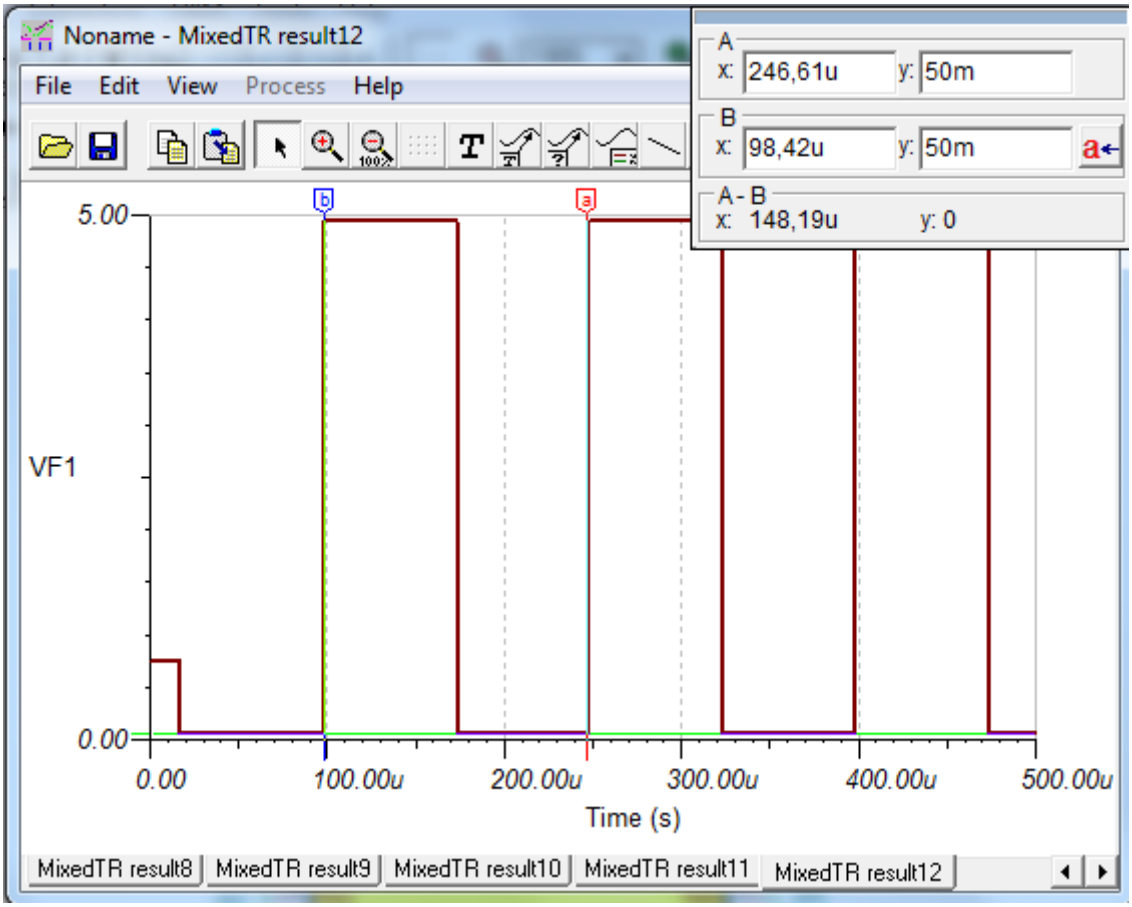


Рис. 5.8. Измерение периода мигания в макете

Контрольные вопросы

1. Как надо выбрать языковые инструменты для использования компилятора MPLAB XC8 ?
2. Какие директивы надо включить в программу на языке Си для использования компилятора MPLAB XC8 ?

3. Какие команды в программе (листинг 5.1) записаны из библиотеки компилятора MPLAB XC8 ?

4. Какую функцию применяют для обслуживания прерываний в компиляторе MPLAB XC8 ?

5. Какие функции доспута к EEPROM есть в компиляторе MPLAB XC8 ?

6. Какие директивы используют, чтобы встроить в программу с кодом Си инструкции на ассемблере ?

7. Что такое стимулы и как их применияют в MPLAB IDE ?

8. Как установить асинхронные стимулы и выбрать вид импульсного воздействия?

9. Как устанавливают стимулы на заданных циклах выполнения программы ?

5.6. Лабораторная работа №10

Программирование записи и чтения в EEPROM на языке Си

Цель работы: изучение компиляции и отладки программ на языке Си для чтения и записи в EEPROM.

Лабораторное задание

1. Собрать схему модели РПЗУ (рис. 5.9) из лабораторной работы №8. Сохранить схему в папке TI-LAB9. Напомним, как работает схема. На клавиатуре надо набирать последовательность заданных для бригады трех чисел и вводить эти числа в память ОЗУ по сигналу прерываний на входе RB0/INT.

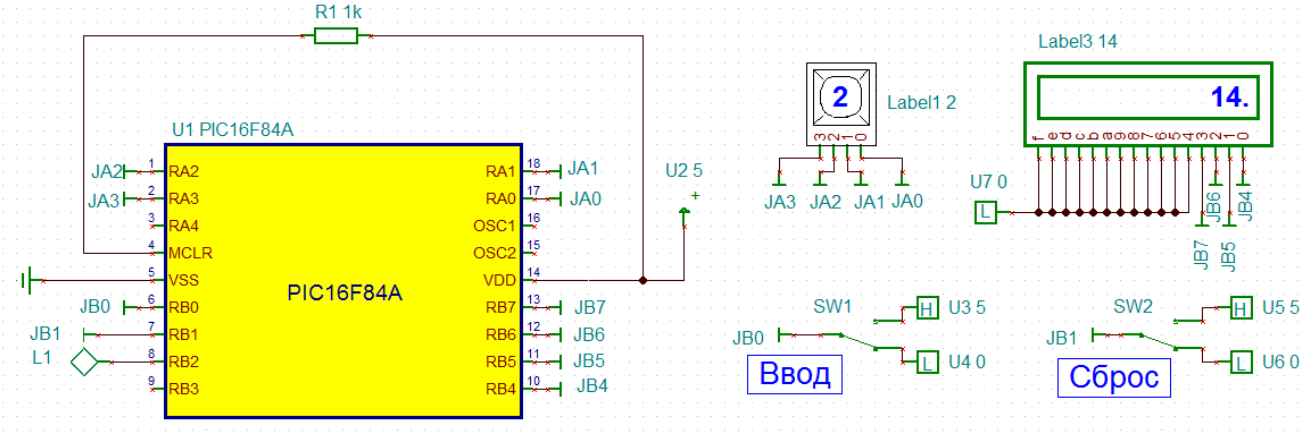


Рис. 5.9. Схема модели РПЗУ

Окончание записи в ОЗУ должно регистрироваться зажиганием индикатора L1. После окончания записи в ОЗУ требуется записать числа в ПЗУ, последовательно считывать и выводить на дисплей с задержкой для удобства чтения. Ключ SW2 «Сброс» позволяет повторить процесс ввода новых чисел и их отображение на дисплее.

2. Возможный вариант текста программы показан в листинге 5.2.

Листинг 5.2

```
// Программа записи и чтения в EEPROM

#include <PIC.h>
#include <xc.h>
#pragma config WDTE=OFF, FOSC=XT, CP=OFF

unsigned char count, count1, n, TEMPA,  address,
value ;   // объявление переменных
unsigned char numb[3];    // массив трех целых чисел

void main(void) //главная функция

{
// Инициализации портов и регистра INTCON
        TRISA=0x1F; //Порт А - все выводы на вход
        TRISB=0x03; /* Порт В - RB0,RB1-входы,
                       ост. выходы*/
        INTCONbits.INTE=1; /*разрешены прерывания по
                             входу INT*/
        INTCONbits.GIE=1; /*глобальное разрешение
                            Прерываний*/
        PORTB=0;          //очистить порт В

        while(1)        //бесконечный цикл
      {
         RB2=0;  //погасить индикатор
         n=0;  //обнулить число циклов записи в ОЗУ
         while(n<3)  //цикл записи чисел в массив
         { continue;
           }
```

```
        RB2^=1;  /* инвертируем вывод и включаем
                    индикатор */
        INTCONbits.INTE=0;  /* запрет прерываний
                            по входу INT*/
        address=0x21; /*установка начального адреса
                        макроса EEPROM_WRITE */
/* Подпрограмма записи в EEPROM*/
         n=0;
         while (n<3)  // цикл записи в EEPROM

        {extern unsigned char address, value, n
          ,TEMPA; //объявление внешних переменных
           extern unsigned char numb[3]; /*объявление
           внешнего массива*/

          value=numb[n]; /*присвоить переменной
                       макроса значение из массива */
          EEPROM_WRITE (address, value);/*функция
                         макроса записи в EEPROM*/

          while(EECON1bits.WR);/*цикл ожидания
                               окончания записи*/

          INTCONbits.GIE=1;   // разрешить прерывания

          EECON1bits.WREN=0;  // запретить запись

          address++;      //инкремент адреса записи
          n++;        //инкремент числа циклов записи
          }
/*Подпрограмма чтения из EEPROM и отображения
 на дисплее*/
          address=0x21;  /*установка начального
                          адреса чтения из EEPROM*/

          for(RB1=0;1-RB1;) /*цикл чтения
                             по условию RB1=0*/
          {
          value=EEPROM_READ(address); /*чтение
                           данного из EEPROM*/
          TEMPA=value;  /*присвоить данное
```

```
                          переменной TEMPA*/
          TEMPA=TEMPA<<4;  //сдвиг влево на 4 разряда
          PORTB=TEMPA;   //вывести данное на порт B
          _delay(10);   /*задержка на 10 циклов
                        для отображения*/
          address++;    //инкремент адреса
          if(address>0x23)   /*условие повторения
                             цикла чтения*/
          address=address-3; /*возврат начального
                             адреса чтения*/
          }

         {INTCONbits.INTE=1; /*разрешение прерываний
          по входу RBO/INT
          PORTB=0;     //очистить порт B
         }
     }
}

   /*Подпрограмма  обслуживания прерываний */
    void interrupt myIsr(void)
        { extern unsigned char n; /*объявление
                                  внешней переменной*/
         TEMPA=PORTA; //чтение порта A
         numb[n]=TEMPA; //запись в массив данных
         n++;           /*инкремент числа циклов
                        записи в ОЗУ*/
         INTCONbits.INTF = 0; /*сброс флага
                             прерываний по RB0/INT*/
         INTCONbits.INTE=1; /*разрешено прерывание
                           по входу RBO/INT*/
          }
```

В программе для записи в EEPROM и чтения из EEPROM использованы макросы `EEPROM_WRITE (address, value)` и `value=EEPROM_READ(address)`.

3. Записать программу из листинга 5.2 в текстовом редакторе Notepad++ и сохранить в папке MP-LAB10 с расширением «.c» как файл LAB10.c.

4. Создать в MPLAB IDE проект LAB10 для микроконтроллера PIC16F84A, выбрать в языковых

инструментах Microchip XC8 Toolsuite. Добавить в проект файл LAB10.c. и скопировать этот файл в проект, установив «С» в окне выбранного файла.

5. В созданном проекте выбрать *Debbuger-Select Tool-MALAB SIM*. Открыть файл программы и выполнить компиляцию, нажав кнопку  .

6. Если компилятор обнаружит ошибки, сделать исправления и добиться успешного результата.

7. Используя стимулы MPLAB IDE, проверить выполнение прерываний по сигналу «Ввод». При отладке записывать в массив число, равное номеру бригады. Просмотреть и зарегистрировать содержание ОЗУ после записи трех чисел.

8. Проверить выполнение записи в EEPROM и чтения из EEPROM. Зарегистрировать содержание EEPROM после записи.

9. Проверить выполнение циклов чтения из EEPROM, отображения данных и выполнения сброса.

10. Загрузить исполняемый HEX-файл и файл LST в модель РПЗУ (рис.5.9) и проверить функционирование. Сделать скриншоты модели.

11. Провести отладку второго варианта программ записи и чтения EEPROM.

Если макросы `EEPROM_WRITE (address, value)` и `value=EEPROM_READ(address)` не поддерживаются для Вашего микроконтроллера, в описании компилятора MPLAB XC8 C есть другие библиотечные функции: `eeprom_write()` и `eeprom_read ()`. В листинге 5.3 приведен текст программы записи данных.

Листинг 5.3

```
/* Программа записи в EEPROM . Вариант 2*/
        n=0;
        while (n<3)

   {extern unsigned char address, value, n ,TEMPA;
    extern unsigned char numb[3];
          void eeprom_write(unsigned char address,
unsigned char value);
```

```
        value=numb[n];
        EEADR = address;  // загрузка адреса

        EEDATA = value;   // загрузка данных

        EECON1bits.WREN=1;
        INTCONbits.GIE=0;
        EECON1bits.WR=1;
        INTCONbits.GIE=0;  // запретить прерывания
        EECON2 = 0x55;    // код разрешения записи
        EECON2 = 0xAA;
        EECON1bits.WR=1;

        while(EECON1bits.WR);

        INTCONbits.GIE=1; // разрешить прерывания

        EECON1bits.WREN=0;  // запретить запись

        address++;

        n++;
    }
```

В листинге 5.4 приведен текст программы чтения данных из EEPROM.

Листинг 5.4.

```
/* Программа чтения из EEPROM. Вариант 2*/
    address=0x21;
    extern unsigned char address, value;
    unsigned char eeprom_read(unsigned char address);

        for(RB1=0;1-RB1;)
       {
        value=eeprom_read(address);
        TEMPA=value;
        TEMPA=TEMPA<<4;
        PORTB=TEMPA;
        _delay(10);
        address++;
        if(address>0x23)
```

```
address=address-3;
}

{INTCONbits.INTE=1;
PORTB=0;
}
```

Заменить фрагменты программы из листинга 5.2 на фрагменты второго варианта. Выполнить компиляцию, отладку программы и проверить функционирование на модели.

Домашнее задание

Оформите отчет по работе. Отчет должен содержать полный листинг программы, скриншоты моделирования в TINA, обсуждение результатов и выводы.

Контрольные вопросы

1. Какие переменные следует объявить в программе для записи и чтения EEPROM ?
2. Объясните сущность команд инициализации портов и регистра INTCON.
3. Объясните, как выполняется подпрограмма записи в EEPROM.
4. Объясните, как выполняется подпрограмма чтения EEPROM.
5. Объясните работу подпрограммы обслуживания прерываний.
6. Как выполняется сброс чтения и повторение записи EEPROM ?
7. Какое наибольшее число можно записать в модели РПЗУ и чем это обусловлено ?

5.7. Лабораторная работа №11

Программирование подключения жидкокристаллического дисплея

Жидкокристаллические дисплеи (ЖК – дисплеи или LCD -Liquid Crystal Display) широко используются в различных устройства для отображения алфавитно-цифровой и графической

информации. Наиболее часто используемые символьные ЖК – дисплеи созданы на основе контроллера HD44780 компании Hitachi или других, которые совместимы с контроллером HD44780. ЖК – дисплеи имеют размеры 16x1, 16x2 или 16x4 (16 столбцов и 1,2 или 4 строки), поддерживают, как правило, 80 символов и имеют 1 контроллер. ЖК –дисплеи, поддерживающие более 80 символов используют два контроллера HD44780. Однако, почти все ЖК – дисплеи соответствуют стандартной спецификации интерфейса. В библиотеке программы TINA имеется дисплей Displaytech 161A с размером 16x1 (16 символов). Каждый символ отображается с помощью 5 × 8 или 5 × 10 точек матрицы. Схема выводов этого дисплея показана на рис.5.10.

В этой работе мы рассмотрим, как программировать вывод данных на дисплей, используя MPLAB IDE и компилятор MPLAB XC8.

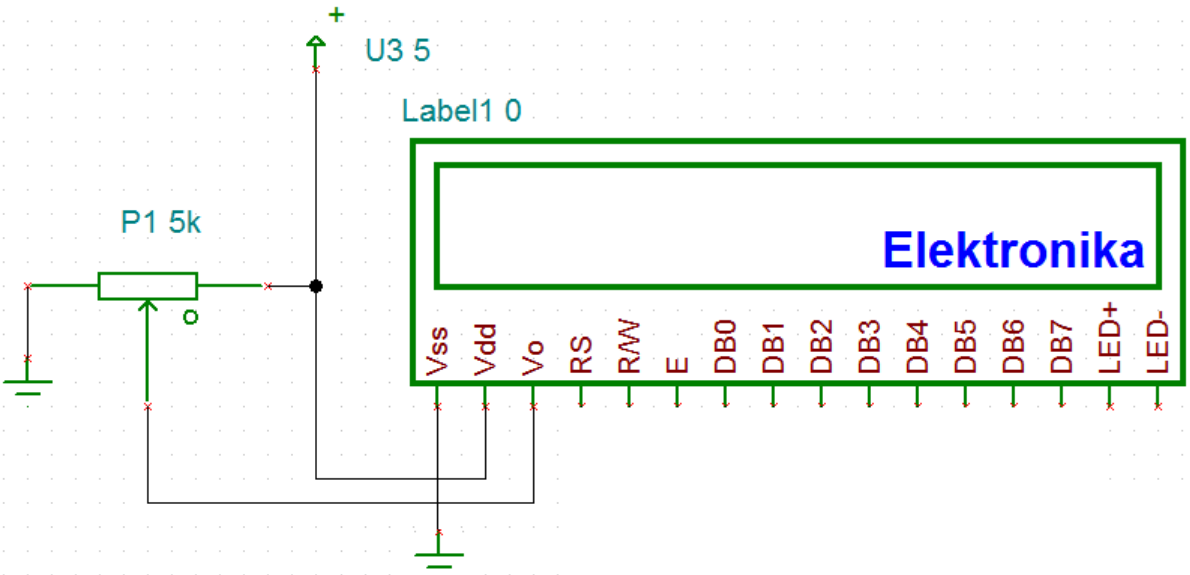


Рис. 5.10. Схема выводов LCD дисплея

Микроконтроллер должен отправлять на дисплей информацию двух типов: информация о данных и информацию о командах управления. Данные это значения ASCII символов, которые будут отображаться на ЖК-экране (LCD Panel). Командная информация определяет положение символа на экране, сдвиг, положение курсора, очистку экрана и т.д.

Упрощенная структурная схема контроллера HD44780 показана на рис. 5.11. Основные элементы, с которыми приходится взаимодействовать при программном управлении: регистр данных (DR), регистр команд (IR), видеопамять (DDRAM), ОЗУ знакогенератора (CGRAM), счетчик адреса памяти (AC), флаг занятости контроллера. Другие элементы (знакогенератор, формирователь курсора, сдвиговые регистры и драйверы и т.д.) участвуют в процессе регенерации изображения на ЖК-дисплее, но с программой непосредственно не взаимодействуют.

Стандартный контроллер HD44780 требует 3 линии контроля, а также 4 или 8 линий ввода/вывода для шины данных. Пользователь может выбрать, будет ли ЖК-дисплей использован для работы с 4-битной шиной данных или с шиной данных 8 бит. При использовании шины данных 4 бита ЖК-дисплей потребует в общей сложности 7 линий передачи данных (3 линии управления плюс 4 линий шины данных). Если используется шина данных 8-бит, ЖК-дисплей потребует в общей сложности 11 линий данных (3 линии управления плюс 8 линий шины передачи данных).

Три линии управления называются EN, RS, и RW.

EN линия называется *Enable* (Включить). Это контрольная линия используется, чтобы сообщить на ЖК-дисплей, что мы отправляем текстовые данные для отображения. Чтобы отправить текстовые данные на ЖК-дисплей, наша программа должна убедиться, что на линии EN низким уровень (0), а затем установить другие две линии контроля RS и RW и поместить данные на шину данных. Когда другие линии полностью готовы, надо установить на EN высокий уровень (1), подождать в течение минимального количества времени, необходимого по техническому описанию ЖК-дисплея для записи в ЖК-дисплей, и в конце, снова перевести EN в (0).

Линия RS называется является линия *Registr Select*. Когда на RS низкий уровень (0), данные следует рассматривать как команду или специальную инструкцию (например, очистить

экран, позиция курсора и т.д.). Когда на RS высокий уровень (1), то значит отправляются текстовые данные, которые должны быть отображены на экране.

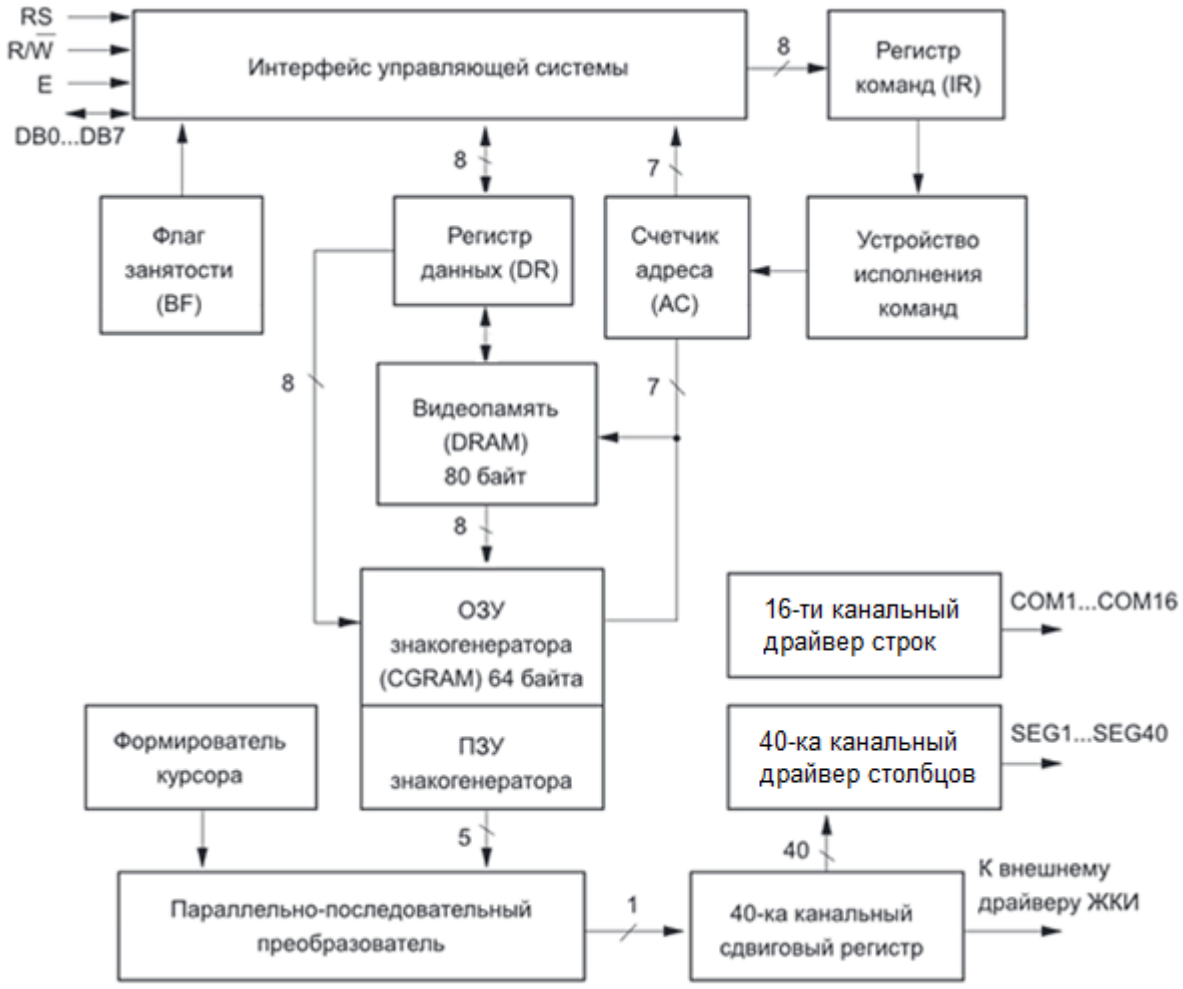


Рис. 5.11. Структурная схема контроллера HD44780

R/W линия *Read/Write* - это контрольная линия. Когда уровень R/W низкий (0), информация с шины данных записывается на ЖК-дисплее. Когда уровень R/W высокий (1), программа отвечает на запросы чтения ЖК-дисплея. Только одна команда *Get LCD status* является командой чтения. Все остальные записи команды - это команды записи. Поэтому R/W почти всегда будет иметь низкий уровень.

Шина данных состоит из 4 или 8 линий (в зависимости от режима работы, выбранного пользователем). В случае шины

данных 8-битной строки выводы называются DB0, DB1, DB2, DB3, DB4, DB5, DB6, и DB7. В 8-битном режиме 8 бит данных и команд надо отправить через линии передачи данных DB0 - DB7 и использовать строб данных через E - ввод ЖК-монитора. Но 4-битный режим использует только 4 линии передачи данных. В этом случае 8-битовые данные и команды разбиты на 2 части (по 4 бита в каждой). Они отправляются последовательно через линии передачи данных DB4 - DB7 с собственным стробом данных на входе E. Целью 4-битного режима является сокращение используемых контактов микроконтроллера. Причем разница в скорости минимальна, так как ЖК-дисплеи медленные устройства, небольшая разница в скорости передачи данных между этими режимами не является существенной. Благодаря инерции зрительного восприятия, мы даже не почувствуем разницу в скорости.

Выводы питания для подсветки обозначены LED + и LED -. Некоторые ЖК-модули поставляются без подсветки. В этом случае, эти контакты отсутствуют или отключены. Рекомендуемое напряжение для LED + составляет 4,2В. LED- должен быть подключен к земле (GND). Изменение значения резистора, подключенного к LED + будет изменять яркость подсветки. Сделать это можно, подключив резистор в линию LED+. Как правило, используют резистор 220 Ом или 330 Ом.

Операции записи для 4-х разрядной шины

Рассмотрим последовательности действий, которые необходимо выполнять управляющей программе при совершении операций записи и чтения для 4-х разрядной шины (таблицы 5.5 и 5.6)

Таблица 5.5

Операция записи данных для 4-х разрядной шины

1	Установить значение линии RS
2	Вывести значение старшей тетрады байта данных на линии шины DB4...DB7
3	Установить линию E = 1

4	Установить линию E = 0
5	Вывести значение младшей тетрады байта данных на линии шины DB4...DB7
6	Установить линию E = 1
7	Установить линию E = 0
8	Установить линии шины DB4...DB7 = HI (высокий импеданс)

Таблица 5.6

Операция чтения для 4-х разрядной шины

1	Установить значение линии RS
2	Установить линию R/W = 1
3	Установить линию E = 1
4	Считать значение старшей тетрады байта данных с линий шины DB4...DB7
5	Установить линию E =0
6	Установить линию E = 1
7	Считать значение младшей тетрады байта данных с линий шины DB4...DB7
8	Установить линию E = 0
9	Установить линию R/W = 0

При выполнении перечисленных команд надо соблюдать временные задержки, указанные в документации ЖК-дисплея. Эти задержки закладываются в программу управления.

Управление контроллером выполняется через интерфейс управляющей системы. Основными объектами взаимодействия являются регистр данных DR и регистр команд IR. Выбор адресуемого регистра производится линией RS: если RS = 0 - адресуется регистр команд (IR), если RS = 1 - регистр данных (DR).

Данные через регистр DR, в зависимости от текущего режима, могут помещаться (или считываться) в видеопамять (DDRAM) (Display Data RAM) или в ОЗУ знакогенератора (CGRAM) (Character Generator RAM) по текущему адресу, указываемому счетчиком адреса (AC). Информация, попадающая

в регистр IR, интерпретируется устройством выполнения команд как управляющая последовательность. Прочтение регистра IR возвращает в 7-ми младших разрядах текущее значение счетчика AC, а в старшем разряде флаг занятости (BF).

Видеопамять, имеющая общий объем 80 байтов, предназначена для хранения кодов символов, отображаемых на ЖК-дисплее. Видеопамять организована в две строки по 40 символов в каждой. Независимо от того, сколько реальных строк будет иметь каждый конкретный ЖК-дисплей, адресация видеопамяти всегда производится как к двум строкам по 40 символов.

ЖК-дисплей является устройством с динамической индикацией. Контроллер производит периодическое обновление информации на ЖК-панели. Каждый символ в простейшем случае состоит из 5x8 точек. Будем различать строку точек и строку символов. В текстовой строке может быть 8, 16 или 40 символов, что требует соответственно 40, 80 или 200 столбцов (сегментов) в строке точек.

Формирование изображения производится по строкам точек. Для этого ЖК-панель имеет 8 параллельных общих электродов строк (подложек), которые последовательно коммутируются драйвером строк. Активными являются точки тех сегментов, которые находятся над включенной строкой. Схема включения ЖК-панели с форматом 1 строка на 16 символов показана на рис. 5.12.

Форматы с большим количеством символов реализуются установкой дополнительных микросхем расширения по столбцам и применением двух контроллеров HD44780.

Код символа в формате ASCII в знакогенераторе CGROM (Character Generator ROM) преобразуется в шаблон символа, содержащий 5x7 точек. Включенные сегменты посторочно выводятся на ЖК-панель и формируют динамическое изображение. Частота регенерации изображения более 25 Гц и мелькания незаметны. Фрагмент карты знакогенератора контроллера HD44780 показан на рис. 5.13.

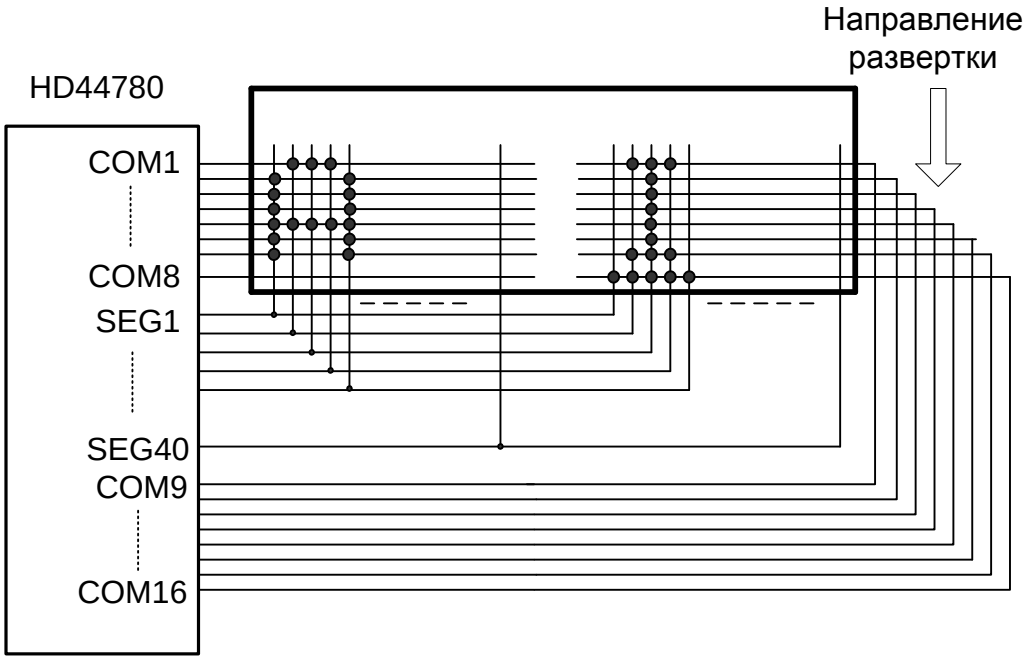


Рис. 5.12. Схема управления ЖК-панели с форматом 1 строка на 16 символов

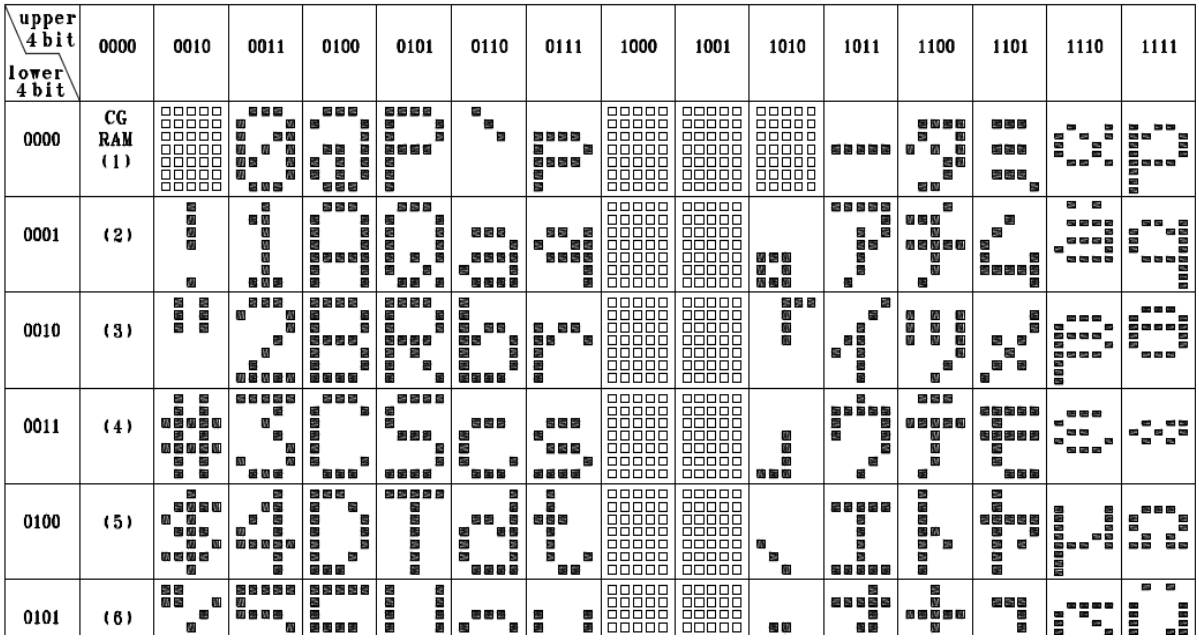


Рис. 5.13. Фрагмент карты знакогенератора

Система команд контроллера HD44780

Контроллер дисплея поддерживает 11 команд, в которые входят и команды инициализации дисплея.

Тип команды определяется сигналами на линиях RS и RW.

RS	RW	Тип команды
0	0	Записать команду. Все команды от *Clear Display* до *Set DDRAM Address* включительно.
0	1	Прочитать параметры. Единственные параметры которые можно прочитать из дисплея это *busy flag* и *address counter***.**
1	0	Записать данные в DDRAM или CGRAM память.
1	1	Прочитать данные из DDRAM или CGRAM память.

Команда *Clear Display* – очистить дисплей

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

После отправки этой команды дисплей начинает записывать во все ячейки DDRAM памяти пустой символ “Space”, а в адресный счетчик (address counter) записывает 0×00. Эта команда выполняется 2~5мс.

Команда *Return Home* - возврат каретки.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	x

Записывает в `address counter` значение 0×00, без изменения DDRAM памяти.

Команда *Entry Mode Set* – режим ввода

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	SH

Определяет, как будут отображаться введенные символы: слева направо или справа налево:

I/D=1-слева направо (инкремент); I/D=0 – справа налево (декремент); SH=1 – активирован сдвиг в выбранном направлении при заполнении строки; SH=0 – дезактивирован сдвиг.

Команда *Display On/Off CONTROL*-управление дисплеем

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

Управление дисплеем: D=1 – включить дисплей; D=0 – отключить дисплей; C=1 – включить курсор (символ подчеркивания); C=0 – отключить курсор; B=1 – включить курсор (черный квадрат); B=0- отключить курсор.

Команда *Cursor or Display Shift* –сдвиг курсора или экрана

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	S/C	R/L	x	x

Можно получить эффект бегущей строки:

S/C=1 – сдвиг дисплея целиком; S/C=0 – сдвиг курсора; R/L=1 – сдвиг вправо; R/L=0 – сдвиг влево.

Команда *Function Set* – настройка параметров дисплея

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	x	x

При помощи этой команды выбирают разрядность шины данных/команд, число рабочих строк в дисплее и размер шрифта:

DL=1 (Data Lenght) – используем 8-битную шину данных/адресов (DB7 - DB0); DL = 0 – используем 4-битную шину данных/адресов (DB7 - DB4), остальные линии надо заземлить; N = 1 – используем обе строки; N = 0 – работаем только с верхней строкой; F = 1 – шрифт размером 5×7 пикселей; F = 0 – шрифт размером 5×10 пикселей.

При использовании обеих строк, шрифт автоматически устанавливается на 5×7 пикселей, независимо от F-бита.

Команда *Set CGRAM Address* – установка адреса CGRAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

Указывает адрес CGRAM-ячейки, в которую будем записан байт данных. Данные в ОЗУ знакогенератора посылаются и получаются после этой установки.

Команда *Set DDRAM Address* – установка адреса DDRAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

Указываем адрес DDRAM-ячейки, в которую будет записан символ. Если адрес находится в видимой области DDRAM памяти, то символ тут же будет отображен. Если используется только верхняя строка, то для нее резервируется DDRAM память от 0×00 до 0x4F. Если используют обе строки, то для каждой резервируется DDRAM память от 0×00 до 0×27 для верхней строки и от 0×40 до 0×67 для нижней, всего по 40 байт строку.

Команда *Read Busy Flag & Address* – считать флаг занятости и адрес

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0

Для того чтобы выполнить эту команду надо установить вывод R/W = 1, тем самым сообщая дисплею, что идет команда чтения. Считываем флаг занятости (BF). Если BF = 1, значит дисплей еще не закончил выполнение предыдущей инструкции и любая другая инструкция, посланная в этот момент, не будет выполнена. Остальные 7 бит, представляют собой содержимое счетчика адреса.

Команда *Write data to RAM* – запись данных в ОЗУ

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D7	D6	D5	D4	D3	D2	D1	D0

Запись данных в CGRAM или DDRAM память. Прежде чем начать запись данных, следует указать в какую именно память

будет вестись запись. Это выполняется при помощи команды *Set DDRAM Address* или *Set CGRAM Address*. Сперва указывают стартовый адрес любой из двух видов память, а дальше посылают символы. При этом счетчик адреса автоматически инкрементируется или декрементируется (переходит на следующий адрес), в зависимости от настроек режима ввода.

Команда *Read data from RAM* – чтение данных из ОЗУ

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D7	D6	D5	D4	D3	D2	D1	D0

Чтение данных из CGRAM или DDRAM памяти. Прежде чем выполнить эту команду, указывают память (начальный адрес), из которой будет проходить чтение. Счетчик адреса автоматически инкрементируется или декрементируется в зависимости от настроек режима ввода.

Важное примечание: время выполнения команд достигает 2~5 мс, поэтому в программу управления дисплеем надо вводить временные задержки.

Лабораторное задание

1. Собрать схему модели микроконтроллера с ЖК-дисплеем (рис. 5.14).

2. Возможный вариант текста программы показан в листинге 5.5. Программа составлена для отладки с компилятором MPLAB XC8, в котором нет библиотечных функций для ЖК-дисплеев. Поэтому функции инициализации дисплея, очистки, посылки команд, вывода текста и т.п. приходится программировать самостоятельно. Библиотечные функции для работы со строками включены директивой `#include <string.h>`.

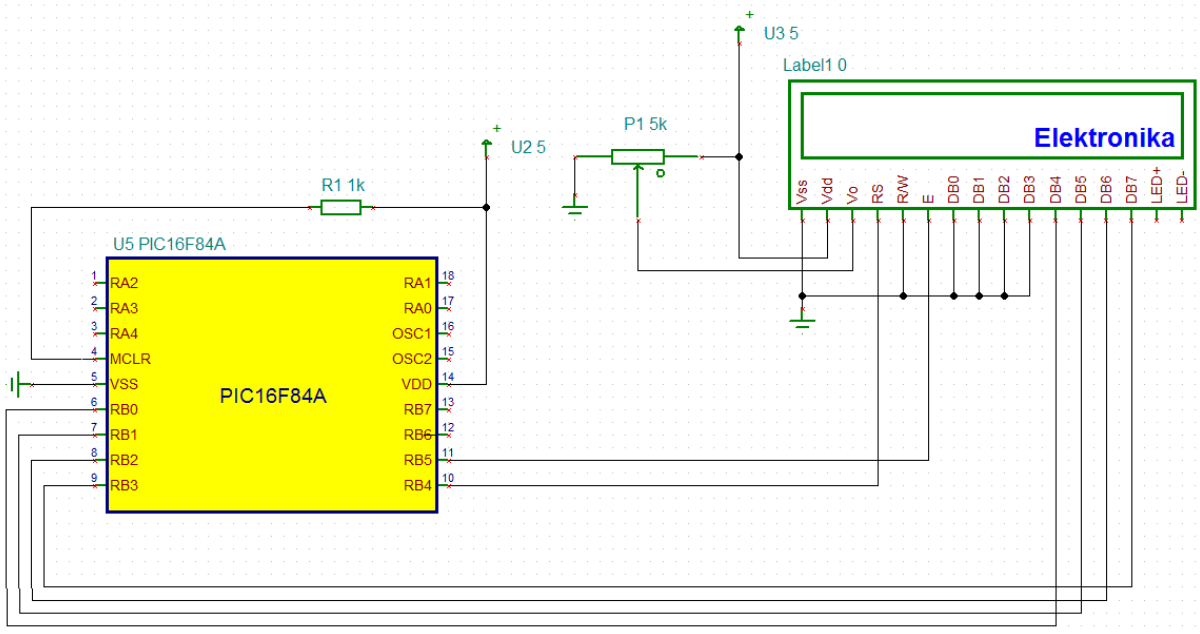


Рис. 5.14. Схема микроконтроллера с ЖК-дисплеем

Листинг 5.5

```
#include <xc.h>          /* XC8 General Include File
*/
#include <string.h>          /* XC8 General Include
File */
#define _XTAL_FREQ 8000000
#define DELAY2 10
typedef unsigned char BYTE;
char LCDString[48];
void my_delay(int u)
{
 int i;
 for (i=0; i<u; i++) ;
}
// LCD
void init()
{
 TRISB = 0x00;
}
void send_byte(BYTE c)
{
 PORTB = c;
}
```

```
void lcd_send(BYTE val, BYTE is_data)
{
BYTE tmp;
 tmp = val >> 4;
 if (is_data) tmp |= 0x10;
 send_byte( tmp ); /* старший полубайт, RS=1 когда
посылаются данные, RS=0 когда посылается команда, E=0
*/
 tmp |= 0x20; send_byte( tmp );    // E=1
 tmp &= 0xDF; send_byte( tmp );    // E=0
}
void lcd_send2(BYTE val, BYTE is_data)
{
 BYTE tmp;
tmp = val >> 4; if (is_data) tmp |= 0x10; send_byte(
tmp ); /* старший полубайт, RS=1 когда посылаются
данные, RS=0 когда посылается команда, E=0  */
 tmp |= 0x20; send_byte( tmp );    // E=1
 tmp &= 0xDF; send_byte( tmp );    // E=0
 tmp = val & 0x0F; if (is_data) tmp |= 0x10;
send_byte( tmp ); /* младший полубайт, RS=1 когда
посылаются данные, RS=0 когда посылается команда, E=0
*/
 tmp |= 0x20; send_byte( tmp );    // E=1
 tmp &= 0xDF; send_byte( tmp );    // E=0
 //my_delay(2);
}
void lcd_clear()
{
 lcd_send2(0x01,0);
 lcd_send2(0x02,0);
}
void write_string(char* u)
{
 int i, l;
 l = strlen(u);
 for (i=0; i<l; i++)
  lcd_send2(u[i],1);
}
void lcd_init()
{
```

```
 lcd_send(0x30,0); my_delay(DELAY2);
 lcd_send(0x30,0);

 lcd_send(0x30,0);
 lcd_send(0x20,0);

 lcd_send(0x20,0);
 lcd_send(0x80,0);

 lcd_send(0x00,0);
 lcd_send(0x80,0);

 lcd_send(0x00,0);
 lcd_send(0x10,0);

 lcd_send(0x00,0);
 lcd_send(0x60,0);

 lcd_send(0x00,0);
 lcd_send(0xE0,0);

 lcd_send(0x80,0);
 lcd_send(0x00,0);
   //write_string(message_boot); my_delay(1000);
}
void main(void)
{
 init();
 lcd_init();
 strcpy(LCDString, "Elektronika");
 write_string(LCDString); my_delay(1000);
 while(1) ;
}
```

3. Записать программу из листинга 5.4 в текстовом редакторе Notepad++ и сохранить в папке MP-LAB11 с расширением «.c» как файл LAB11.c.

4. Создать в MPLAB IDE проект LAB11 для микроконтроллера PIC16F84A, выбрать в языковых инструментах Microchip XC8 Toolsuite. Добавить в проект файл

LAB11.c и скопировать этот файл в проект, установив «С» в окне выбранного файла.

5. В созданном проекте выбрать *Debbuger-Select Tool-MALAB SIM*. Открыть файл программы и выполнить компиляцию, нажав кнопку  .

6. Если компилятор обнаружит ошибки, сделать исправления и добиться успешного результата.

7. После успешной компиляции загрузить HEX- файл в модель микроконтроллера, выполнить моделирование и убедится в правильном отображении текста на дисплее.

8. Вывести текст с указанием номера бригады. Например: «MY_GROUP_7». Выполнить компиляцию и моделирование. Зарегистрировать полученные результаты.

Домашнее задание

Оформите отчет по работе. Отчет должен содержать программы вывода данных на дисплей, скриншоты моделирования в TINA, обсуждение результатов и выводы.

Контрольные вопросы

1. Какие символьные жидкокристаллические дисплеи применяют с микроконтроллерами ?
2. Объясните назначение выводов LCD дисплея.
3. Какую структурную схему имеет контроллер дисплея ?
4. Объясните назначение основных элементов контроллера дисплея.
5. Объясните назначение и работу линий управления дисплеем.
6. Какие режимы передачи данных применяют в дисплеях и как устанавливают режим передачи ?
7. Что такое динамическая индикация ?
8. Как формируется изображение при динамической индикации ?
9. Поясните работу знакогенератора символов.

10. Какие типы команд используются в дисплее и чем устанавливается тип команды ?

11. Какая команда выполняет настройку дисплея и как работает эта команда ?

Глава 6. МИКРОКОНТРОЛЛЕР PIC16F877A

Все наиболее важные разработки в электронике связаны с изобретением и применением микроконтроллеров. Микроконтроллер может управлять различными периферийными устройствами: жидкокристаллическими экранами, клавиатурами, устройства измерения и контроля, выполнять запись на внешние карты памяти, обеспечивать связь с другими компьютерами. Но для этого требуются более мощные микроконтроллеры, чем изученный нами PIC16F84A.

Изучение микроконтроллеров мы продолжим на примере PIC16F877A. Эта популярная и высокопроизводительная модель восьмибитного микроконтроллера имеет развитую систему модулей для связей с объектами управления и позволяет выполнять многие важные технические задачи. Система команд этого микроконтроллера такая же, как у изученного нами PIC16F84A.

6.1. Технические характеристики микроконтроллера PIC16F877A

Описание микроконтроллера PIC16F877A приведено в [3].

Характеристики микроконтроллера:

- Высокоскоростная RISC архитектура;
- 35 инструкций;
- Тактовая частота 20МГц;
- Один машинный цикл 200нс;
- FLASH память программ (14-разрядных слов) 8К;
- Память данных (байт) 368;
- EEPROM память данных (байт) 256;
- Прерываний 14;
- Порты ввода/вывода: Порты A, B, C, D, E;
- Таймеры 3;
- Модуль захват/сравнение/ШИМ 2;
- Модули последовательного интерфейса MSSP, USART;

- Модули параллельного интерфейса PSP;
- Модуль 10-разрядного АЦП 8 каналов.

6.2. Особенности архитектуры

Архитектура микроконтроллера PIC16F877 показана на рис. 6.1.

Основные отличия микроконтроллера PIC16F877 от изученного нами PIC16F84A состоят в следующем.

PIC16F877 имеет дополнительно:

- пять портов ввода/вывода (PORTA, PORTB, PORTC, PORTD, PORTE);
- три таймер (TMR0, TMR1, TMR2);
- два модуля захват/сравнение/ШИМ (Capture/Compare/PWM) - CCP1, CCP2);
- десятиразрядный аналогово-цифровой преобразователь (АЦП) на восемь каналов;
- синхронный последовательный порт MSSP;
- последовательный интерфейс USART;
- модуль параллельного интерфейса PSP.

6.3. Организация памяти программ

Микроконтроллеры PIC16F87X имеют 13-разрядный счетчик команд PC, способный адресовать 8Kx14 слов FLASH памяти команд. Адрес вектора сброса – 0000h. Адрес вектора прерываний – 0004h (рис. 6.2).

6.4. Организация памяти данных

Память данных разделена на четыре банка, которые содержат регистры общего и специального назначения. Биты RP1 (STATUS<6>) и RP0 (STATUS<5>) предназначены для управления банками данных. В таблице 6.1 показано состояние управляющих битов при обращении к банкам памяти данных.

В таблице 6.2 представлена карта памяти микроконтроллера PIC16F877.

Объем банков памяти данных до 128 байт (7Fh). В начале банка размещаются регистры специального назначения, затем регистры общего назначения выполненные как статическое ОЗУ.

Все банки содержат регистры специального назначения. Часто используемые регистры специального назначения могут отображаться в других банках памяти.

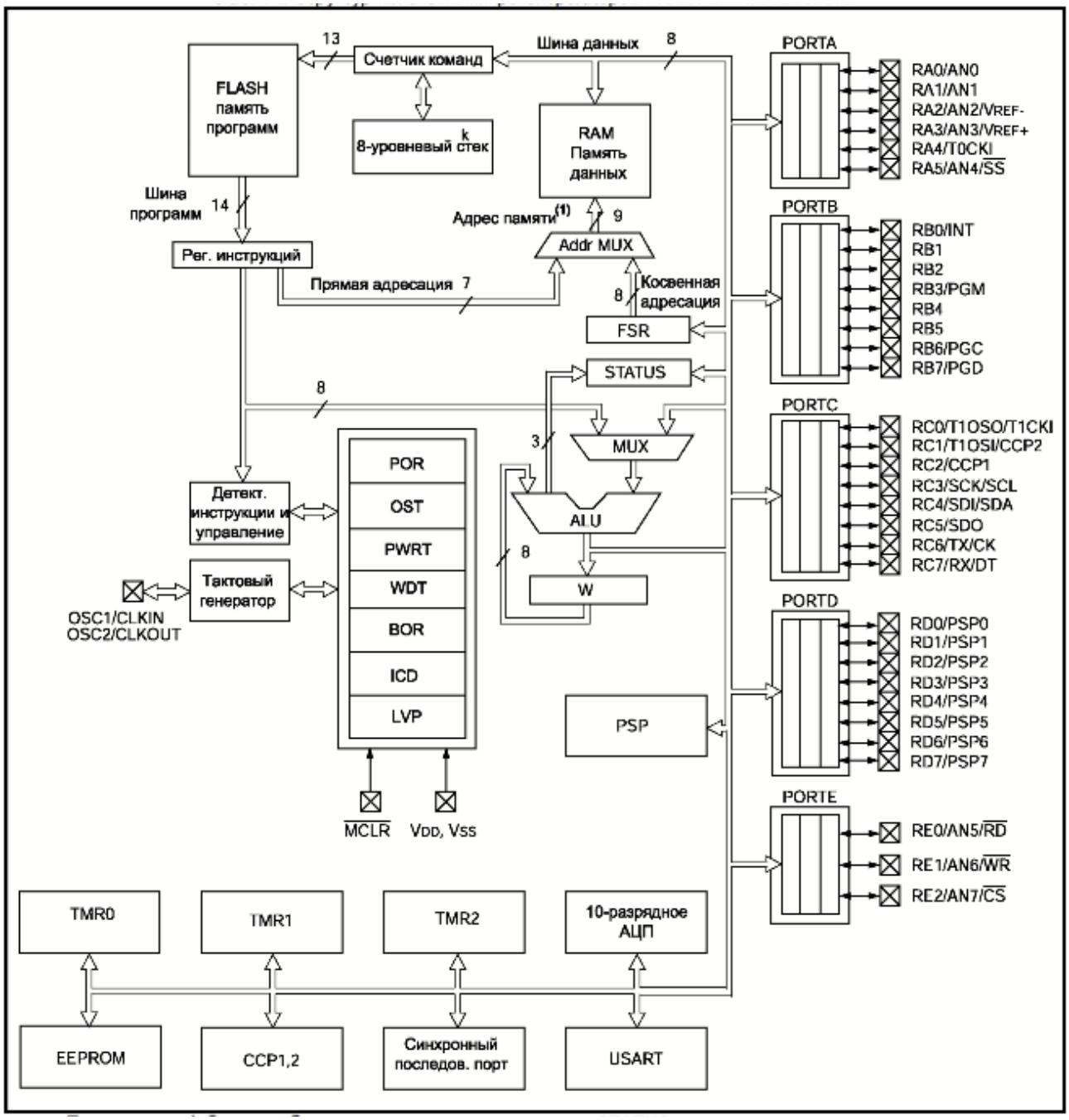


Рис.6.1. Архитектура микроконтроллера PIC16F877

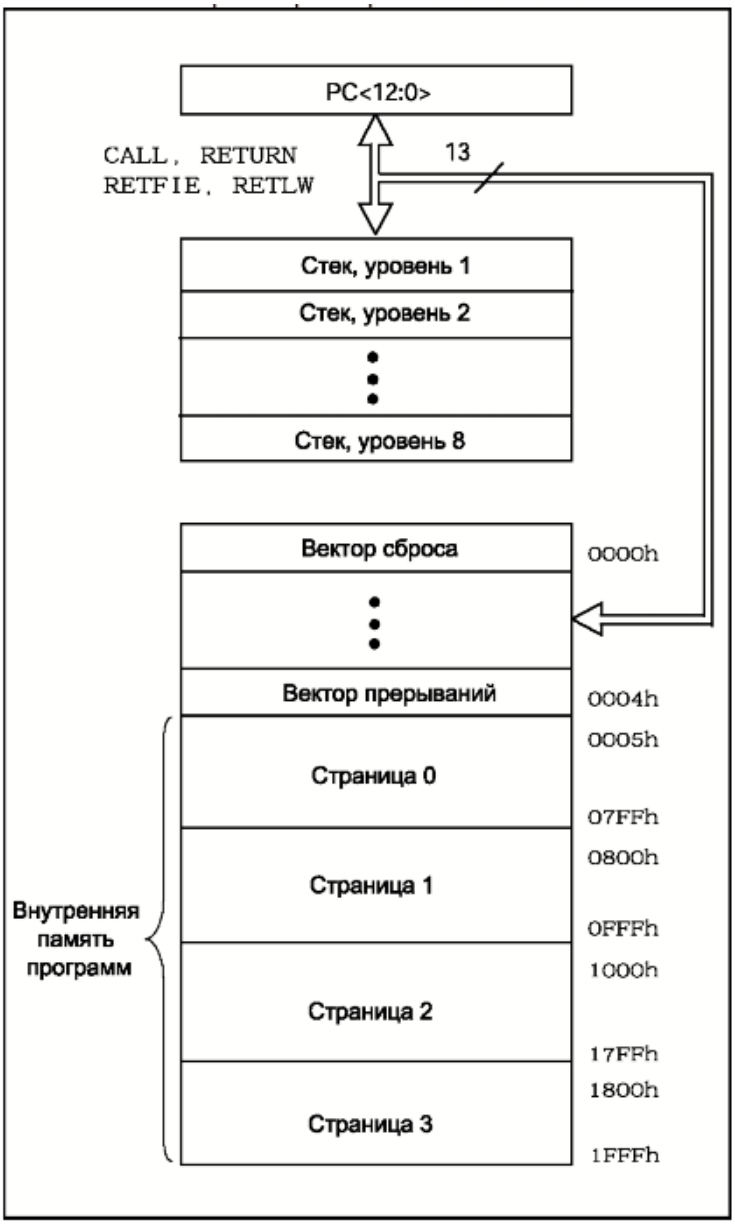


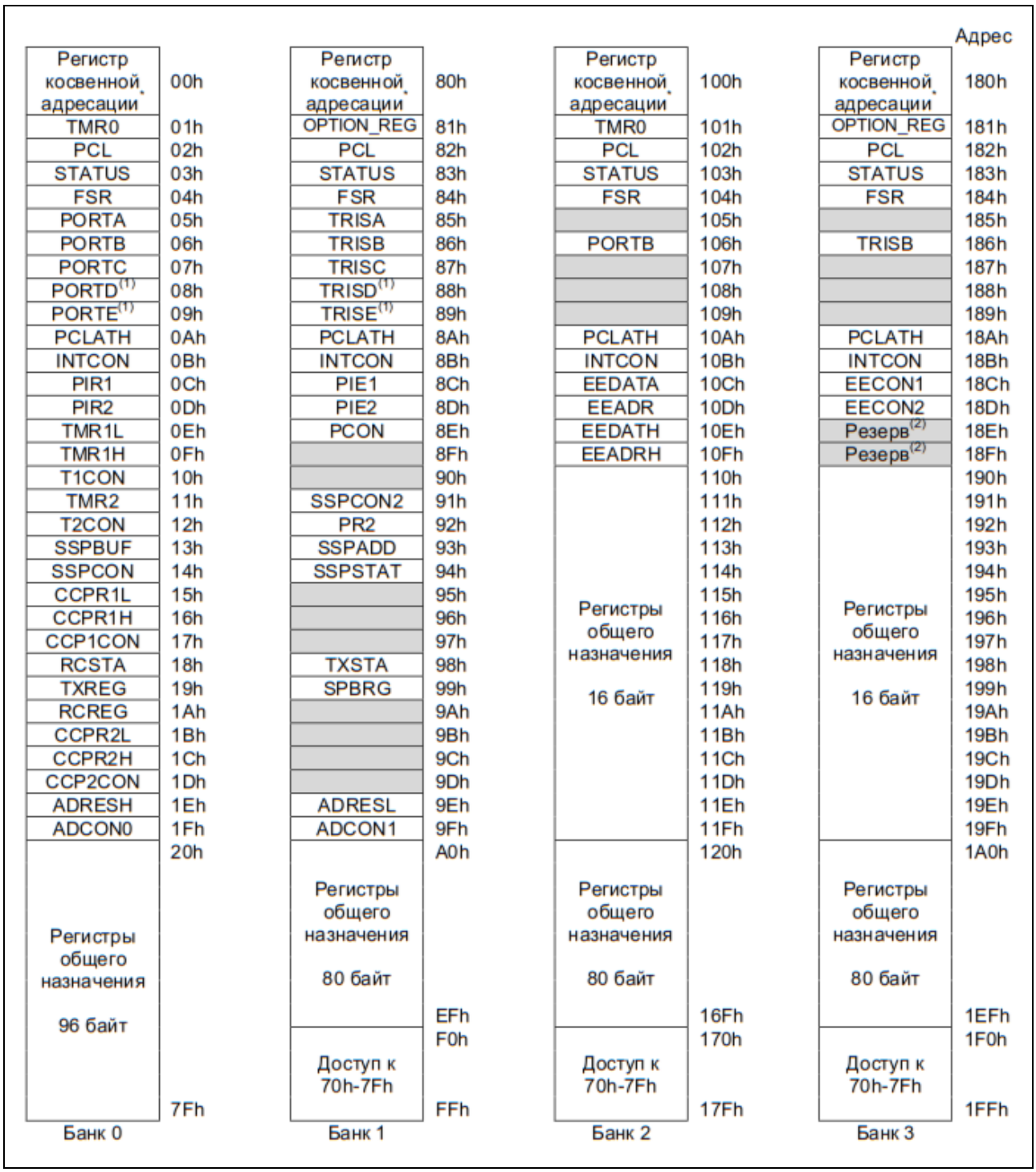
Рис.6.2. Организация памяти программ

Таблица 6.1

RP1: RP0	Банк
00	0
01	1
10	2
11	3

Таблица 6.2

Карта памяти данных микроконтроллера PIC16F877



6.5. Регистры специального назначения

С помощью регистров специального назначения выполняется управление функциями ядра и периферийными модулями микроконтроллера. Регистры специального назначения реализованы как статическое ОЗУ. Сначала мы рассмотрим регистры, управляющие функциями ядра микроконтроллера.

Регистр STATUS Адрес 03h, 83h, 103h или 183h

Наименование и назначение бит регистра STATUS соответствует таблице 1.1 для микроконтроллера PIC16F84A.

Регистр OPTION_REG Адрес 81h или 181h

Наименование и назначение бит регистра OPTION_REG соответствует таблице 1.2 для микроконтроллера PIC16F84A.

Регистр INTCON Адрес 0Bh, 8Bh, 10Bh или 18Bh

Отличие от таблицы 1.4 заключается в наименовании и назначении бита 6:

Бит 6	PEIE	Разрешение прерываний от периферийных модулей 1=разрешены все немаскированные прерывания периферийных модулей 0=прерывания от периферийных модулей запрещены

Регистр PIE1 Адрес 8Ch

Регистр PIE1 доступен для чтения и записи, содержит биты разрешения периферийных прерываний. Чтобы разрешить периферийные прерывания необходимо установить в `1` бит PEIE(INTCON<6>)

Бит 7	PSPIE	Разрешение прерывания записи/чтения ведомого параллельного порта 1= прерывание разрешено 0=прерывание запрещено

Бит 6	ADIE	Разрешено прерывание по окончанию преобразования АЦП 1= прерывание разрешено 0=прерывание запрещено
Бит 5	RCIE	Разрешение прерывания от приемника USART 1= прерывание разрешено 0=прерывание запрещено
Бит 4	TXIE	Разрешение прерывания от передатчика USART 1= прерывание разрешено 0=прерывание запрещено
Бит 3	SSPIE	Разрешение прерывания от модуля синхронного последовательного порта 1= прерывание разрешено 0=прерывание запрещено
Бит 2	CCP1IE	Разрешение прерывания от модуля CCP1 1= прерывание разрешено 0=прерывание запрещено
Бит 1	TMP2IE	Разрешение прерывания по переполнению TMR2 1= прерывание разрешено 0=прерывание запрещено
Бит 0	TMP1IE	Разрешение прерывания по переполнению TMR1 1= прерывание разрешено 0=прерывание запрещено

Регистр PIR1 Адрес 0Ch

Регистр PIR1 доступен для чтения и записи и содержит флаги прерываний периферийных модулей.

Бит 7	PSPIF	Флаг прерывания от ведомого параллельного порта 1= произошла операция чтения или записи

		(сбрасывается программно) 0 = операции чтения или записи не происходило
Бит 6	ADIF	Флаг прерывания от модуля АЦП 1 = преобразование АЦП завершено 0 = преобразование АЦП не завершено
Бит 5	RCIF	Флаг прерывания от приемника USART 1= буфер приемника USART полон 0 = буфер приемника USART пуст
Бит 4	TXIF	Флаг прерывания от передатчика USART 1 = буфер передатчика USART пуст 0 = буфер передатчика USART полон
Бит 3	SSPIF	Флаг прерываний от модуля MSSP 1= выполнено условие возникновения прерывания от модуля MSSP (сбрасывается программно). Условия возникновения прерывания: • SPI - выполнен прием/передача данных . • Ведомый I2C - выполнен прием/передача данных. • Ведущий I2C - выполнен прием/передача данных. - завершено формирование на шине бита START. - завершено формирование на шине бита STOP. - завершено формирование на шине бита повторный START. - завершено формирование на шине бита подтверждения . - обнаружено на шине формирование бита START (для режима с несколькими ведущими). - обнаружено на шине формирование бита

		STOP (для режима с несколькими ведущими). 0 = условие возникновения прерывания от модуля MSSP не выполнено
Бит 2	CCP1IF	Флаг прерывания от модуля CCP 1 Режим захвата 1= выполнен захват значения TMR1 (сбрасывается программно) 0 = захвата значения TMR1 не происходило Режим сравнения 1 = значение TMR1 достигло указанного в регистрах CCPR1H:CCPR1L(сбрасывается программно) 0 = значение TMR1 не достигло указанного в регистрах CCPR1H:CCPR1L ШИМ режим Не используется
Бит 1	TMR2IF	Флаг прерывания по переполнению TMR2 1= произошло переполнение TMR2 (сбрасывается программно) 0 = переполнения TMR2 не было
Бит 0	TMR1IF	Флаг прерывания по переполнению TMR1 1= произошло переполнение TMR1 (сбрасывается программно) 0 = переполнения TMR1 не было

Регистр PIE2 Адрес 8Dh

Регистр PIE2 доступен для чтения и записи, содержит биты разрешений прерываний от модуля CCP2, возникновения коллизий на шине и окончания записи в EEPROM память данных.

Бит 7		Не реализован: читается как `0`
Бит 6	CMIE	Разрешение прерываний компаратора 1=разрешены прерывания компаратора 0=не разрешены прерывания компаратора

Бит 5		Не реализован: читается как `0`
Бит 4	EEIE	Разрешение прерывания записи в EEPROM данных. 1=разрешены прерывания записи в EEPROM данных 0=не разрешены прерывания записи в EEPROM данных
Бит 3	BCLIE	Разрешение прерываний шины коллизий 1= разрешены прерывания по шине коллизии 0=не разрешены прерывания коллизий по шине коллизий
Биты 2-1		Не реализованы: читается как `0`
Бит 0	CCP2IE	Разрешение прерываний CCP2 1= разрешены прерывания CCP2 0=не разрешены прерывания CCP2

Регистр PIR2 Адрес 0Dh

Содержит биты флагов прерываний от модуля CCP2, возникновения коллизий на шине и окончания записи в EEPROM память данных.

Бит 7		Не реализован: читается как `0`
Бит 6	CMIF	Флаг прерываний компаратора 1= вход компаратора изменился (сбрасывается программно) 0=вход компаратора не изменялся
Бит 5		Не реализован: читается как `0`
Бит 4	EEIF	Флаг прерывания по окончанию записи в EEPROM данных. 1=запись в EEPROM данных завершена (сбрасывается программно) 0=запись в EEPROM данных не завершена или не была начата
Бит 3	BCLIF	Флаг возникновения коллизий на шине

		1= на шине обнаружены коллизии (только в режиме ведущего I^2C) 0= коллизий не обнаружено
Биты 2-1		Не реализованы: читается как `0`
Бит 0	CCP2IF	Флаг прерывания от модуля CCP2 Режим захвата 1=выполнен захват значения TMR1 (сбрасывается программно) 0=захвата значения TMR1 не происходило Режим сравнения 1=значение TMR1 достигло указанного в регистрах CCPR2H:CCPR2L(сбрасывается программно) 0=значение TMR1 не достигло указанного в регистрах CCPR2H:CCPR2L ШИМ режим-не используется.

Регистр PCON Адрес 8Eh

Содержит флаги, с помощью которых можно определить источник сброса микроконтроллера.

6.6. Порты ввода/вывода

На рис.6.3 показано расположение и обозначение выводов микроконтроллера PIC16F877A. Микроконтроллер имеет пять портов ввода/вывода (PORTA, PORTB, PORTC, PORTD, PORTE). Некоторые каналы портов мультиплицированы с периферийными модулями микроконтроллера. Такие выводы имеют сложные обозначения (например, RA2/AN2/VRF-). Когда периферийный модуль включен, вывод не может использоваться как универсальный канал ввода/вывода.

Регистры PORTA и TRISA

PORTA – шестиразрядный порт вводы/вывода. Все каналы порта A управляются соответствующими битами направления в регистре TRISA. Запись `1` переводит канал на ввод. Запись `0`

переводит канал на вывод. Канал RA4 имеет триггер Шмидта на входе и открытый сток на выходе.

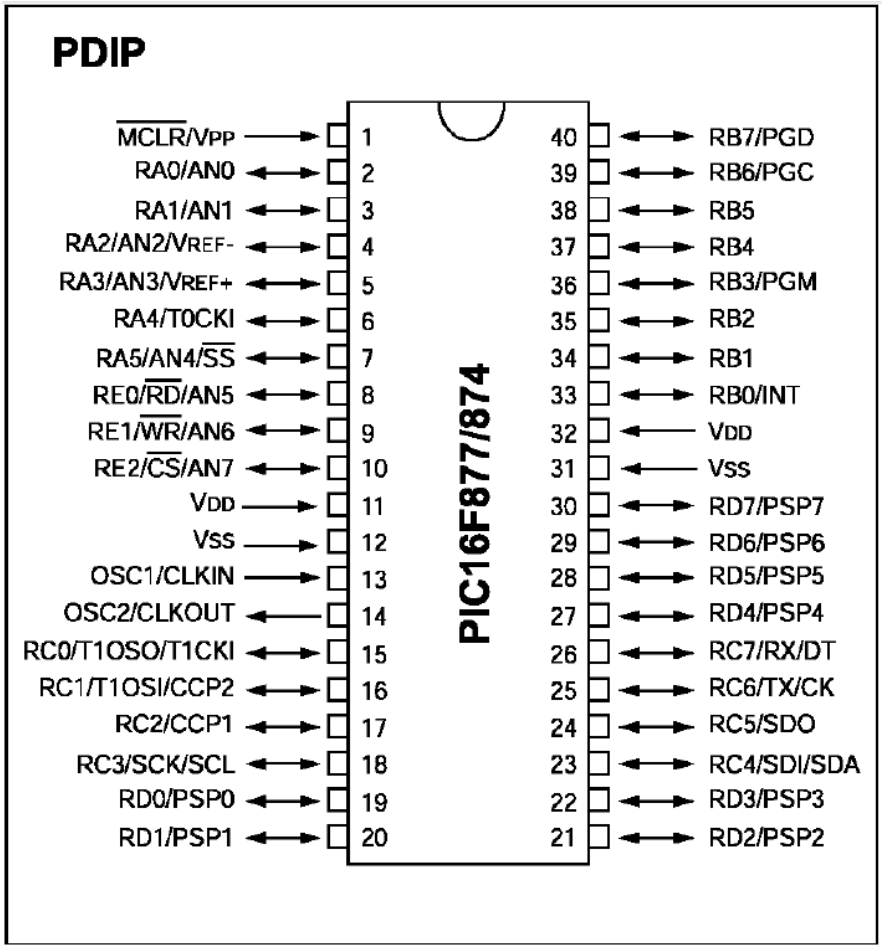


Рис.6.3. Расположение выводов микроконтроллера PIC16F877A

Каналы порта А мультиплицированы с аналоговыми входами АЦП и аналоговым входом источника опорного напряжения V_{REF}. Биты управления режимами работы каналов порта А находятся в регистре ADCON1. После сброса по включению питания выводы настраиваются как аналоговые входы, а чтение дает результат `0`.

Биты регистра TRISA управляют направлением каналов PORTA даже, когда они используются в качестве аналоговых входов.

Для установки выводов PORTA в режим цифровых входов-выходов в регистре ADCON1 управляющие биты PCFG3:PCFG0 должны быть установлены в `011x`.

Пример инициализации PORTA приведен на листинге 6.1.

Листинг 6.1

```
BCF         STATUS,RP1
BCF         STATUS,RP0  ; Выбрать банк 0
CLRF        PORTA   ;Инициализация защелок
                    ; PORTA
BSF         STATUS,RP0  ;Выбрать банк 1
MOVLW       0x06        ;
MOVLF       ADCON1      ; Каналы PORTA –
                        ; цифровые
MOVLW       0xCF        ;Значение для
   ; инициализации направления каналов PORTA
MOVWF       TRISA     ;Настроить RA<3:0> как
                    ; входы,RA<5:4> как выходы
                   ;Биты TRISA<7,6> всегда `0`
```

Регистры PORTB и TRISB

PORTB – это 8-разрядный двунаправленный порт ввода/вывода. Направление определяют биты регистра TRISB.

Три вывода PORTB мультиплицированы со схемой низковольтного программирования: RB3/PGM, RB6/PGC, RB7/PGD.

Каждый вывод PORTB имеет внутренние подтягивающие резисторы, которые могут быть подключены в режиме ввода управляющим битом –RBPU (OPTION_REG<7>).

Четыре канала PORTB <RB7:RB4>, настроенные на вход, могут генерировать прерывания по изменению логического уровня сигнала на входе.

RBO/INT – вход внешнего источника прерываний, который настраивается битом INTEDG (OPTION_REG <6>).

Регистры PORTC и TRISC

PORTC – 8-разрядный двунаправленный порт ввода/вывода. Биты регистра TRISC определяют направление каналов порта.

Выводы PORTC мультиплицированы с несколькими периферийными модулями. В описаниях периферийных модулей указано, как надо настраивать биты регистра TRISC для каждого вывода PORTC.

Регистры PORTD и TRISD

PORTD - 8–разрядный двунаправленный порт ввода/вывода. Управляется битами регистра TRISD.

PORTD может работать как 8– разрядный микропроцессорный порт (ведомый параллельный порт), если бит PSPMODE(TRISE<4>) установлен в `1`.

Регистры PORTE и TRISE

PORTE имеет три вывода, которые индивидуально настраиваются на вход или выход.

Каналы PORTE могут быть управляющими выводами ведомого параллельного порта, когда бит PSPMODE(TRISE<4>) установлен в `1`. В этом режиме биты TRISE<2:0> должны быть установлены в `1`. В регистре ADCON1 выводы PORTE надо настроить как цифровые каналы.

Выводы PORTE мультиплицированы с аналоговыми входами. При этом биты TRISE надо установить в `0`.

Регистры ADCON0 (адрес 1Fh) и ADCON1 (адрес 9Fh)

Регистр ADCON0 используется для настройки модуля АЦП и имеет следующие биты.

Биты 7-6	ADCS1:ADCS0	Выбор источника тактового сигнала 00=Fosc/2, 01= Fosc/8, 10= Fosc/16, 11=F_{RC} (внутренний RC генератор АЦП)
Биты 5-3	CHS2:CHS0	Выбор аналогового канала 000=канал 0, (RA0/AN0) 001=канал 1, (RA1/AN1) 010=канал 2, (RA2/AN2) 011=канал 3, (RA3/AN3) 100=канал 4, (RA4/AN4)

		101=канал 5, (RA5/AN5) 110=канал 6, (RA6/AN6) 111=канал 7, (RA7/AN7)
Бит 2	GO/-DONE	Бит статуса АЦП Если ADON=1 1=модуль АЦП выполняет преобразование (установка бита вызывает начало преобразования) 0=состояние ожидания (аппаратно сбрасывается по завершению преобразования)
Бит 1		Не используется: читается как 0.
Бит 0	ADON	Бит включения модуля АЦП 1=модуль АЦП включен 0=модуль АЦП выключен и не потребляет тока

Результаты преобразования записываются в регистры ADRESH (старший байт) и ADRESL (младший байт)

С помощью регистра ADCON1 устанавливается, какие входы микроконтроллера будут использоваться модулем АЦП и в каком режиме (аналоговый вход или цифровой порт ввода/вывода).

Таблица 6.3

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

A=аналоговый вход D=цифровой канал ввода/вывода.

ADCON1 имеет управляющие биты:

Бит 7 ADFM: Формат сохранения 10-разрядного результата.

1= правое выравнивание, 6 старших бит ADRESH читаются как `0`.

0= левое выравнивание, 6 младших бит ADRESL читаются как `0`.

Биты 6-4 не используются.

Биты 3-0: PCFG3:PCFG0: управляющие биты настройки каналов АЦП (Таблица 6.3).

В описании микроконтроллера дана рекомендованная последовательность действий для работы с АЦП.

6.7. Биты конфигурации

Биты конфигурации расположены в памяти программ по адресу 2007h за пределами пользовательской памяти программ. К регистру конфигурации можно обратиться только в режиме программирования микроконтроллера.

Слово конфигурации содержит 14 бит.

Биты 13-12 Биты 5-4	CP1:CP0 CP1:CP0	Биты защиты программ. Чтобы установить защиту памяти программ, все пары CP1:CP0 должны иметь одинаковое значение. 11=защита памяти программ выключена 10=защищены адреса 1F00h-1FFFh 01=защищены адреса 1000h-1FFFh 00=защищены адреса 0000h-1FFFh
Бит 11	DEBUG	Бит включения внутрисхемной отладки. 1=отладка выключена, RB6 и RB7 работают как каналы ввода/вывода, 0=отладка включена, RB6 и RB7 используются отладчиком.
Бит 10	Не реализован	Читается как `1`

Бит 9	WRT	Бит разрешения записи во FLASH память программ. 1=разрешена запись во FLASH память программ через регистры управления EECON 0= запрещена запись во FLASH память программ через регистры управления EECON
Бит 8	CPD	Бит защиты EEPROM памяти данных 1=защита памяти данных выключена 0= защита памяти данных включена
Бит 7	LVP	Бит разрешения низковольтного программирования 1=вывод RB3/PGM работает как PGM, режим низковольтного программирования включен 0=вывод RB3/PGM работает как цифровой порт ввода/вывода, вывод HV используется для программирования микроконтроллера.
Бит 6	BODEN	Бит разрешения сброса по снижению напряжения питания 1= разрешен сброс BOR 0=запрещен сброс BOR
Бит 3	-PWRTE	Бит разрешения работы таймера включения питания 1=PWRT выключен 0= PWRT включен
Бит 2	WDTE	Бит разрешения работы сторожевого таймера 1=WDT включен 0= WDT выключен
Биты	FOSC1:FOCS0	Биты выбора режима тактового генератора

1-0		11=RC генератор 10= HS генератор 01= XT генератор 00= LP генератор

На этом мы завершим краткое описание микроконтроллера PIC16F877A. Более подробное изучение мы продолжим на примерах проектов устройств с использованием этого микроконтроллера.

Контрольные вопросы

1. Расскажите об особенностях архитектуры микроконтроллера PIC16F877A.
2. Какие дополнительные модули имеет PIC16F877A по сравнению с PIC16F84A?
3. Как организована память PIC16F877A ?
4. Какие регистры специального назначения имеет PIC16F877A и для чего они используются ?
5. Расскажите о назначении выводов микроконтроллера PIC16F877A.
6. Какие порты ввода/вывода имеет PIC16F877A ?
7. Что означает мультиплицирование портов и как оно выполняется ?
8. Какие регистры используют для настройки модуля АЦП и как это делают ?
9. Какой порт можно использовать как аналогоый вход АЦП и сколько аналоговых каналов можно подключить к PIC16F877A?
10. Расскажите о назначении битов слова конфигурации.

Глава 7. ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ В СРЕДЕ mikroC

Компилятор MPLAB XC8 C подходит и для PIC16F877A. Однако, в этом компиляторе библиотека функций невелика и для подключения к микроконтроллеру, например, жидкокристаллического дисплея требуется составлять достаточно длинную программу на языке ассемблера или Си.

В настоящее время разработчики программ для микроконтроллеров используют новую среду mikroC PRO for PIC компании MikroElektronika [6], которая имеет большой набор готовых подпрограмм и функций, выполняющих сопряжение микроконтроллеров с многочисленными типами интерфейсов и стандартных внешних устройств.

Работа с программой mikroC не сильно отличается от изученной нами изученной нами программы MPLAB IDE. Поэтому вполне разумно потратить некоторые усилия на освоение этого нового инструмента и сэкономить время в будущем на программировании практических задач.

Программу mikroC можно получить на сайте www.mikroe.com. Мы будем изучать версию 6.5.0. Вообще, для профессионального применения программа требует регистрации и лицензии. Однако, если скомпилированный Вами исполняемый HEX-файл не превышает 2 Кбайт, то можно работать бесплатно. Для учебных программ это вполне подходит, так как все последующие программы в этой книге работали в mikroC.

7.1. Создание проекта в mikroC

Начнем изучение mikroC с простого устройства на микроконтроллере PIC16F877A с мигающими светодиодами (рис.7.1). Светодиоды подключены к выводам порта B и должны мигать с постоянным периодом.

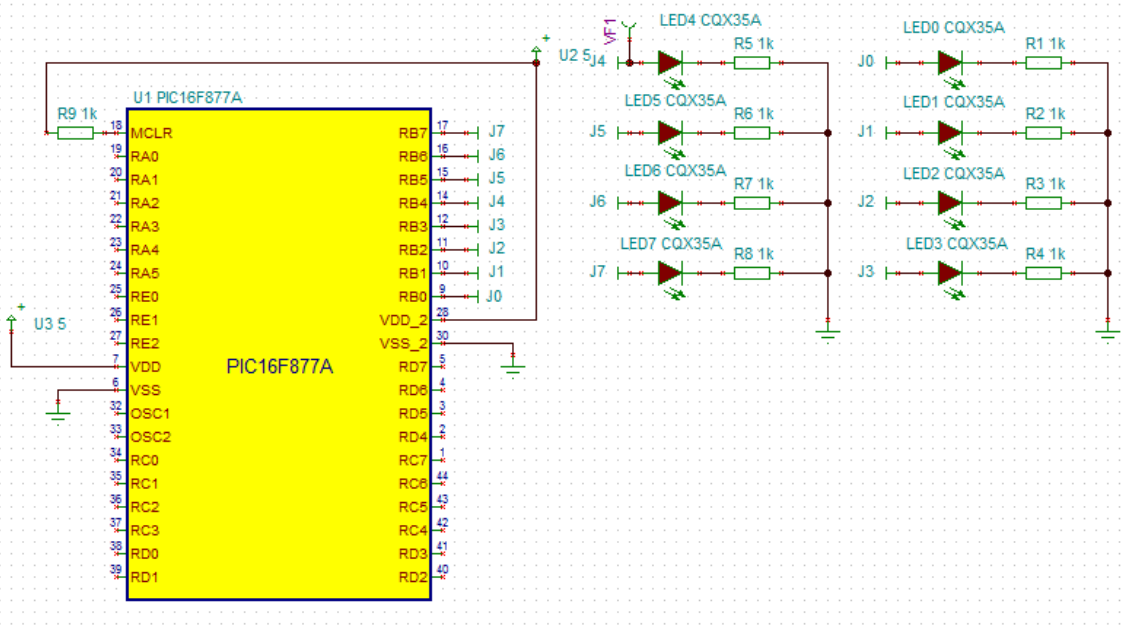


Рис.7.1. Схема с мигающими светодиодами

Создаем новый проект в mikroC под названием Led-blink в папке MKC-Led Blinking. Для этого выбираем *Project -New Project* и в окне *Welcome to the New Project Wizard* нажимаем *Next.* В окне *Step 1: Project Settings* (рис.7.2) задаем имя проекта, папку хранения, тип микроконтроллера и тактовую частоту.

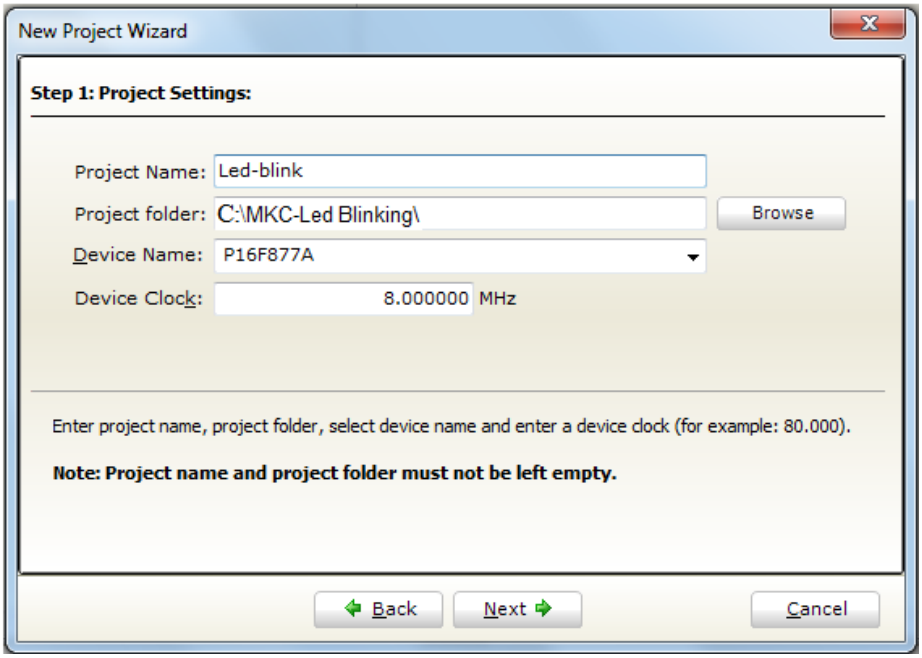


Рис.7.2. Окно задания имени и папки проекта, типа микроконтроллера и частоты

На втором шаге можно добавить в проект дополнительные файлы, которые могут понадобиться: некоторые заголовки, исходные файлы и т.п. Так как мы создаем первый простой проект, мы не будем добавлять никакие файлы и нажимаем *Next* (рис.7.3).

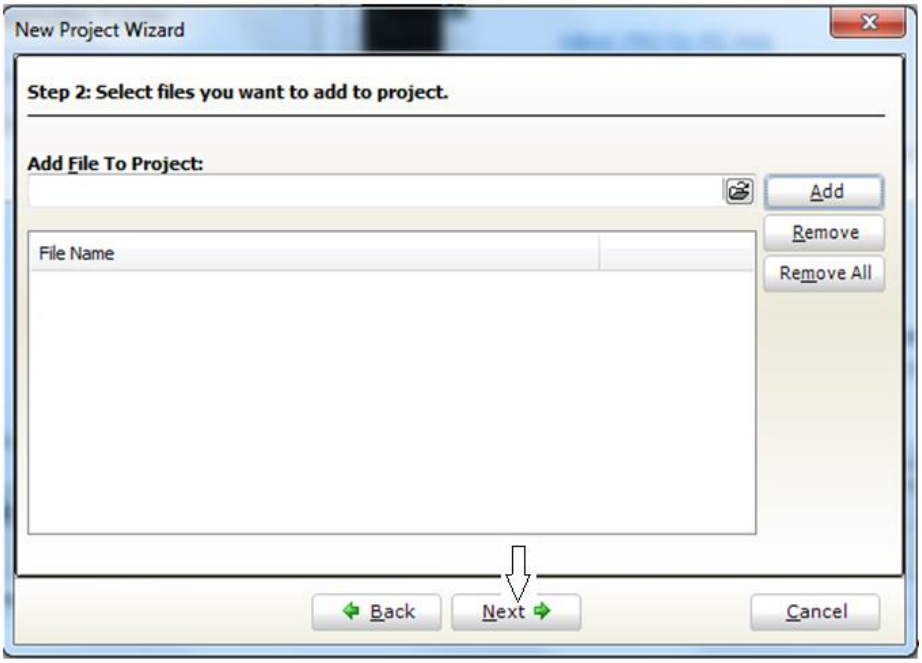


Рис.7.3. Добавление исходного файла в проект

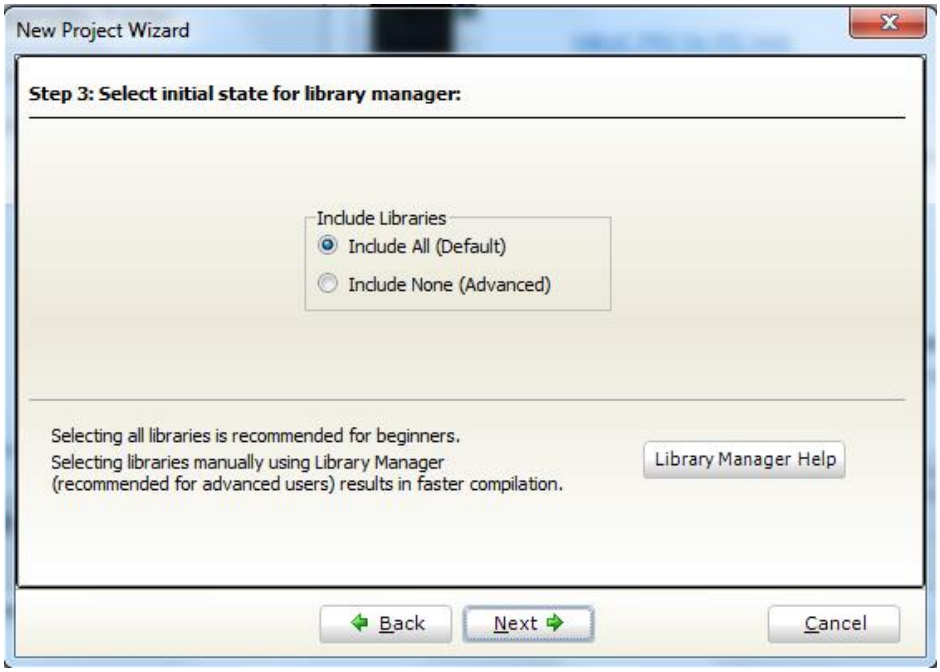


Рис.7.4. Подключение библиотек

На третьем шаге мы подключаем к проекту библиотеки (рис.7.4).

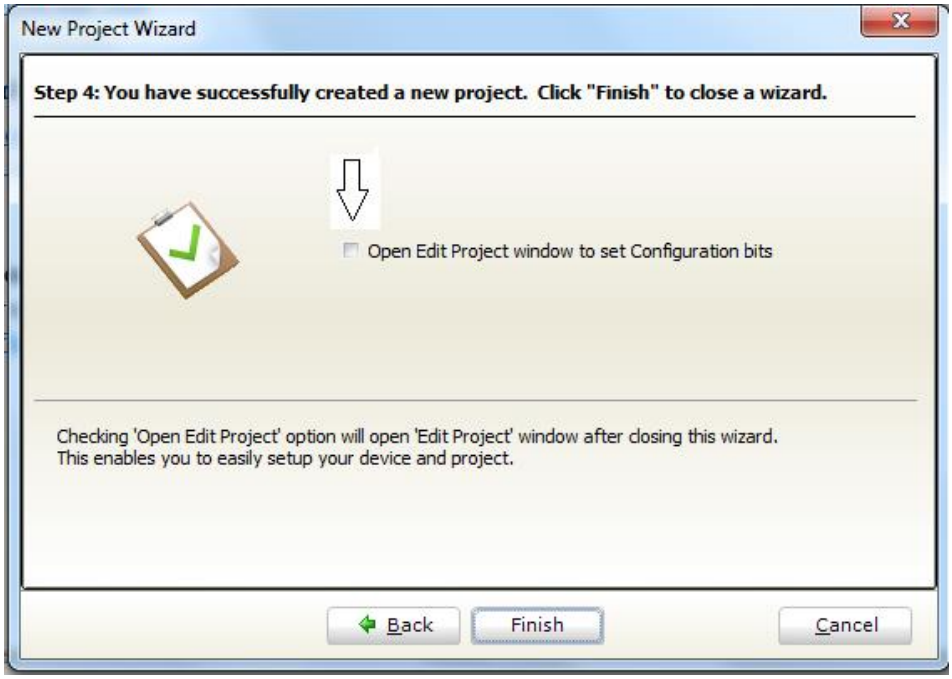


Рис.7.4. Окно завершения создания проекта

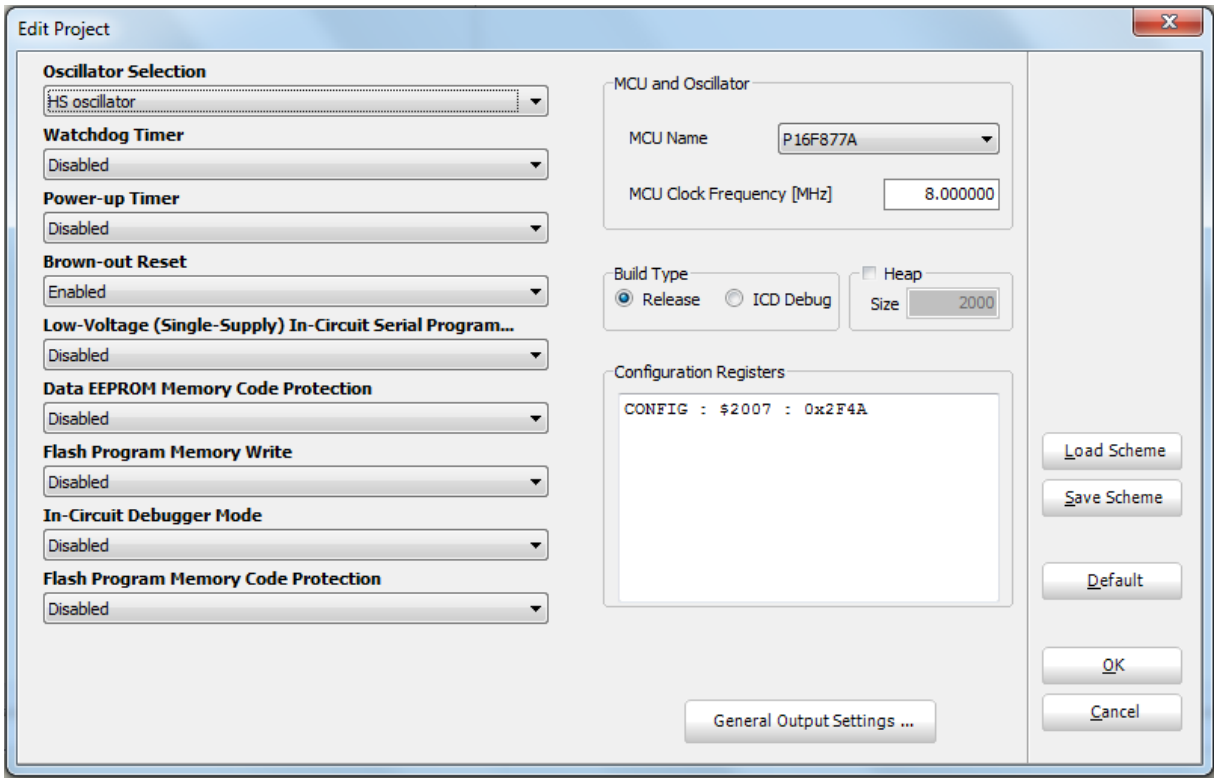


Рис.7.5. Окно установки битов конфигурации микроконтроллера

Четвертый шаг подтверждает успешное создание нового проекта (рис.7.4) и позволяет открыть окно редактора проекта для установки битов конфигурации микроконтроллера (рис.7.5). Впрочем, это можно не делать. Тогда биты будут установлены по умолчанию.

В окне (рис.7.5) можно загрузить одну из нескольких наиболее популярных схем осциллятора и сохранить эту схему.

На вкладке Build Type можно задать вид компиляции: для внутрисхемной отладки (ICD Debug) или для окончательного кода (Release).

На этом создание проекта завершается и открывается рабочее поле (бланк) среды mikroC (рис.7.6).

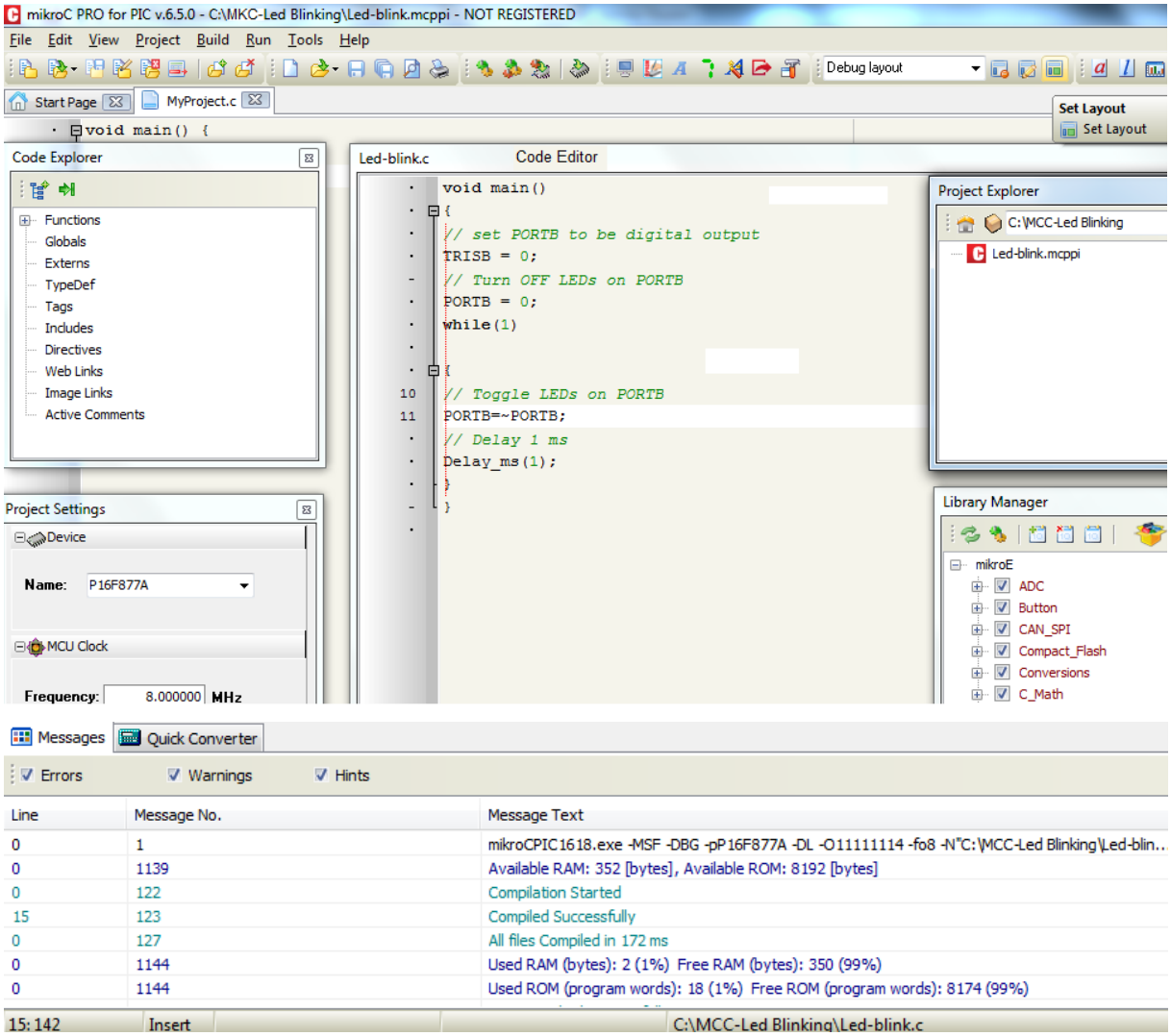


Рис.7.6. Рабочее поле среды mikroC

Ввод в проект программы

На рабочем поле находится редактор кода (*Code Editor*) с настраиваемой контекстной подсветкой, помощью в коде (*Code Assistant*), помощью в параметрах (*Parameters Assistant*), автокоррекцией общих опечаток (*Auto Correct*) и кодовыми шаблонами с автозавершением (*Auto Complete*).

В окне *Code Editor* набирем текст программы для схемы с мигающими светодиодами. Возможный вариант исходного файла показан в листинге 7.1.

Листинг 7.1.

```
// Led-blink.c
void main()
{
TRISB = 0; // установка цифровых выходов PORTB
PORTB = 0; // выключение светодиодов на выходах PORTB
while(1) //бесконечный цикл
{
PORTB=~PORTB;/* переключение светодиодов на выходах
                      PORTB*/
Delay_us(100);  // задержка переключения 100 мкс
  }
    }
```

В программе использована использована библиотечная функция задержки Delay_us(100). Длительность задержки составит 100 мкс.

В редакторе кода можно редактировать проект и повторно ввести биты конфигурации, выбрав *Project-Edit Project.*

Можно создать новый проект (*File-New Project*), новый исходный файл (*File-New Unit*), открыть существующий файл (*File-Open*), напечатать файл программы (*File-Print*). Открытый файл выводится на собственной вкладке.

Можно выполнить сохранение и закрытие файла.

Внизу рабочего поля находится окно ошибок (*Error Window*), которое отображает все ошибки, обнаруженные во время компиляции и компоновки.

7.2. Установка расположения окон на рабочем поле

На вкладке *View* можно открыть и закрепить на рабочем поле много дополнительных окон. Проводник по коду (*Code Explorer*) с окнами просмотра клавиатурных команд и быстрой справки служит для облегчения управления проектом.

Общие настройки проекта могут быть сделаны в окне настроек проекта (*Project Settings*).

Менеджер библиотек (*Library Manager*) включает сведения о всех библиотечных функциях, используемым в проекте.

Можно увеличить пространство для просмотра и редактирования кода, расположив окна нужным образом.

Щелкните дважды по заголовку окна и появится канцелярская кнопка автоматического скрытия окна Auto Hide (рис.7.7).

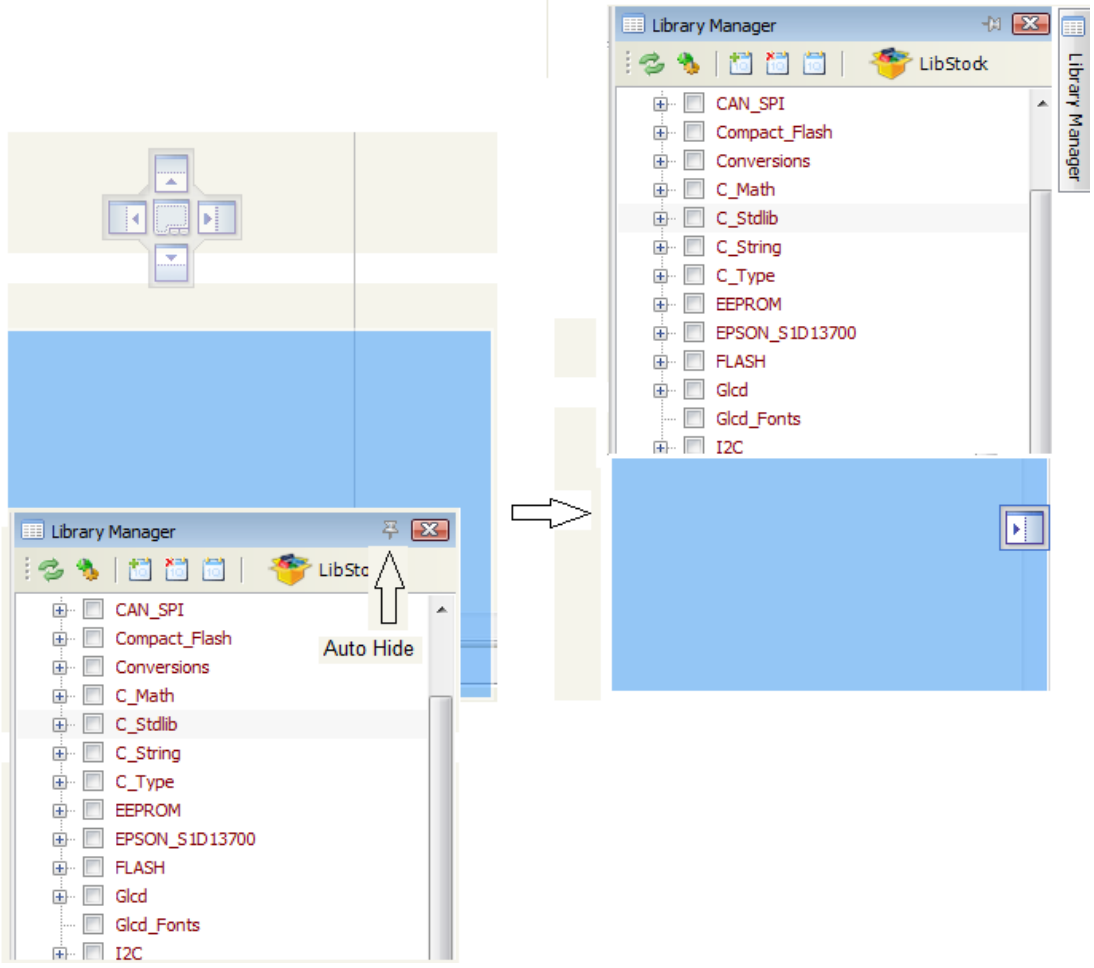


Рис.7.7. Привязка и автоматическое скрытие окон

Сначала ставим канцелярскую кнопку вертикально. Перетаскиваем окно к центру поля и видим крест из четырех кнопок выбора привязки к сторонам поля. Одновременно появляются четыре кнопки на сторонах поля. Надо перетащить окно к выбранной стороне и коснуться курсором кнопки выбора привязки. После этого канцелярскую кнопку в заголовке окна надо перевести в горизонтальное положение. Окно будет автоматически скрываться, если щелкнуть в центре рабочего поля.

Новые окна можно открыть, выполнив *View-Windows* и выбрав нужное окно на вкладке *Windows List*.

Выбранное расположение окон можно сохранить в проекте. Для этого вводим название расположения и нажимаем кнопку *Save Layout* (рис.7.8). Кнопка *Set Layout* восстановит запомненное расположение. Кнопка *Delete Layout* удалит это расположение из памяти проекта.

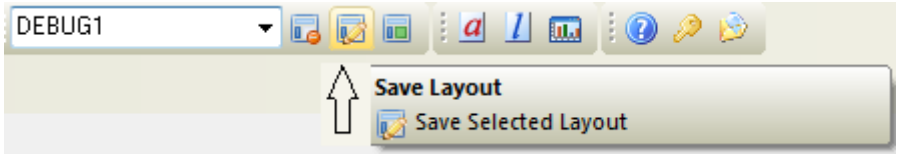


Рис.7.8. Сохранение расположения окон

7.3. Компиляция и проверка первой программы

Теперь испытаем нашу первую программу. Для этого выполним компиляцию. В главном меню выбираем *Build*, на вкладке еще раз выбираем *Build*. В окне ошибок к нашему удовольствию получаем *Finished Successfully*.

Загружаем HEX – файл и LST-файл в модель (рис.7.1), в интерактивном режиме делаем *Start* и наблюдаем мигание светодиодов.

В режиме *Analysis-Transient* получим временную диаграмму напряжения на первом светодиоде. Период включения равен 200 мкс (рис.7.9). Первая программа работает правильно !

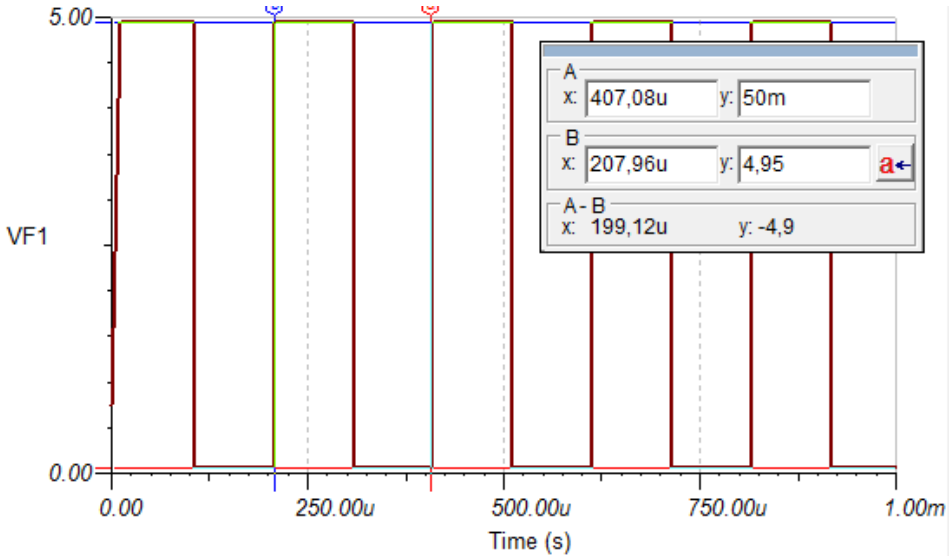


Рис.7.9. Сигнал на светодиоде

7.4. Расширенные возможности редактирования

Редактор исходного кода - это современный текстовый редактор, который включает возможности копирования, вставки и отката, общие для среды Windows.

Настройку редактора кода можно выполнить, выбрав *Tools-Options*. На вкладках *Editor Settings*, *Editor Colors*, *Auto Correct*, *Auto Complete*, *IDE Style* при желании можно изменить настройки редактора, цвета отображения, автокоррекцию ошибок, использовать кодовые шаблоны и т.д.

Помощник в коде *Code Assistant* работает так. Если напечатать первые несколько символов слова и потом нажать *Ctrl+Space*, все разрешенные идентификаторы, соответствующие напечатанным символам, будут предложены в всплывающем окне (рис.7.10). Теперь можно продолжить печать для сужения выбора или с помощью стрелок на клавиатуре выбрать подходящий вариант из предложенного и нажать *Enter*.

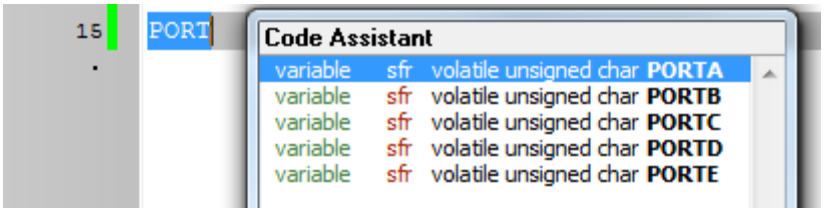


Рис.7.10. Окно *Code Assistant*

Можно легко перейти на нужную строку программы, если нажать *Ctrl+G* и ввести номер строки.

Проводник по коду *Code Explorer* дает представление о каждом из объявленных элементов исходного кода (рис.7.11). Можно выполнить копирование любого элемента или дерева кода на рабочий стол с помощью "правого клика" на него в окне проводника. Для развертывания или свертывания дерева кода в окне проводника используется иконка.

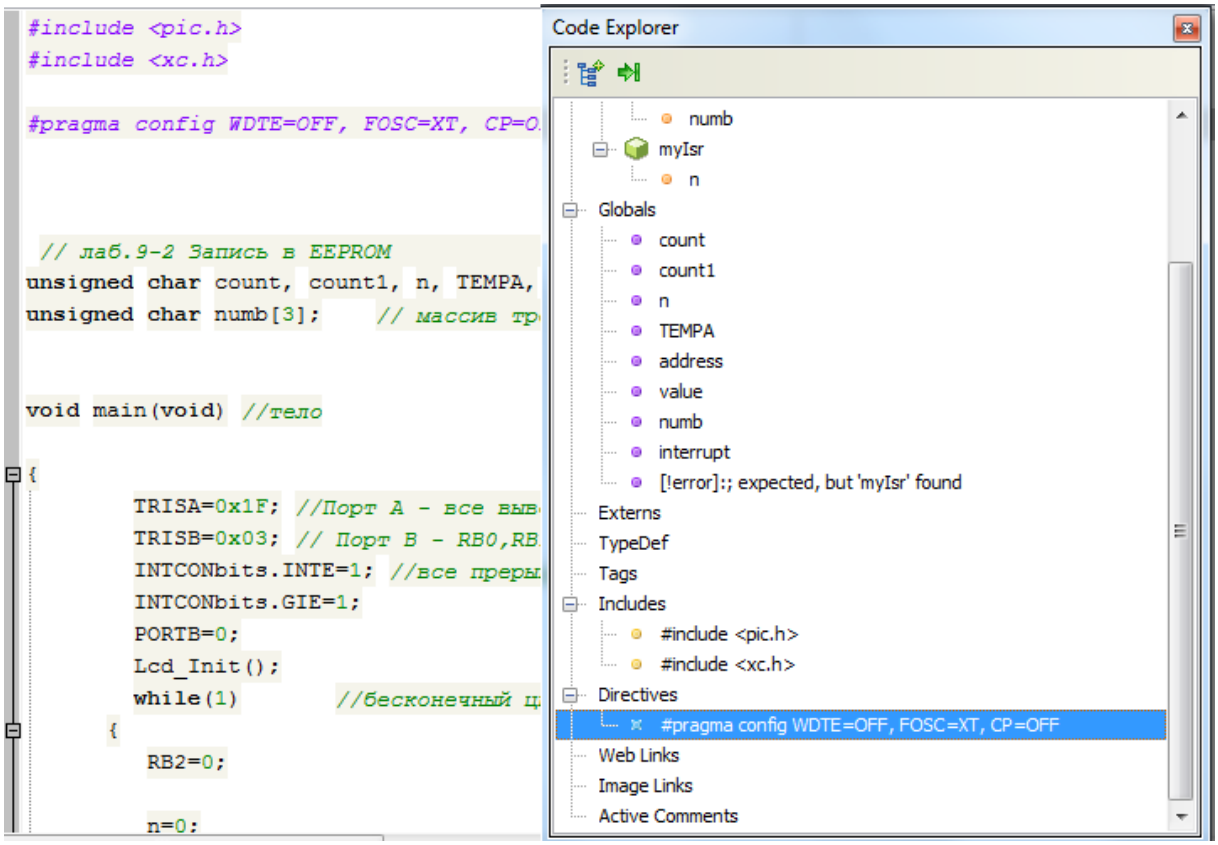


Рис.7.11. Окно *Code Explorer*

Лист программ (рис.7.12) открывается нажатием клавиш *Ctrl+L* и показывает список программ в файле и их начальные адреса. Двойным щелчком левой кнопкой мыши можно легко перейти в начало нужной программы.

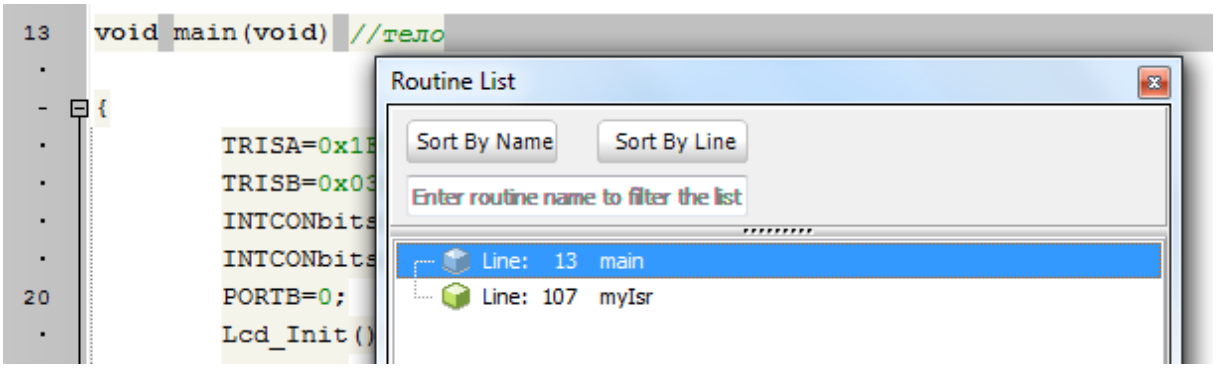


Рис.7.12. Лист программ

7.5. Настройка проектов

Среда mikroC организует приложения как проекты, состоящие из одного файла проекта (с расширением `.ppc`) и одного или больше файлов исходного кода (расширение `.c`). Компилировать исходный файл можно, только если он является частью проекта.

Файл проекта содержит следующую информацию:

- имя проекта и необязательное описание;
- целевое устройство (тип микроконтроллера;
- установки устройства (слово конфигурации);
- тактовая частота устройства;
- список исходных файлов проекта с путями.

Создание нового проекта с помощью *New Project Wizard* мы уже рассмотрели. При необходимости можно выполнить редактирование проекта, используя выпадающее меню *Project-Edit Project*. Проект можно переименовать, изменить его описание, используемый кристалл, частоту, слова конфигурации и т.п. Чтобы удалить проект, достаточно удалить папку, в которой располагается файл проекта (расширение .ppc).

На рабочем поле можно установить окно *Project Manager* (рис.7.13).

Если мы создали уже несколько проектов, их можно объединить в группу и сохранить как группу, щелкнув по значку  *Save Project Group* . Группу проектов можно открыть,

щелкнув по значку *Open Project Group*. Все данные о группе проектов хранятся в групповом файле группы (расширение *.mpgroup*). В группу можно добавить новый проект, щелкнув . Для того, чтобы закрыть проект из группы, надо щелкнуть . Закрыть всю группу проектов можно кнопкой *Clouse Project Group* .

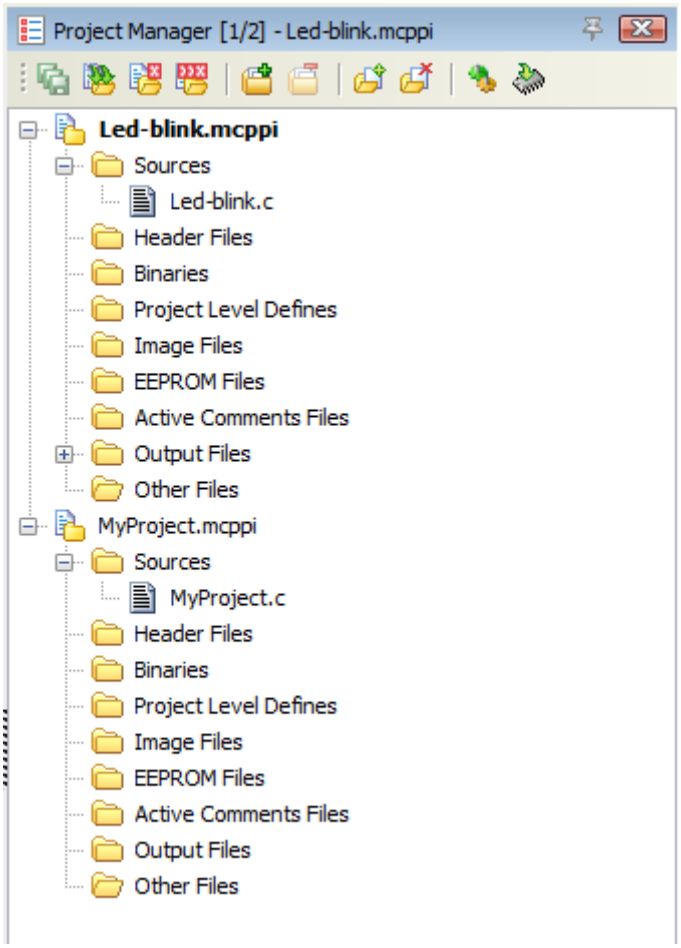


Рис.7.13. Окно менеджера проектов

Проект может содержать произвольное количество исходных файлов (расширение `.c`). Список имеющих отношение к проекту файлов сохраняется в файле проекта (расширение `.ppc`).

Чтобы добавить исходный файл в проект, надо выбрать *Project-Add to Project* из выпадающего меню или кликнуть на иконку добавления файла к проекту  . Каждый добавляемый исходный файл должен после препроцессорной обработки содержать все необходимые определения.

Для удаления файла из проекта следует выбрать *Project-Remove from Project* из выпадающего меню или кликнуть на иконку удаления файла из проекта  .

Из окна менеджера проектов можно выполнить компиляцию, нажав *Build Project*  , и включить программатор  .

Директива препроцессора `#include` позволяет вставлять заголовочные файлы (расширение `.h`) в исходный код.

Синтаксис директивы `#include` имеет два формата.

Если использован формат `#include <header_name>`, поиск будет производиться последовательно в следующих каталогах:

1. в подкаталоге "`include`" каталога, куда установлен mikroC;
2. в каталогах, пути к которым указаны настройках проекта.

Если использован формат `#include "header_name"`, поиск будет производиться последовательно в следующих каталогах:

1. в каталоге проекта (это том, который содержит файл проекта `.ppc`);
2. в подкаталоге "`include`" каталога, куда установлен mikroC;
3. в каталогах, пути к которым указаны настройках проекта.

Если путь к заголовочному файлу `header_name` указан явно, поиск будет производиться только в указанном каталоге. Например:

```
#include "C:\my_files\test.h"
```

7.6. Компиляция проекта

После редактирования выполняем компиляцию исходного файла с расширением .c. Для этого в главном меню выбираем *Build*  . Если открыто сразу несколько проектов, можно компилировать все сразу, выбрав *Build All Projects*  .

Результат компиляции мы увидим в окне ошибок. В документации mikroC есть подробное описание сообщений об ошибках. При необходимости придется обращаться к этим указаниям. Среди сообщений об ошибках есть такое: *Output limit has raised 2K words*. Это значит, что бесплатная программа прекратила работу и надо получить лицензию или уменьшить программу.

После успешной компиляции в окне менеджера проектов (рис.7.14) появляются четыре выходных файла с расширениями .hex, .asm, .lst, .log. Каждый файл можно посмотреть в окне редактора кода, дважды щелкнув по нему левой кнопкой мыши. Точный размер HEX – файла можно узнать в свойствах файла в папке проекта. К примеру, файл Led-blink.hex занимает 125 байт.

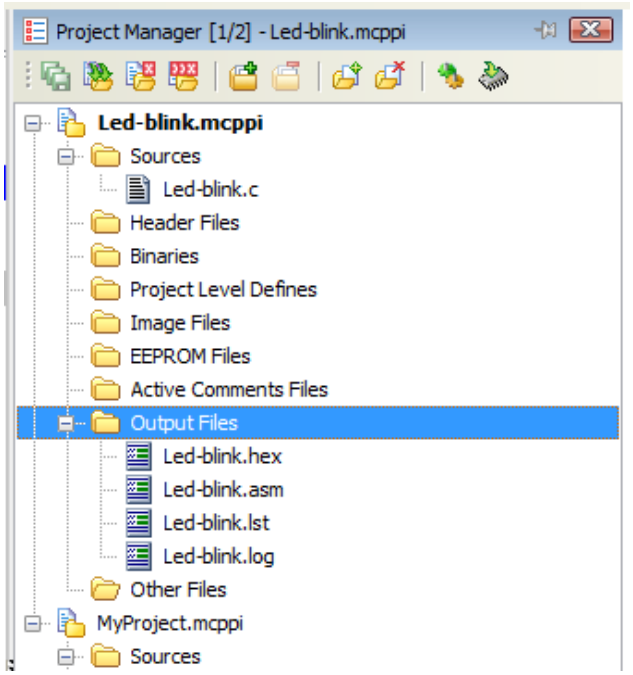


Рис.7.14. Выходные файла проекта

7.7. Отладка программы

После успешной компиляции проекта можно приступить к отладке программы.

Возможны два варианта: использование программного симулятора или использование внутрисхемного симулятора mikroICD, когда отладка проводится в реальном времени на реальном микроконтроллере. Второй метод требует дополнительных аппаратных средств. Поэтому подробно рассмотрим программный симулятор, который является встроенным программным средством mikroC. Для включения программного симулятора в окне *Project Setting* на вкладке *Build Type* надо установить *Release*, а на вкладке *Debuger* установить *Software*. После успешной компиляции проекта можно запустить программный симулятор.

На вкладке *Run* и на панели отладчика *Debuger Toolbar* есть также опции:

Start Debuger  (F9).

Run/Pause Debuger  (F6).

Stop Debugger  (Ctrl+F2).

Step Into  (F7). Выполняет текущую строку программы, затем останавливается. Если выполняемая строка программы вызывает другую подпрограмму, отладчик входит в подпрограмму и остановливается после выполнения первой инструкции.

Step Over  (F8). Выполняет текущую строку программы, затем останавливается. Если выполняемая строка программы вызывает другую подпрограмму, то отладчик не будет входить в нее. Целая подпрограмма будет выполняться и отладчик остановится на первой инструкции после вызова программы.

Step Out  (Ctrl+F8). Выполняет все остающиеся строки программы в пределах подпрограммы. Отладчик останавливается непосредственно после выхода из подпрограммы.

Run to Cursor  (F4). Выполняет программу до достижения позиции курсора.

Toggle Breakpoints  (F5). Опция переключения точек останова. Устанавливает новые точки останова или удаляет уже установленные в текущей позиции курсора.

При отладке можно использовать несколько вспомогательных окон, которые открываются на вкладке *View-Windows*.

Окно *Breakpoints* (рис.7.15) показывает все точки останова и строки, в которых эти точки находятся.

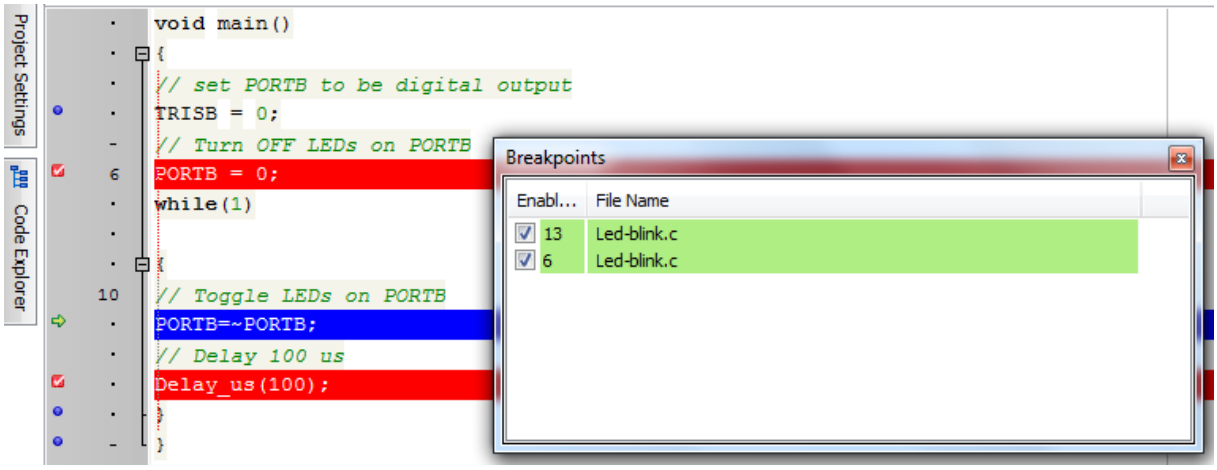
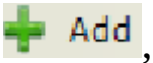


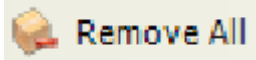
Рис.7.15. Окно точек останова

Окно наблюдения отладчика - это основное окно отладки, которое позволяет контролировать элементы программы во время отладки. Чтобы открыть окно наблюдения, надо выбрать *View-Debug Windows-Watch Window* из выпадающего меню (рис.7.16).

Окно наблюдения отображает переменные и регистры микроконтроллера вместе с их адресами и значениями. Переменные и регистры выбирают из выпадающего листа и пользуются кнопками 

, 

, 

, 

.

Можно выделить значение переменной, выбрать *Properties* и изменить значение, используя различные представления чисел.

В окне наблюдения для удобства дублируется панель управления отладчика.

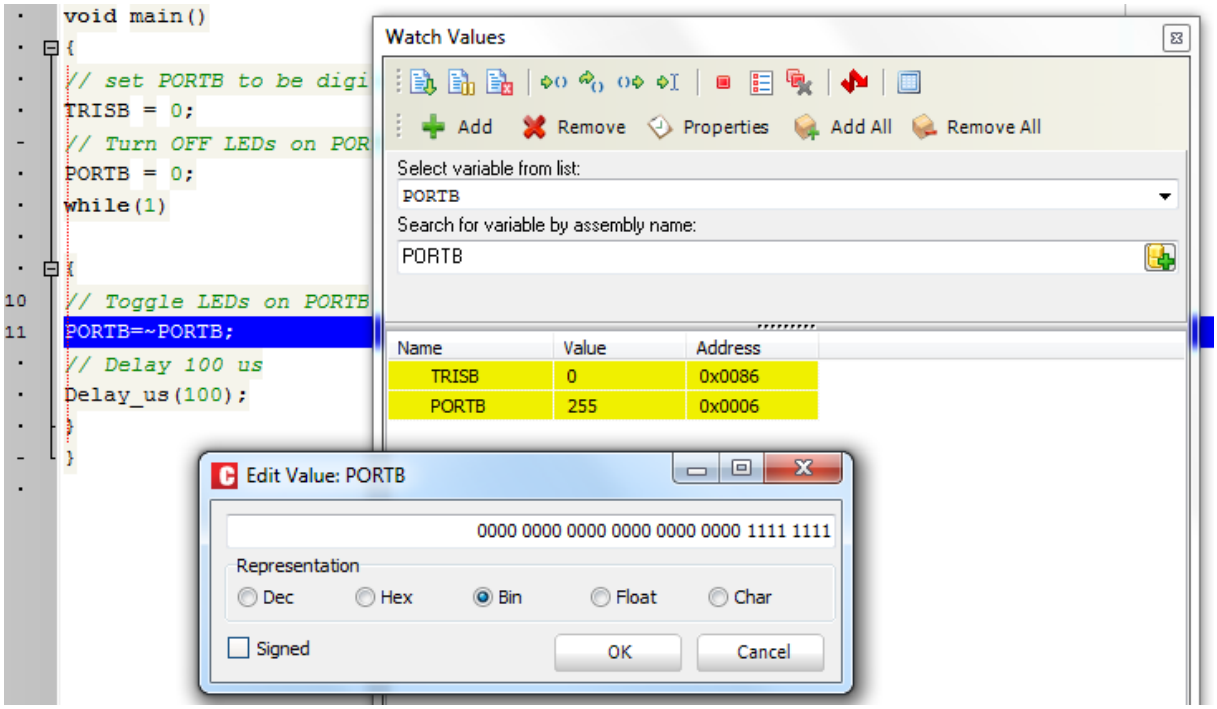


Рис.7.16. Окно наблюдения

Окно просмотра памяти (рис.7.17) доступно из выпадающего меню: *View-Debug Windows-RAM Windows***.** Самые последние изменения отмечены красным цветом.

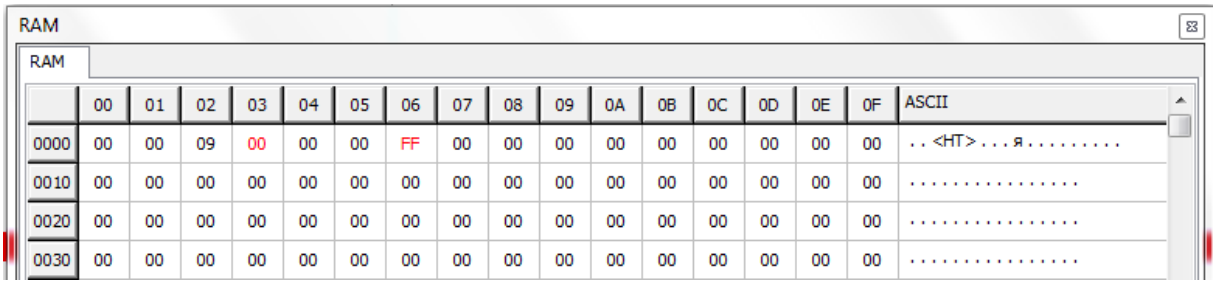


Рис.7.17. Окно просмотра памяти

Окно хронометража (Stopwatch Window) (рис.7.18) отладчика доступно из выпадающего меню: *View-Debug Windows-Stop Watch.* В окне хронометража отображается текущий счетчик (*Current count*) периодов тактовой частоты и секунд от момента запуска отладчика. Секундомер (*Stopwatch*) измеряет время исполнения в периодах тактовой частоты и

секундах от момента запуска отладчика и может быть сброшен в `0` в любое время. Разность (*Delta*) представляет фактическое время выполнения участка программы от предыдущей точки останова до текущей в периодах и секундах (при пошаговом исполнении показывает время выполнения одной строки кода программы на Си). Также в окне отображается текущая тактовая частота микроконтроллера (*Clock*).

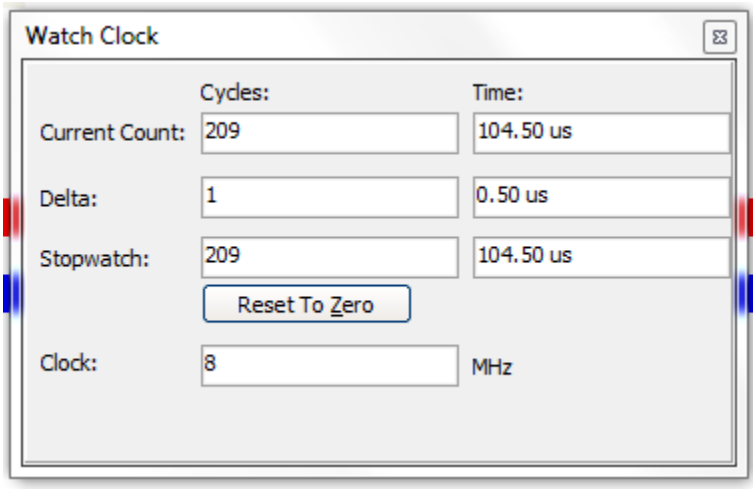


Рис.7.18. Окно хронометража

Примечание: Тактовую частоту в окне хронометража можно менять, это приведет к пересчету времени в секундах. Это изменение не влияет на текущие установки проекта, где тоже задана тактовая частота микроконтроллера, а влияет только на расчет времени симуляции.

Программный симулятор моделирует процесс выполнения программы и выполнение отдельных команд, но это не может полностью эмулировать поведение устройства, так как не происходит обновления таймеров, флагов прерывания, и т.д.

7.8. Статистика

После успешной компиляции можно посмотреть статистические сведения о коде. Следует выбрать *Project-View Statistics* из выпадающего меню или кликнуть на иконку статистики. Появится окно с 11 вкладками, на которых представлены следующие сведения: использование памяти RAM; переменные, отсортированные по адресам; расположение

регистров специального назначения; использование памяти ROM; постоянные памяти ROM; функции с их адресами; функции, отсортированные по имени, размеру и адресам; дерево функций; суммарные результаты по RAM и ROM памяти.

7.9. Встроенные средства

В mikroC включены встроенные вспомогательные средства, которые можно запустить из выпадающего меню *Tools*:

терминал связи по USART для работы с интерфейсом RS232;

карта ASCII - практически полезное средство при работе с LCD;

редактор EEPROM для действий с памятью микроконтроллера, при использовании которого компилятор генерирует Intel hex файл (*Project*_name.ihex), содержащий данные от редактора;

редактор графического дисплея Graphic Lcd Bitmap Editor, который формирует код, совместимый с PIC микроконтроллерами;

HID-терминал для USB связи;

ЖК-редактор пользовательских символов с совместимым для PIC выходным кодом;

загрузчик - это небольшая программа для приема и записи в память микроконтроллера новой прошивки (возможно в PIC16F87X).

Имеется еще несколько встроенных средств.

7.10. Менеджер библиотек

Library Manager обеспечивает простую работу с библиотеками, которые будут использоваться в проекте. В окне диспетчера перечислены все библиотеки (расширение .mcl), которые сохраняются в папке компилятора. Нужную библиотеку можно добавить в папку *Uses* проекта, отметив галочкой поле рядом с библиотекой. Можно подключить все библиотеки, нажав кнопку *Check All*, или отключить все библиотеки, нажав *Clear All*. Только выбранные библиотеки будут участвовать в компоновке.

Менеджер библиотек управляется следующими кнопками:

 -обновить файлы в папке *Uses* после добавления новых файлов;

 - перестроить все доступные библиотеки, когда доступные источники библиотек нуждаются в обновлении;

 -включить все доступные библиотеки в текущий проект;

 - библиотеки не включены в текущий проект;

 - восстановить библиотеку на состояние до последнего сохранения проекта.

Контрольные вопросы и задания

1. Установите в своем компьютере программу mikroC PRO for PIC v.6.5.0 или откройте уже установленную программу. Выполните самостоятельно создание проекта Led-Blinking, введите программу (листинг 7.1), выполните компиляцию и проверку работы программы в среде TINA.
2. Как выполняется установка окон на рабочем поле ?
3. Как сохранить расположение окон в проекте ?
4. Как можно использовать менеджер проектов ?
5. Какие выходные файлы проекта формируются после умпешной компиляции ?
6. Назовите основные опции отладчика программ.
7. Как можно использовать окно наблюдения ?
8. Как используют окно хронометража?
9. Откройте менеджер библиотек, в разделе Hardware Libraries посмотрите, например, библиотеки ADC, LCD, PWM. Расскажите о содержании этих разделов.

Глава 8. СРЕДА СКВОЗНОГО ПРОЕКТИРОВАНИЯ Proteus VSM

Моделирование сложных микропроцессорных устройств с разнообразной периферией мы будем проводить в программной среде Proteus фирмы Labcenter Electronics [9]. Эта система имеет более обширную библиотеку моделей электронных компонентов, периферийных модулей и микроконтроллеров по сравнению с программой TINA. Интерфейс Proteus сложнее, чем в программе TINA, но его освоение даст Вам новые мощные инструменты проектирования микропроцессорных устройств.

Proteus VSM, созданная фирмой Labcenter Electronics, является интерактивной средой моделирования и сквозного проектирования. Это означает создание устройства, начиная с его графического изображения (принципиальной схемы) и заканчивая изготовлением печатной платы устройства, с возможностью контроля на каждом этапе производства.

В каталоги примеров Proteus VSM входят как простейшие аналоговые устройства, так и сложные системы, созданные на популярных микроконтроллерах. Доступна огромная библиотека моделей элементов, пополнять которую может сам пользователь. Возможность анимации схем позволяет программе стать прекрасным учебным пособием в школах и вузах. Достаточный набор инструментов и функций, среди которых вольтметр, амперметр, осциллограф, всевозможные генераторы, способность отлаживать программное обеспечение микроконтроллеров делают Proteus VSM хорошим помощником разработчику электронных устройств.

Proteus VSM состоит из двух самостоятельных программ: *ISIS и ARES*. *ISIS* – это программа моделирования электронных схем. *ARES* – программа разработки печатных плат с возможностью создания своих библиотек корпусов.

Основной программой является *ISIS*, в ней предусмотрена связь с *ARES* для передачи проекта для разводки платы.

Мы будем изучать моделирование микроконтроллеров в программе *ISIS*, а инструменты аналогового моделирования

рассмотрим кратко. При необходимости Вы можете воспользоваться справочными данными из меню *Help.*

8.1. Создание нового проекта

В Proteus 8.1 все необходимые для проектирования устройства файлы должны быть собраны в общей базе данных, которую называют проектом. На главной странице программы можно открыть существующий проект, создать новый проект или импортировать уже существующую принципиальную схему и макет с помощью соответствующих кнопок.

Создание нового проекта выполним с помощью мастера *Project Wizard.* Используем обучающий пример *PIC Traffic Light Controller* из раздела *Help* программы Proteus. Создадим папку PRO-Traffic для размещения проекта и на первом шаге указываем имя проекта и путь к этой папке (рис.8.1).

На втором шаге выбираем шаблон для проектирования схемы (рис.8.2). Третий шаг служит для выбора шаблона проектирования печатной платы (рис.8.3).

На четвертом шаге мы устанавливаем тип известного нам микроконтроллера PIC16F84A и компилятора MPASM, который используется для программ, написанных в ассемблере (рис.8.4). Выбрав *Create Quick Start Files*, мы получим заготовку проекта программы с установленными параметрами для компиляции. Итоговое окно (рис.8.5) показывает все выполненные установки.

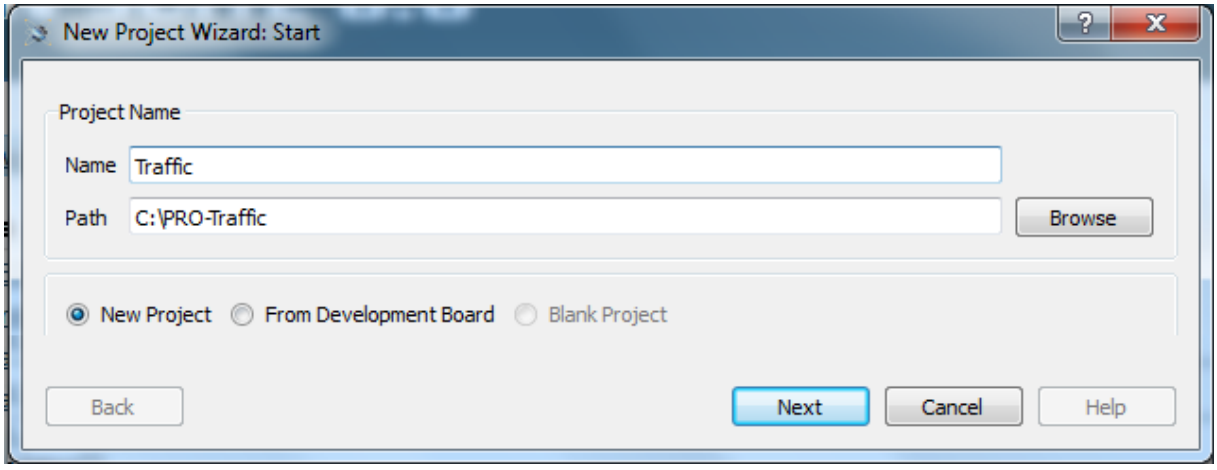


Рис.8.1. Первый шаг создания проекта

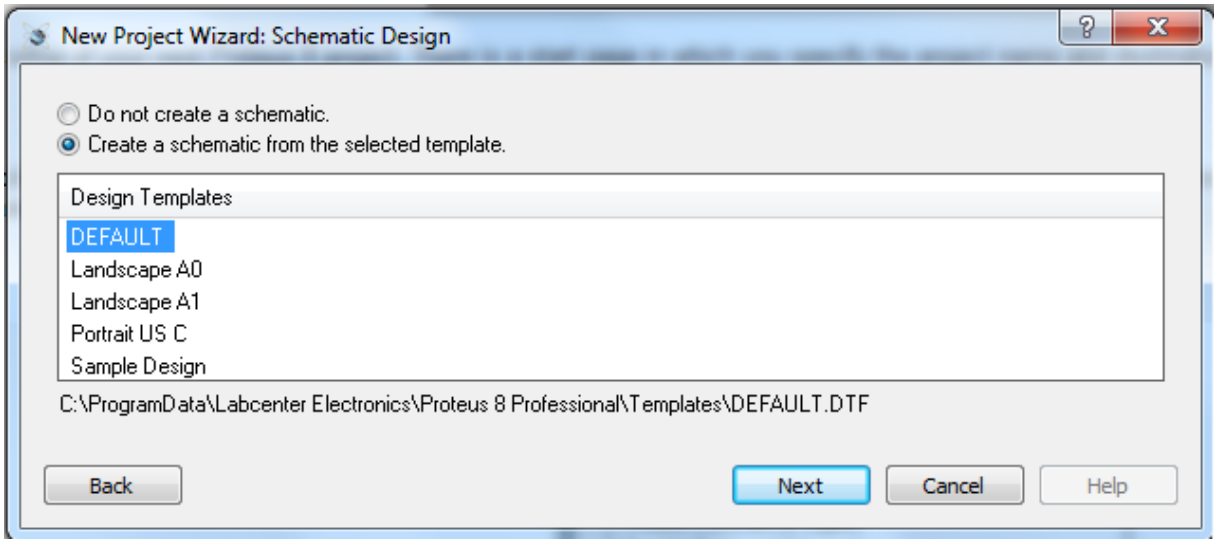


Рис.8.2. Выбор шаблона для проектрования схемы

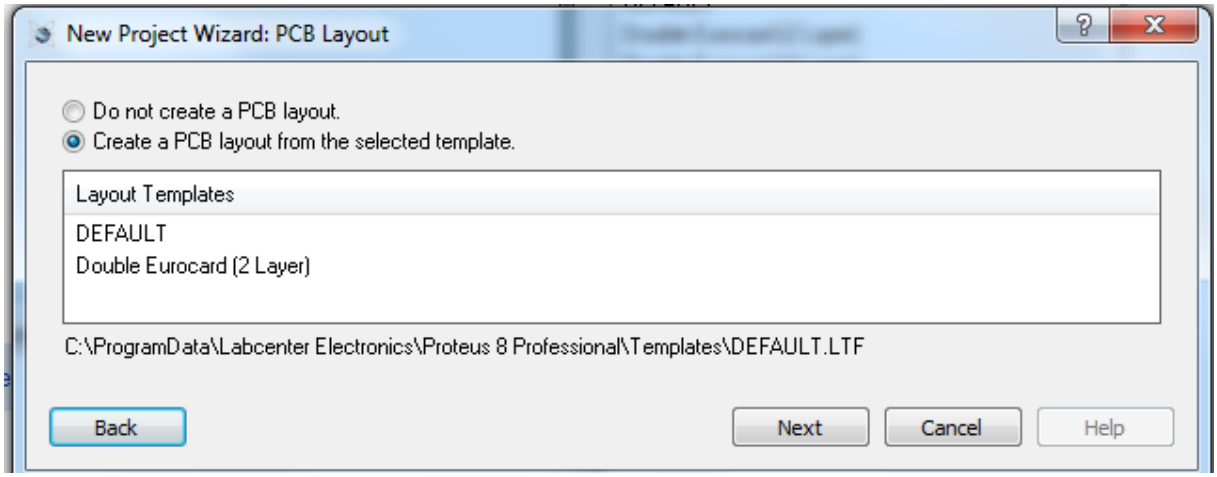


Рис.8.3. Выбор шаблона для проектирования печатной платы

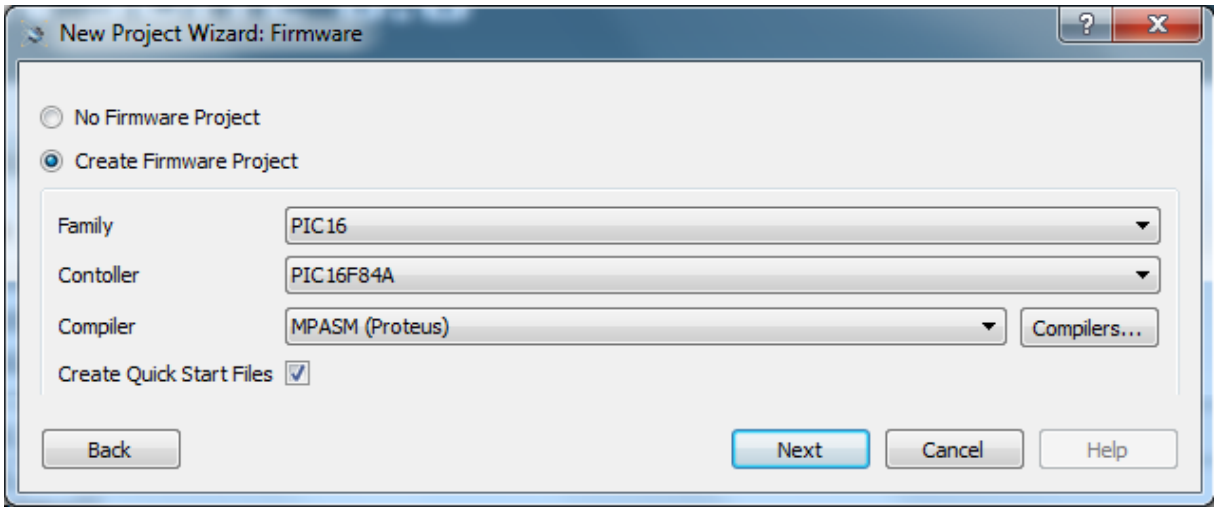


Рис.8.4. Выбор типа микроконтроллера и компилятора

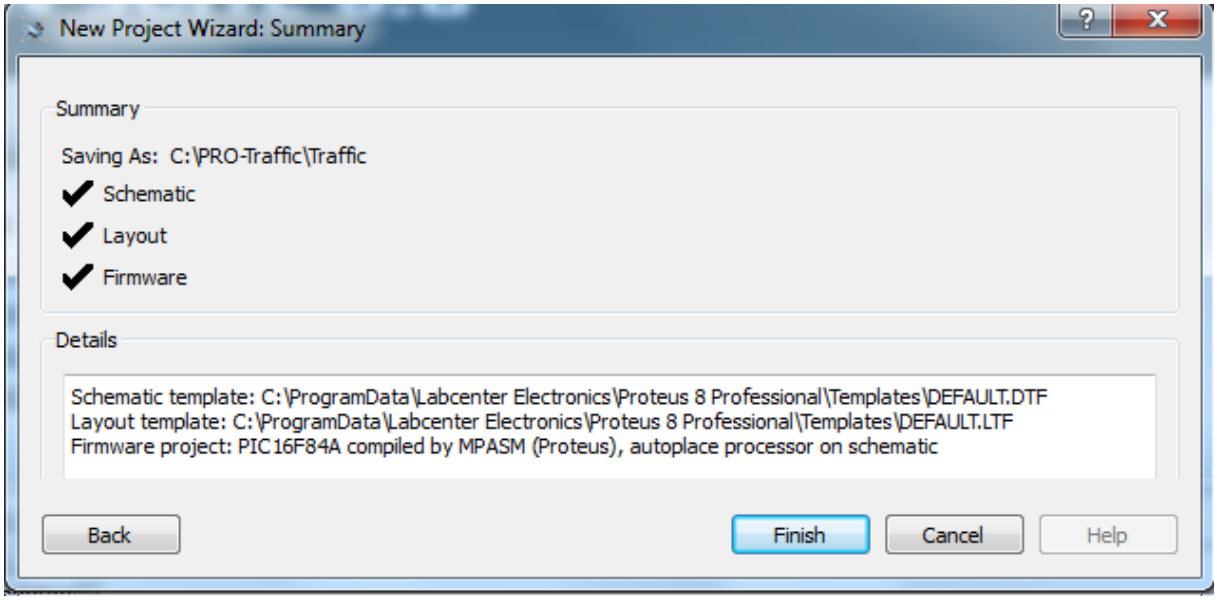


Рис.8.5. Итоговое окно создания проекта

8.2. Интерфейс программы ISIS

После завершения операций по созданию нового проекта, нажав кнопку  , запускаем программу моделирования электронных схем.

На экране программы ISIS (рис.8.6) расположены: окно редактирования 1, селектор объектов 2, окно обзора 3, главное меню 4, вкладка верхнего уровня 5, панель инструментов 6.

На вкладках главного меню 4 расположены разнообразные средства управления проектом и вызова команд, с которыми мы будем знакомиться по мере необходимости.

На вкладке 5 расположены, в частности, иконки включения приложений:

 - домашняя страница;

 - программа моделирования;

 - программа проектирования печатных плат;

 - объемная 3D визуализация;

 - просмотр печатных плат в формате Gerber;

 - проводник проекта отображает список всех элементов, которые использованы на текущей странице проекта;

 - список материалов, компонентов и стоимости;

 - исходный код программы;

 - обзор справочных данных.

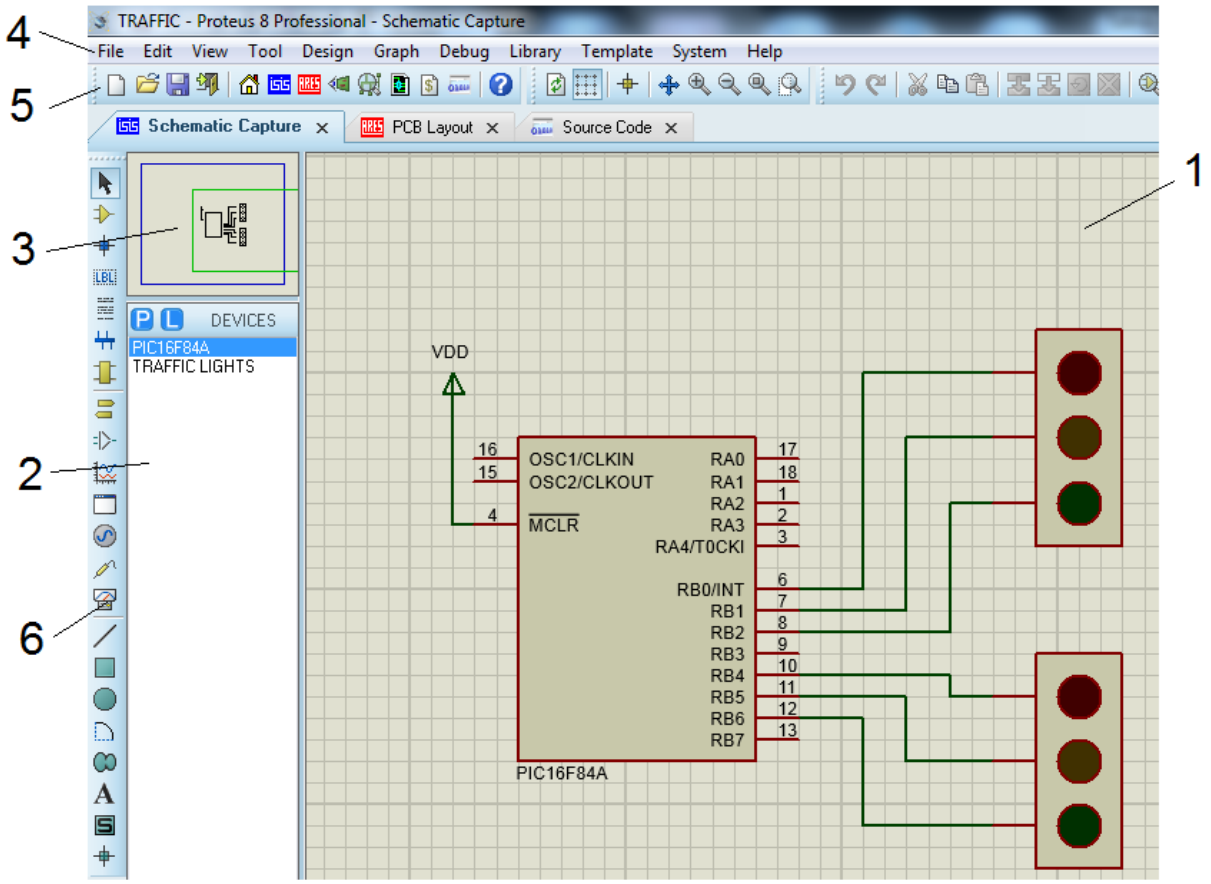


Рис.8.6. Экран программы ISIS

Панель инструментов 6 содержит четыре набора кнопок.

Первый набор - Главные режимы

 *Режим выбора* позволяет щелчком левой кнопкой мыши выделять компоненты и структурные блоки и, нажав правую кнопку мыши, выполнять редактирование, копирование,

перемещение, повороты, масштабирование и т.п. Поместив указатель в окно обзора и нажав левую кнопку мыши, можно позиционировать схему в окне редактирования.

Компоненты. Нажав на эту кнопку, в окне селектор объектов получаем список компонентов схемы. Щелчок левой кнопкой на названии компонента отображает его в окне обзора, а последующий двойной щелчок в окне редактирования переносит компонент на рабочее поле. Компонент можно выделить и перемещать. Нажав правую кнопку мыши, открываем вкладку редактирования, которая позволяет перемещать компонент, вращать, копировать, удалять, а также изменять электрические параметры и обозначение компонента. Двойной щелчок правой кнопкой удаляет компонент из окна редактирования.

Если в селекторе объектов нажать кнопку *P*, откроется окно браузера библиотек Pick Devices (рис.8.7). Для выбора микроконтроллера введем ключевое слово PIC16, выберем категорию Microprocessors ICs и модель PIC16F84A. В окнах просмотра мы увидим расположение выводов и корпус микроконтроллера. Нажав Ok, мы переносим компонент в список окна селектора устройств. После этого компонент можно перенести в окно редактирования.

Библиотеки Proteus содержат каталоги всевозможных компонентов, которые сгруппированы по категориям, производителям, наименованиям в группе. Каждый компонент имеет графическое изображение и установочный чертеж.

Точка соединения позволяет соединить любое пересечение проводников на схеме.

Разметка проводника позволяет присвоить наименование любому проводнику. Проводники, поименованные одинаково, будут условно соединены.

Вставка текста в любое место окна редактирования. В окне *Edit Script Block* можно менять ориентацию и центровку текста, импортировать готовый текст, редактировать стиль текста.

Шина. При нажатой кнопке можно выполнять соединения на схеме шинами, содержащими несколько проводников. Первый щелчок левой кнопкой мыши определяет начало шины. Следующие одиночные щелчки делаем в точках поворотов. Двойной щелчок завершает рисование шины. Причем повторный двойной щелчок создаст копию шины в окне редактирования.

Субсхема позволяет создавать функциональные блоки с выводами – соединителями. Созданному модулю через изменение свойств можно присвоить имя. Выделив окантовку модуля и нажав правую кнопку мыши, можно добавить порты к модулю, перенести модуль на отдельный лист.

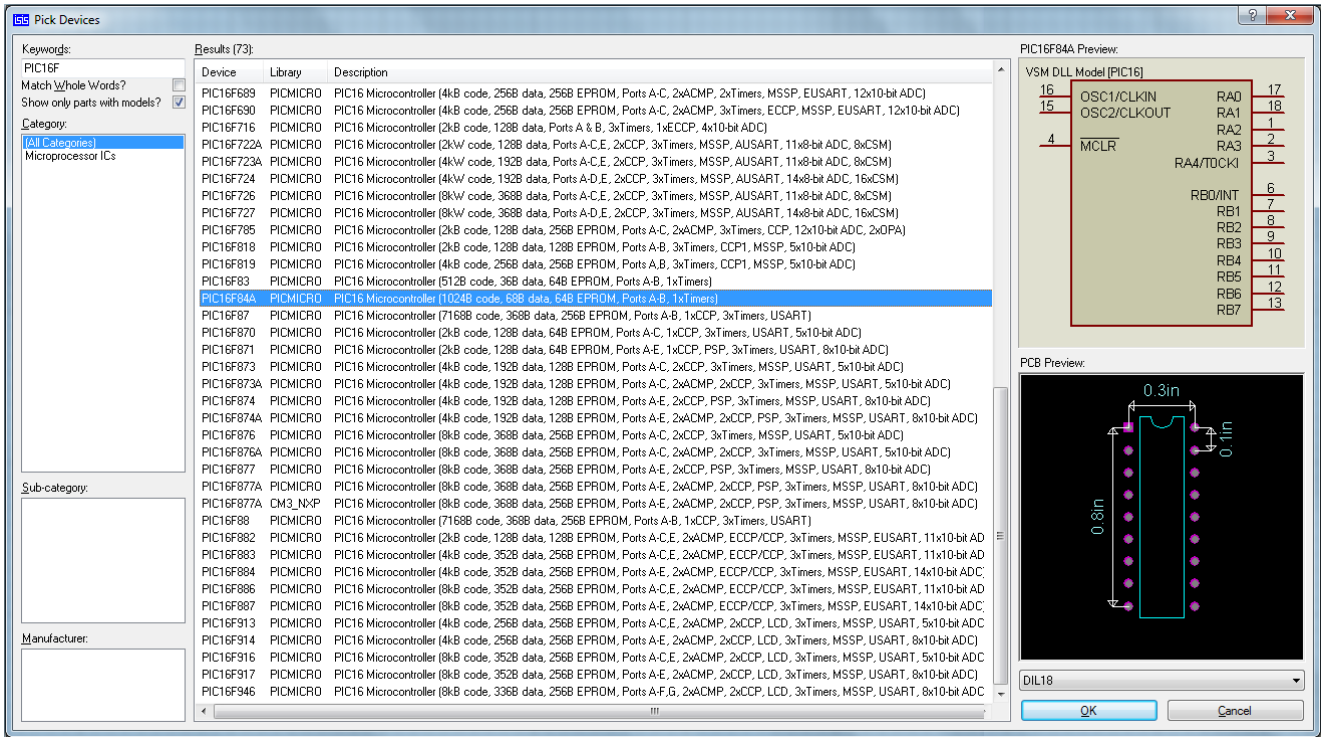


Рис.8.7. Окно выбора библиотечных компонентов Pick Devices

Второй набор - Устройства

Терминалы. Эта кнопка открывает в селекторе устройств перечень терминалов: обычный контакт DEFAULT, вход INPUT, выход OUTPUT, двунаправленный терминал BIDIR, терминал питания POWER, терминал земли GROUND, терминал шины BUS. Если терминалы POWER и GROUND не поименованы, считается, что они подключены к глобальным для проекта питанию и земле.

Пины устройства. Кнопка добавляет выводы к модели устройства.

Диаграмма. Режим размещения дисплеев для отображения графиков при моделировании устройства. В окне редактирования можно получить экран дисплея, перемещая по диагонали карандаш при нажатой левой кнопке мыши. Возможны более 12 вариантов отображения: ANALOGUE, DIGITAL, TRANSFER, FOURIER и т.д.

Активное всплывающее окно. Выделяет область анимации схемы.

Генераторы. Кнопка включает режим выбора и расстановки генераторов (более 12 видов).

Пробники. Пробники напряжения и тока ставят на провода схемы (рис.8.8).

Виртуальные инструменты. При нажатии кнопки в селекторе объектов появляется список приборов: осциллограф, генератор сигналов, логический анализатор, вольтметры, амперметры и т.д. Приборы можно перенести в окно редактирования и подключить к модели устройства.

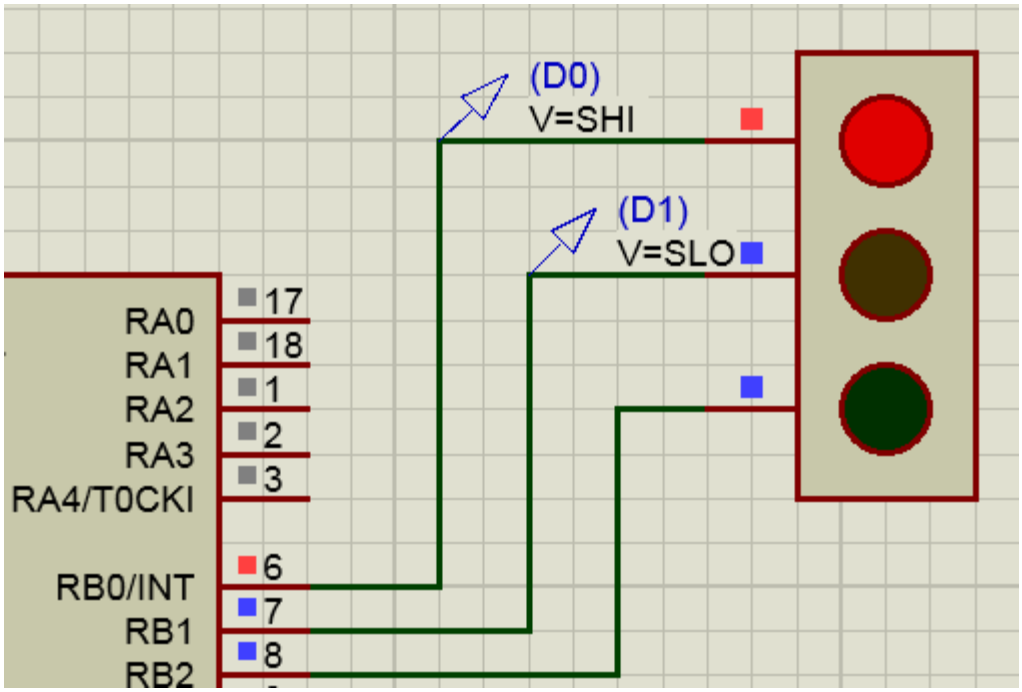


Рис.8.8. Пробники в схеме

Третий набора – Двумерная графика

Кнопки двумерной графики (рис.8.9) используются при создании собственных компонентов.

Рис.8.9. Кнопки двумерной грпфики

Четвертый набор – Ориентация

Эти кнопки (рис.8.10) меняют ориентацию объектов в окне обзора.

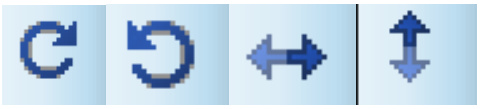


Рис.8.10. Кнопки ориентации объектов

8.3. Основы рисования схем

Подробно рассмотрим, как нарисовать схему (рис.8.8).

Выбор и размещение компонентов схемы

В селекторе объектов набираем *P* и в окне Pick Devices набираем ключевое слово PIC16. Выбираем из каталога наш микроконтроллер PIC16F84A и нажимаем OK. Микроконтроллер отображается в окне селектора объектов и в окне предварительного просмотра. Левым щелчком мыши переносим микроконтроллер в нужное место окна редактирования и повторным левым щелчком закрепляем объект в этом месте.

Затем снова в окне Pick Devices делаем поиск по ключевому слову *Traffic* и загружаем в селектор объектов светофор. Переносим два светофора в окно редактирования.

Если требуется переместить объект или изменить ориентацию, выделяем его правым щелчком и в контекстном меню выбираем нужное действие (перемещение, редактирование свойств, вращение и т.п.). Позиционирование схемы выполняют перемещением рамки в окне обзора.

Перед началом соединений полезно бывает изменить масштаб в конкретной области. Чтобы сделать это, можно, удерживая нажатой клавишу SHIFT, перетащить рамку в окне обзора на ту часть схемы, которую надо увеличить. Кроме того, можно использовать среднее колесо мыши или F6 и F7 на клавиатуре для увеличения и уменьшения масштаба вокруг текущего положения курсора мыши. Вернуться к полноэкранному просмотру можно в любое время с помощью клавиши F8. Наконец, иконки меню (рис.8.11) в верхней части вкладки 5 обеспечивают доступ к тем же функциям.

Рис.8.11. Иконки изменения масштабов

8.4. Соединение компонентов

Для соединения одиночными проводниками подводим указатель мыши к контакту вывода 6 (RBO/INT). На контакте появляется красный квадратик, а курсор превращается сначала в зеленый карандаш  , а после начала движения становится

белым . На поворотах проводника надо щелкнуть левой кнопкой. В конце соединения после появления красного квадратика надо еще раз щелкнуть левой кнопкой. Если в меню Tool включить режим Wire Autorouter, то провод будет прокладываться под прямыми углами и обходить все занятые компонентами или скриптами места.

Для подключения верхнего светофора используем шину (рис.8.12). Для этого в панели инструментов 6 нажимаем кнопку *Buses Mode*, после щелчка левой кнопкой мыши начинаем проводку шины из нужного места, а затем двойным левым щелчком устанавливаем шину в окне редактирования. При необходимости в режиме выбора выделяем шину, меняем ориентацию, длину и положение. Затем синим карандашом соединяем выводы микроконтроллера и верхнего светофора с центром шины, на котором появляются красные квадратики. И, наконец, в режиме разметки проводников обозначаем концы проводников одинаковыми метками.

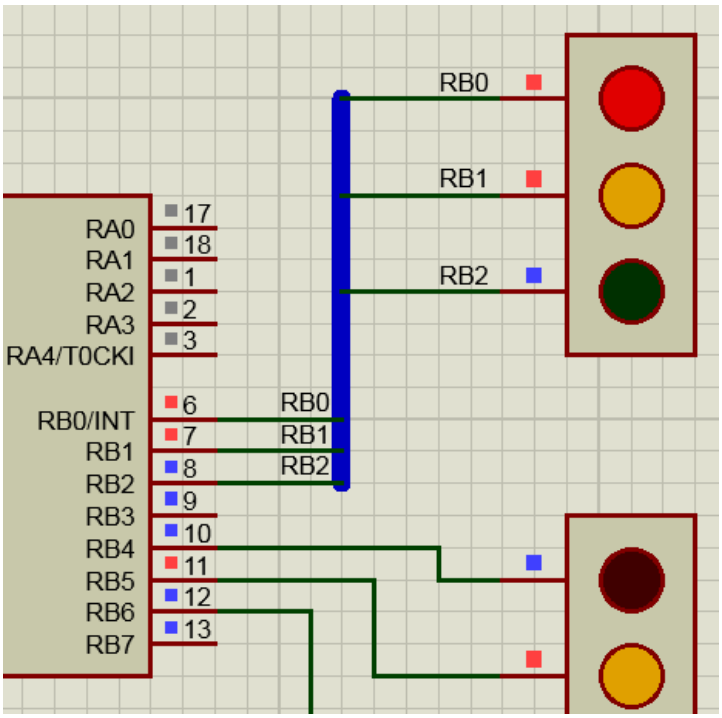


Рис.8.12. Соединение компонентов

На рис.8.13 показан второй способ соединения шиной. Мы используем два терминала Bus, обозначенные одинаково A[0..2], где 0 – начальный разряд, далее следуют две точки, 2 – конечный разряд. Терминалы с одинаковым обозначением считаются соединенными шиной. Проводники, подсоединенные к терминалам, обозначены A0, A1, A2.

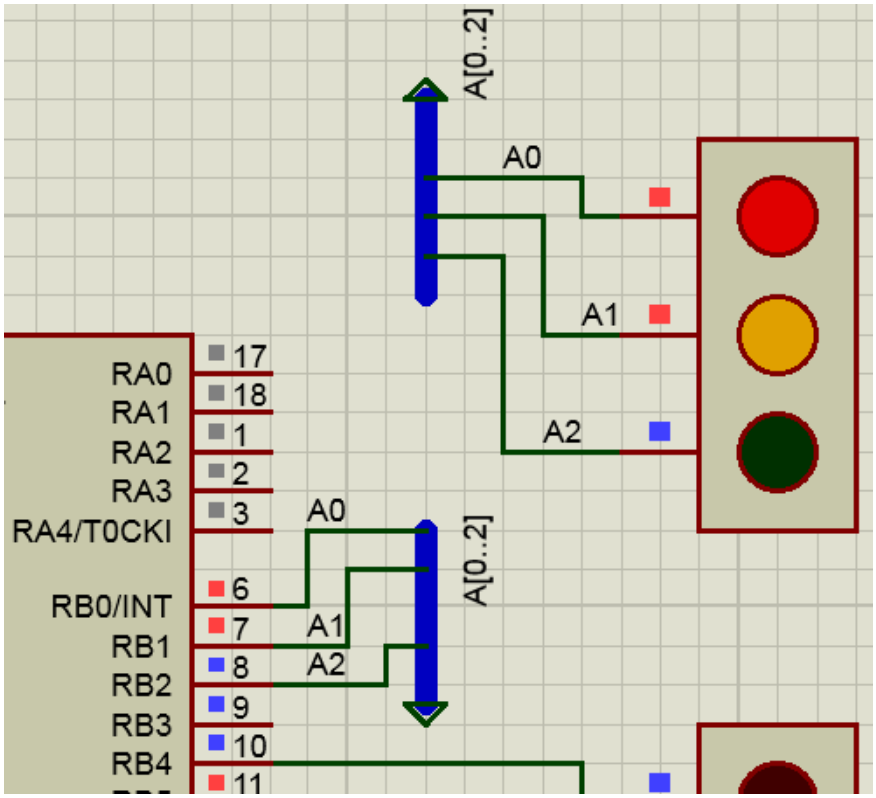


Рис.8.13. Соединение шиной с терминалами

В схеме (рис.8.6) установим еще терминал питания Power с меткой VDD и подключим его к выводу -MCLR микроконтроллера. Остальные выводы питания и земли на схеме микроконтроллера отсутствуют. Нужные режимы на них устанавливаются по умолчанию. Тактовую частоту устанавливают в редакторе свойств микроконтроллера.

8.5. Написание программы

После завершения рисования схемы перейдем к написанию программы. Для этого открываем вкладку исходный код  и получаем шаблон программы для микроконтроллера PIC16F84A.

В этот шаблон можно записать текст своей программы или скопировать текст готовой программы. Мы скопируем текст программы *Traffic* в ассемблере из раздела *Help*. Если отдельные блоки в программе будут дублироваться, надо исправить это. Текст программы показан на листинге 8.1.

Листинг 8.1

```
;=================================================
; DEFINITIONS
;=================================================
#include p16f84a.inc     ; Include register
                          ;definition file
;=================================================
; VARIABLES
;=================================================
 cblock 0x10   ; Temporary storage
   state
   l1,l2
   endc
;=================================================
; RESET and INTERRUPT VECTORS
;================================== ============
      ; Reset Vector
RST   code  0x00
      goto  setports
;================================================
; CODE SEGMENT
;================================================
PGM   code
setports
      clrw
      movwf PORTA
      movwf PORTB
      bsf STATUS,RP0
      clrw
      movwf TRISB
      bcf STATUS,RP0
initialise
      clrw
      movwf PORTB
loop
```

```
      call getmask
      movwf PORTB
      incf state,W
      andlw 0x03
      movwf state
      call wait
      goto loop

; Function to return bitmask for output port for
;current state.

; Top nibble controls one set of lights and bottom
;nibble the other.
getmask
      movf state, W
      addwf PCL,F
      retlw 0x41 ; Green and Red.
      retlw 0x23 ; Amber and Red/Green.
      retlw 0x14 ; Red and Green.
      retlw 0x32 ; Red/Amber and Green.
; Simple delay routine.
wait
      movlw 0xFF
      movwf l1
w1    decfsz l1
      goto w1
     return
wait2 clrf l2
w2    decfsz l2
      goto w2
      return
;=========================================
      END
```

8.6. Компиляция программы

Для получения исполняемого файла выполним компиляцию программы. Для этого в главном меню выбираем *Build – Build Project.* Если в программе нет синтактических и орфографических ошибок и все использованные функции

понятны компилятору, в окне Output получим сообщение об успешной компиляции (рис.8.14).

```
Device Database Version 1.14
Copyright (c) 1998-2011 Microchip Technology Inc.
Errors    : 0

Message[302] ..\MAIN.ASM 62 : Register in operand not in bank 0. Ensure that bank bits are correct.
Message[305] ..\MAIN.ASM 114 : Using default destination of 1 (file).
Message[305] ..\MAIN.ASM 122 : Using default destination of 1 (file).
Compiled successfully.
```

Рис. 8.14. Сообщение об успешной компиляции

В окне Source Code появляется вкладка с файлом описания микроконтроллера PIC16F84.INC, в котором определены адреса регистров специального назначения, управляющие биты и т.п.

8.7. Испытание программы в модели

Так как компиляция выполнена успешно, можно проверить работу программы на модели. Для этого в окне ISIS выделяем микроконторрллер, *выбираем Edit Properties – Program Files* и из папки PIC16F84A, которая расположена в папке проекта, открываем файл *Debug cof* (Рис.8.15). Это выходной исполняемый файл в формате *COFF* (*Common Object File Format*) используется при отладке проектов.

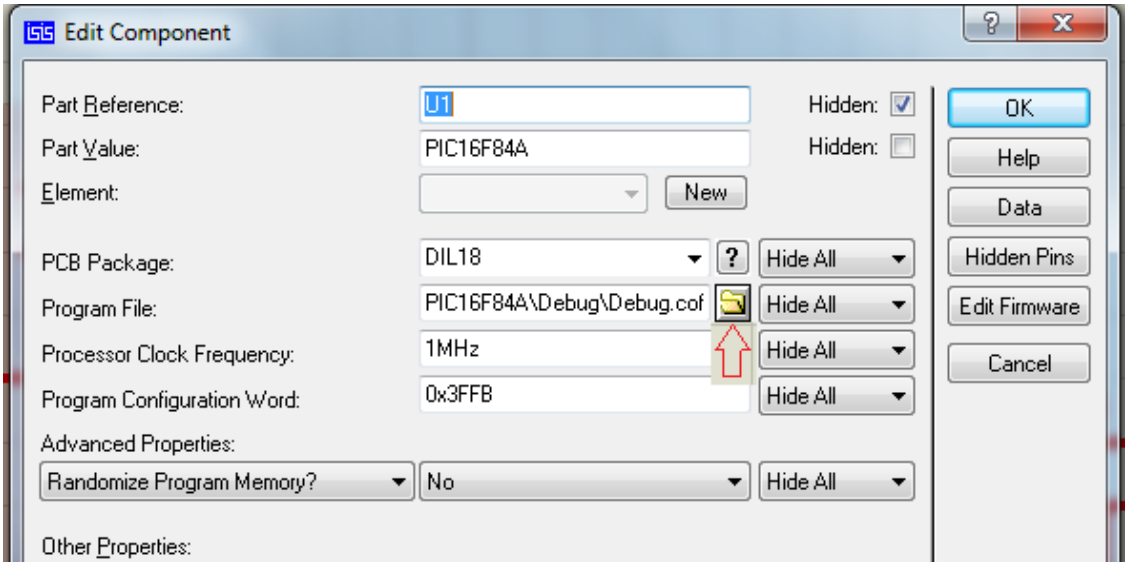


Рис.8.15. Загрузка файла программы в модель

Для моделирования на панели управления анимацией нажимаем кнопку *Play*  и видим переключения светофоров. Время от начала моделирования и процент загрузки микроконтроллера отображается в нижней строке состояния, а на схеме мы видим текущее состояние цепи.

8.8. Отладка программы

Для запуска отладки надо нажать кнопку шаговой анимации  или кнопку *Pause*  . В выпадающем меню *Debug* выбрать код программы *PIC CPU Source Code*. В области редактирования интегрированной среды IDE мы увидим окно с текстом программы и линией, наложенной на текущую команду (рис.8.16). При первом запуске это будет стартовая команда.

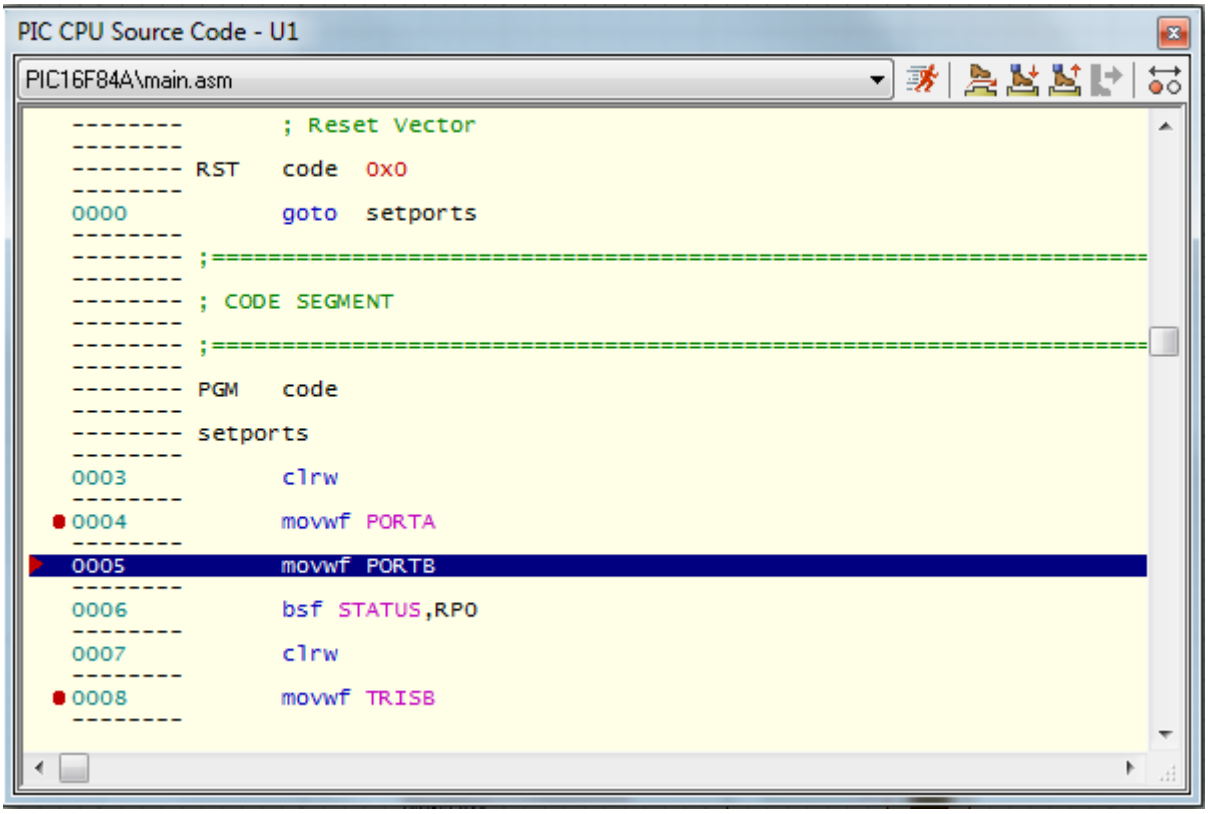


Рис.8.16. Окно исполняемого кода

Для пошагового прохождения программы применяют команду *Step Into Source Line*  из панели управления в окне с кодом программы или клавишу F11. Для выхода из цикла

применяют команду *Step Out Source Line*  или сочетание клавиш Ctrl+F11. Если текущая команда (Call) вызывает подпрограмму, обойти ее выполнение можно командой *Step Over Source Line*  или клавишей F10.

Можно перейти на нужную строку программы, выделив ее линией наложения и выполнив команду *Run To Source Line*  или сочетанием клавиш Ctrl+F10.

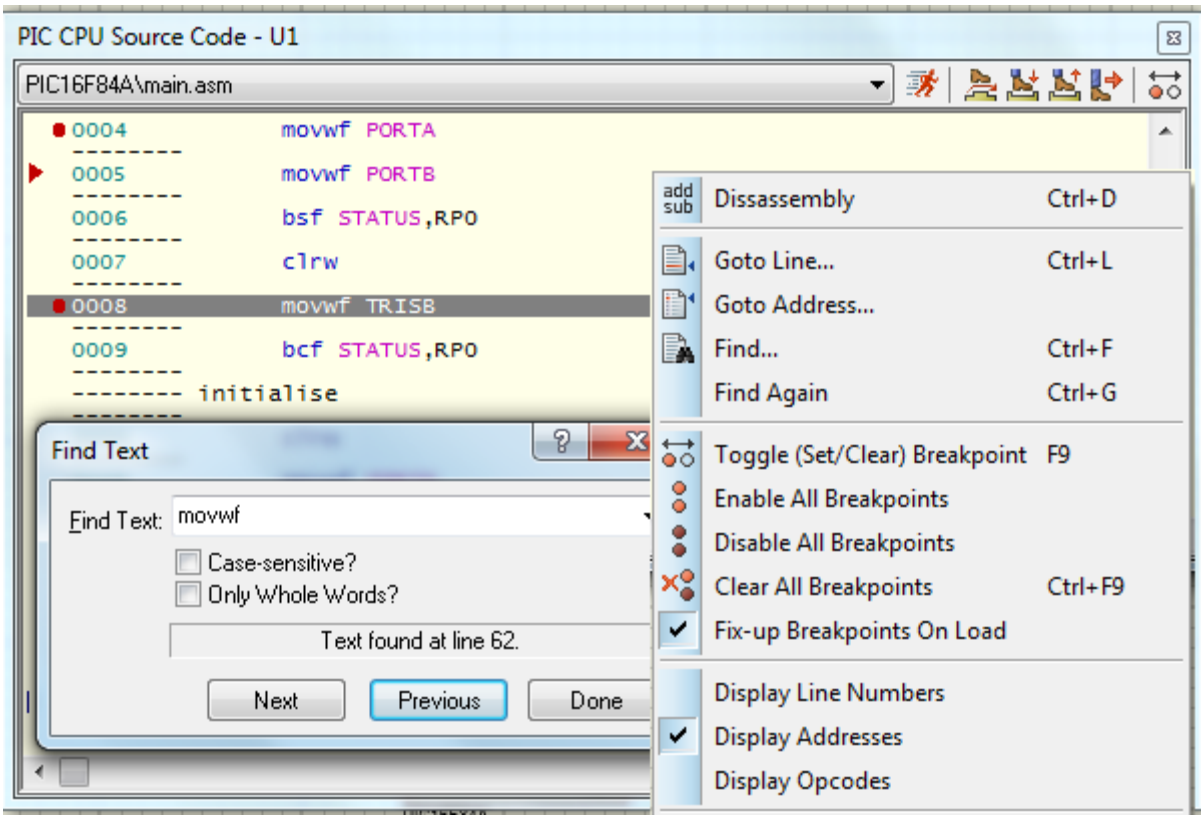


Рис.8.17. Контекстное меню окна исполняемого кода

Точки останова на линии команды можно задать разными способами.

Можно выделить команду и двойным щелчком левой кнопкой мыши установить *Breakpoint*, вторым двойным левым щелчком отключить точку останова, третьим двойным щелчком очистить строку от точки останова.

Это же можно сделать кнопкой *Toggle Breakpoint*  в окне программы или, вызвав правой кнопкой мыши контекстное меню (ри.8.17). В контекстном меню можно перейти к нужному адресу программы или к нужной строке текста программы. Можно также выполнить поиск текста.

8.9. Окна отладки

В меню *Debug* (рис.8.18) мы можем открыть несколько полезных для отладки окон.

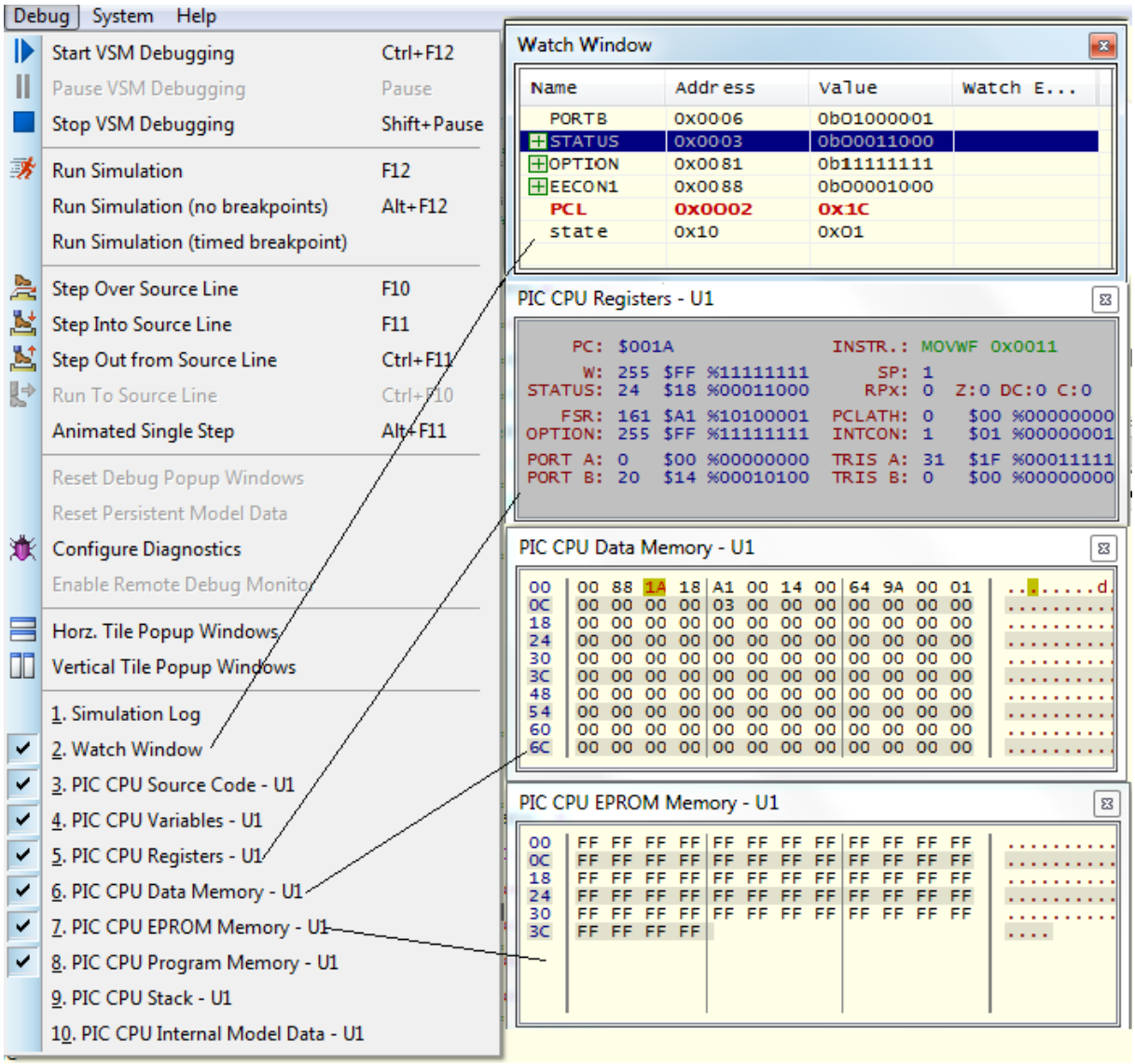


Рис.8.18. Окна отладки из меню *Debug*

В окне *Watch Window* щелчком правой кнопкой мыши открываем контекстное меню (рис.8.19), в котором можно

выбрать и двойным щелчком левой кнопкой мыши добавить для наблюдения элемент по заданному имени (регистры специального назначения) или по адресу (регистры памяти, переменные, константы и т.п.).

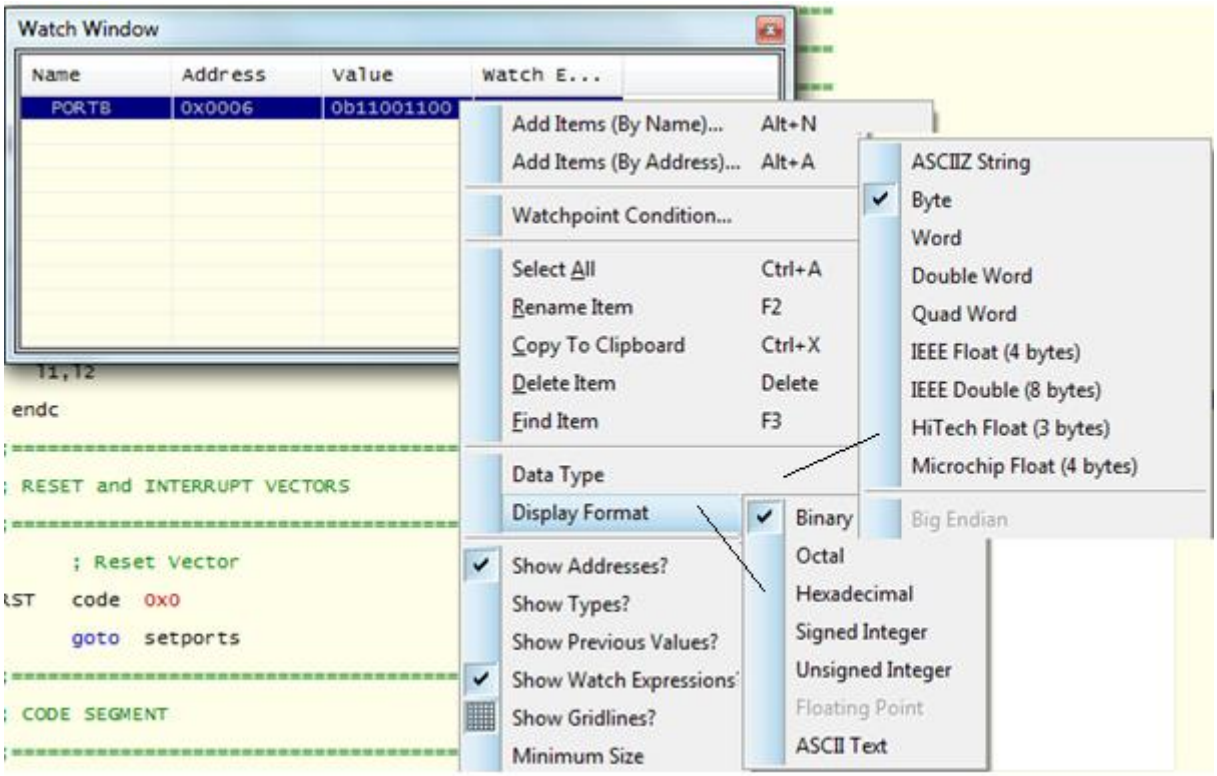


Рис.8.19. Контекстное меню *Watch Window*

Можно переименовать, скопировать в буфер обмена, удалить элемент.

Вкладка *Data Type* позволяет выбрать тип представления данных, а вкладка *Display Format* – формат представления чисел.

Запустим отладку и увидим изменение текущих значений элементов в окне наблюдения.

Выбирем в контекстном меню пункт *Watchpoint Condition* (рис.8.20). Мы можем запретить все остановки по условиям, приостановить симуляцию, если выполняется любое условие (*Suspend the simulation if Any expressions is true*), остановить симулятор, когда все условия соблюдены (*Stop the simulation only when all expression is true*). Остановим симуляцию, когда в верхнем светофоре будут гореть красный и желтый свет. Порт B

при этом будет содержать единицы в нулевом и первом битах. Выполним операцию AND с числом 0x03 и установим условие равенства результата числу 0x03. В режиме отладки выполним Run. Симуляция остановится на команде с адресом 0x000E. При этом PORTB содержит значение 0b00100011

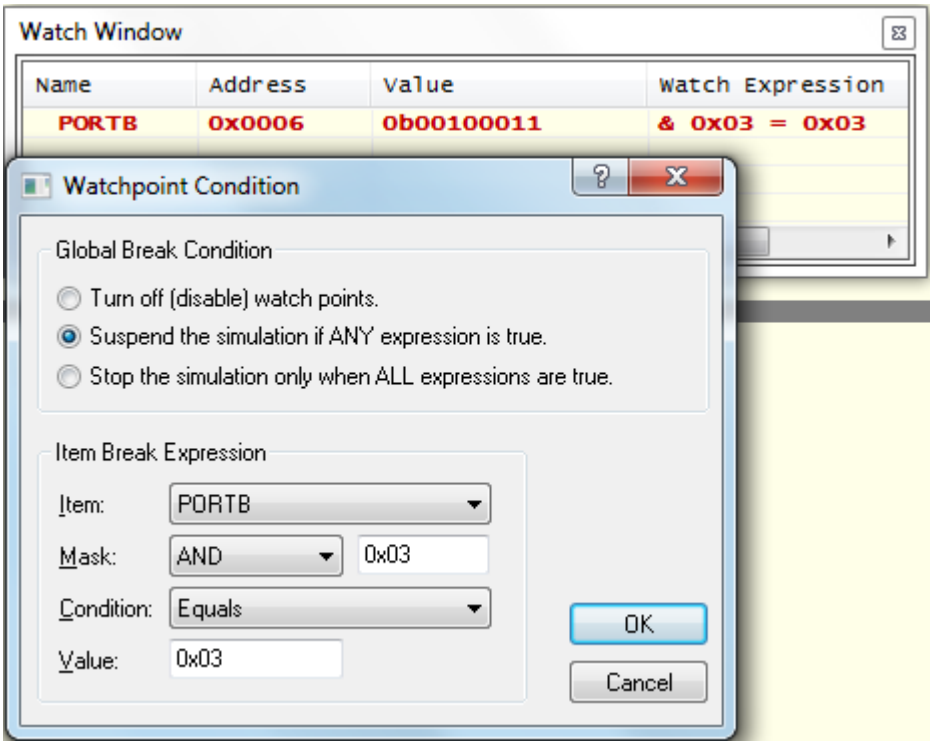


Рис.8.20. Окно условий остановок

В окне регистров специального назначения *PIC CPU Registers* (рис.8.18) отображается адрес и инструкция исполняемой команды и текущее состояние регистров специального назначения.

В окне памяти данных *PIC CPU Data Memory* отображается текущее состояние ОЗУ.

В окне *PIC CPU EPROM Memory* отображается содержание РПЗУ.

8.10. Окно диагностических сообщений

В окне *Simulation Log* в процессе симуляции формируются диагностические сообщения (рис.8.21). Состав и уровень сообщений устанавливают в окне *Configure Diagnostic*, которое открывается в меню *Debug*.

Диагностические сообщения формируются для самого симулятора SYSTEM [ISIS/PROSPICE] и для микроконтроллера PIC16F84A [U1]. Степень детализации сообщений устанавливают в окне конфигурации диагностики. Надо выделить категорию сообщений и выбрать нужный уровень: -отключить категорию (*Disable*); - только предупреждение (*Warning only*); - полная трассировка (*Full Trace*); - отладка (*Debug*).

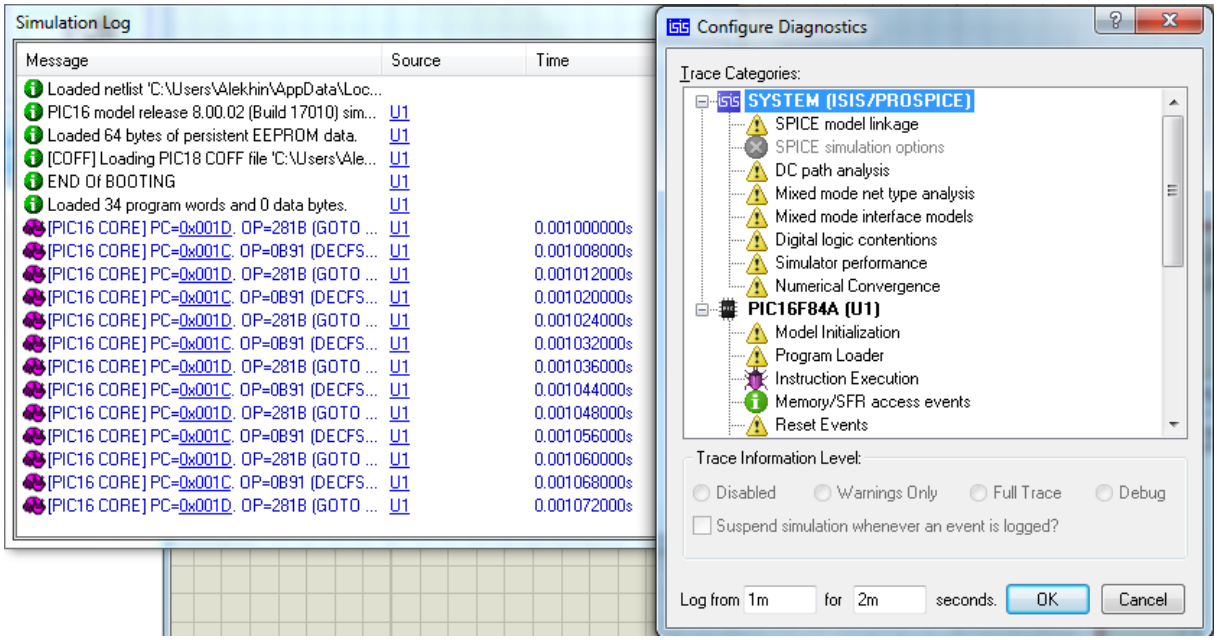


Рис.8.21. Окна диагностических сообщений и конфигурации диагностики

Время начала и конца диагностики устанавливается в окошках нижней строки в секундах или миллисекундах (например, *from 1m - for 2m*). В окне диагностики, начиная с 1 мс, появляется перечень адресов и коды выполняемых команд. Если в режиме паузы щелкнуть по подчеркнутому значению программного счетчика PC=0x001C, то в открывшемся окне *PIC CPU Source Code* будет подсвечена линия с этим адресом.

После окончания отладки опции *Reset Debug Popup Windows* и *Reset Persistent Model Data* в окне отладки (рис.8.18) позволяют сбросить в исходное состояние всплывающие окна отладки и

постоянные моделей данных. Это может потребоваться для продолжения обычной симуляции.

8.11. Добавление и удаление файлов в проекте

В проект можно добавить готовые файлы из других проектов или написанные в текстовом редакторе. Для этого можно открыть вкладку *Source* или щелкнуть правой кнопкой в окне *Projects*, а затем выбрать *Add New Files* или *Add Files* (рис.8.22).

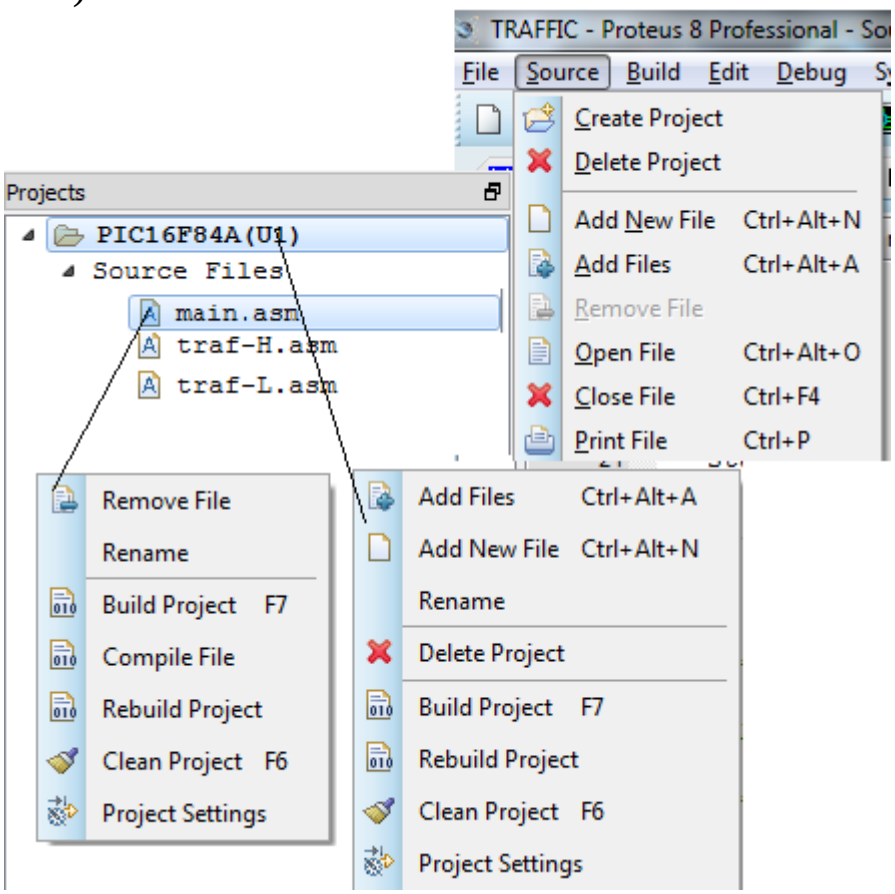


Рис.8.22. Добавление файлов в проект и действия с файлами

Выделенный в окне *Project*s файл на показанных вкладках можно переименовать, удалить из проекта, компилировать.

На вкладке *Source* есть опция *Create Project*, которая позволяет в существующий схемотехнический проект подключить новую программу микроконтроллера. Изменим исходную программу для светофоров так, чтобы в программе *traf-H* работал только верхний светофор, а в программе *traf-L*

работал только нижний светофор. Для верхнего светофора в программе *main.asm* перед загрузкой регистра TRISB вместо команды `clrf` запишем `movlw 0xF8`, а для нижнего светофора запишем `movlw 0x8F`. Сохраним измененные файлы в папке проекта с именами *traf-H.asm* и *traf-L.asm*.

Далее на вкладке *Source* выбираем опцию *Create Project* и дважды заполняем окно нового программного проекта (рис.8.23).

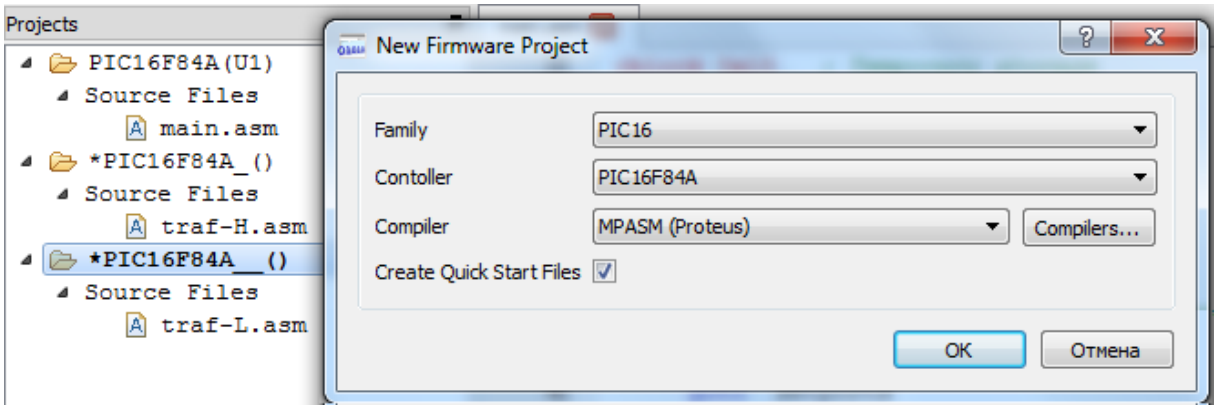


Рис.8.23. Создание новых программных проектов

Во второй проект добавляем файл *traf-H.asm*, выделяем название файла и в контекстном меню выполняем *Build Project или Rebuild Project*. Загружаем в микроконтроллер файл *debug.cof* и убеждаемся, что работает только верхний светофор.

Повторим испытание для проекта с файлом *traf-L.asm*. Загрузим последний по времени файл *debug.cof*. Для этого в окне *Select File Name* надо перейти выше на два уровня и выбрать папку PIC16F84A с соответствующим временем создания файла.

8.12. Анализ выходных сигналов в цифровом анализаторе

Сигналы на выходе микроконтроллера можно наблюдать в цифровом анализаторе. Для этого на панели инструментов выбираем *Graf*  и в селекторе объектов выбираем *Digital*. Открываем в окне редактирования экран *Digital Analysis* (рис.8.24).

На проводниках A0…A6 установим пробники напряжения и обозначим их также, как проводники. Затем выделяем правой

кнопкой пробник A0 и перетаскиваем его на экран цифрового анализатора. Появляется горизонтальная ось с меткой пробника. Повторяем это для всех пробников.

Выделяем цифровой анализатор и в его свойствах устанавливаем *Start time* 0 и *Stop time* 20m. Включаем симуляцию и, немного выждав, нажимаем стоп. Затем правой кнопкой выделяем анализатор, в контекстном меню выбираем *Clear Graf Data* и очищаем экран от старых данных. Затем повторно выделяем анализатор правой кнопкой и нажимаем клавишу пробела. На экране появляются новые графики выходных сигналов микроконтроллера. В контекстном меню опцией *Maximize (Show Windows)* цифровой анализатор можно развернуть на весь экран.

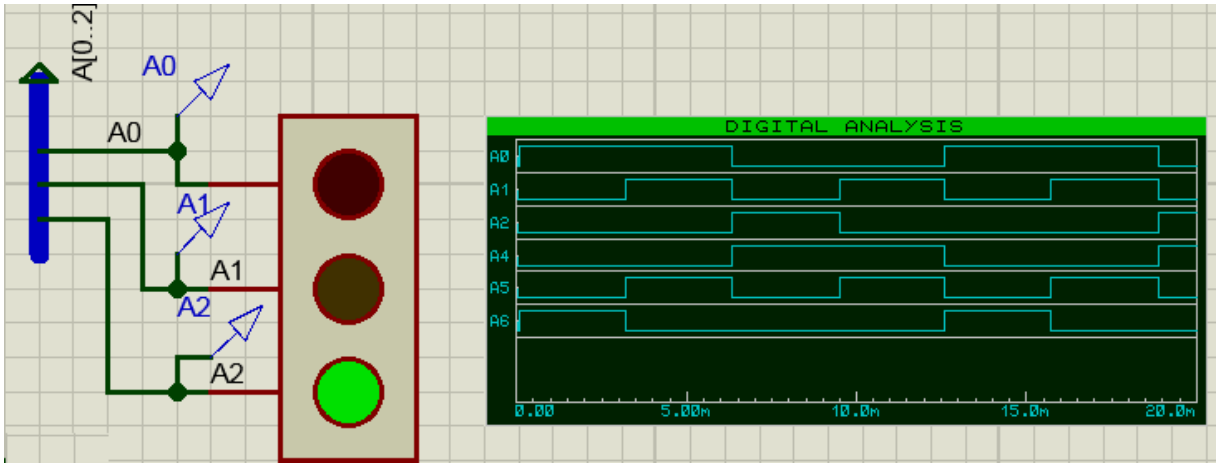
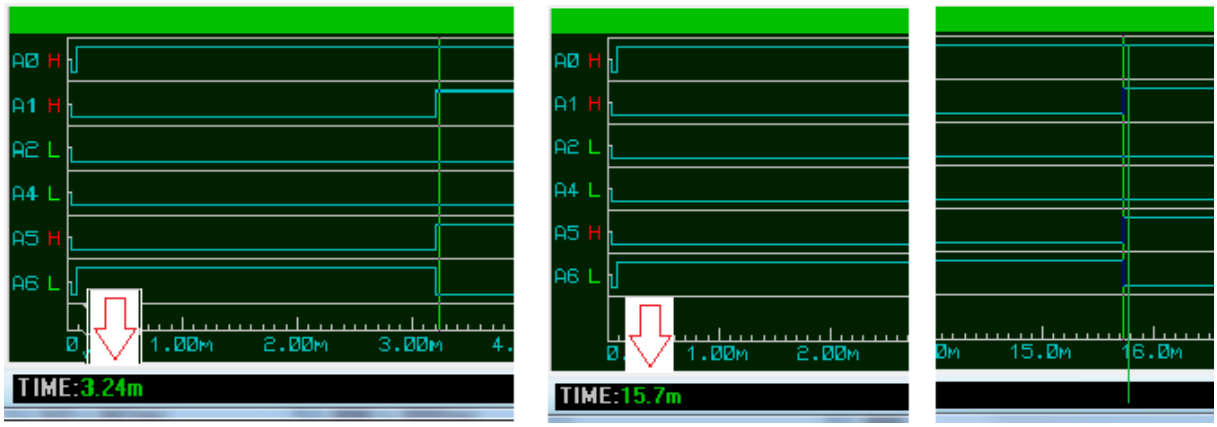


Рис.8.24. Наблюдение выходных сигналов в цифровом анализаторе

По временным диаграммам цифрового анализатора определим период переключения светофоров. Сначала установим курсор в первой позиции, где включены красный и желтый свет верхнего светофора (A0=H, A1=H). Текущее время от пуска программы составляет 3,24 мс (рис.8.25а). Второй раз такое событие наступает в момент 15,7мс (рис.8.25б). Следовательно, период переключения для установленной частоты микроконтроллера 1 МГц составляет ~12,5 мс.



а) б)

Рис.8.25. Определение периода переключений в цифровом анализаторе

Теперь определим период переключения в отладке, используя *Watch Window*. Установим условия остановки, которые показаны на рис.8.20. Делаем пуск симуляции  . Первая остановка по заданному условию происходит через 3,2 мс (рис.8.26). . Командой *Step Out Source Line*  снова запускаем симуляцию. Вторая остановка происходит в момент 15,7 мс. Мы снова получили значение периода переключений светофора 12,2 мс.

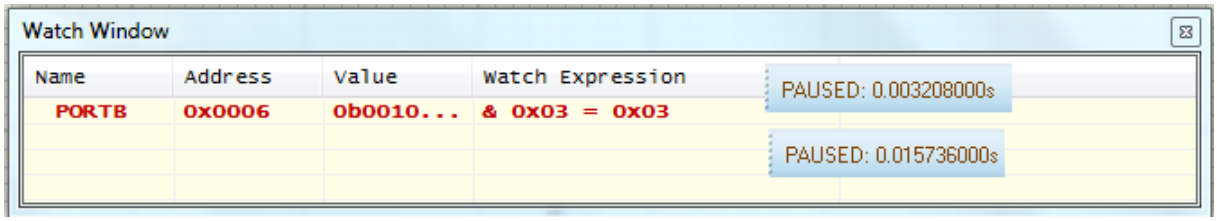


Рис.8.26. Определение периода переключений окне наблюдения

Контрольные вопросы и задания

1. Установите в своем компьютере программу Proteus 8.1 или откройте уже установленную программу. Выполните самостоятельно создание проекта Traffic, соберите схему (рис.8.6), подключите пробники и цифровой анализатор, введите программу (листинг 8.1), выполните компиляцию и испытания модели в программе Proteus ISIS.

2. Какие окна, вкладки и панели расположены на экране программы ISIS и каково их назначение ?

3. Для чего служит окно обзора и как выполняется масштабирование схемы ?

4. Как следует выбирать нужные компоненты для схемы ?

5. Как осуществляют рисование и подключение шин на схеме ?

6. Что такое терминалы и как их используют ?

7. Какие виды диаграмм можно использовать и как подключить диаграмму к схеме ?

8. Как установить в схеме пробники и виртуальные приборы ?

9. Как можно ввести в проект программу ?

10. Как загрузить в модель исполняемый файл программы ?

11. Расскажите о командах и средствах отладки программы.

12. Какие окна используют при отладке и как их открывают ?

13. Как можно добавить в проект новые файлы ?

14. Как загрузить в модель один из нескольких файлов проекта ?

Глава 9. СОПРЯЖЕНИЕ МИКРОКОНТРОЛЛЕРОВ С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ В СРЕДАХ mikroC и и Proteus

9.1. Ввод текста с клавиатуры на ЖК-дисплей

Описание аппаратных и программных средств

В этом разделе мы познакомимся с подключением к микроконтроллеру клавиатуры и жидкокристаллического дисплея и спроектируем простую систему безопасности с проверкой цифрового пароля. Программирование мы будем проводить в среде mikroC, а моделирование в Proteus, используя примеры из библиотек этих программ. Сначала выберем подходящие компоненты, которые есть в библиотеках наших программных сред.

В качестве ЖК-дисплея мы используем LM016L с размером 2x16 (32 символов), который имеет тот же контроллер HD44780, что и изученный нами ЖК- дисплей Displaytech 161A. Система команд у этих дисплеев одинаковая.

На рис. 9.1 показана схема моделирования системы безопасности.

Принципы управления ЖК-дисплеем мы рассмотрели в главе 5. Теперь нашей задачей будет изучение работы клавиатуры и составление программы с использованием библиотечных функций из mikroC.

Proteus предлагает три варианта клавиатуры: два варианта интерактивной матричной клавиатуры для калькулятора и интерактивную матричную клавиатуру для телефона (KEYPAD PHONE). Для ввода цифрового пароля телефонная клавиатура вполне подходит.

Рассмотрим, как работает клавиатура.

Клавиатура имеет 12 клавиш, установленных в три столбца и 4 ряда. Каждый столбец и ряд подключены к микроконтроллеру отдельными проводниками. Нажатие на

клавишу соединяет пересекающиеся под клавишей ряд и столбец. Столбцы C1, C2, C3 подключены к входным выводам PORTB <RB0:RB2>. Ряды R1, R2, R3, R4 подключены к выходным выводам порта <RB4:RB7>. В начальный момент логическая «1» подается на первый ряд R1. Остальные ряды имеют логический «0».

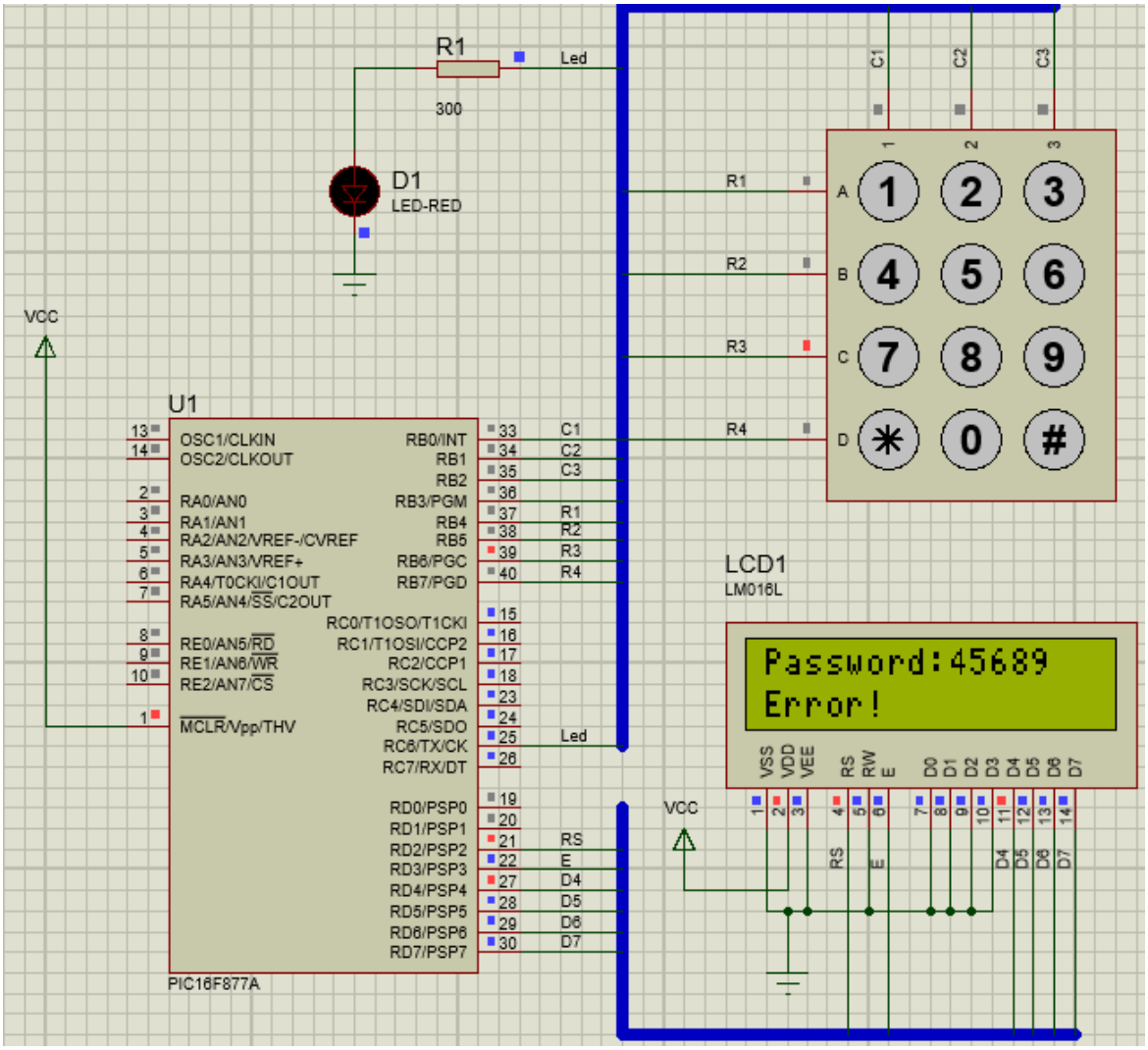


Рис.9.1. Схема модели системы безопасности

Во время подключения «1» к первому ряду микроконтроллер опрашивает (сканирует) входы порта, поключенные к столбцам клавиатуры C1:C3. Если ни на одном из столбцов нет сигнала «1», значит клавиши первого ряда не нажаты. Логическая «1» подается на второй ряд и сканирование столбцов повторяется. Процесс сканирования повторяется, пока

на одном из столбцов не будет обнаружена «1». Соответствующие этому событию номер столбца и номер ряда определяют нажатую клавишу и введенный знак. Количество выводов микроконтроллера для соединения с клавиатурой должно равняться сумме числа рядов и столбцов. Ряды специфицируются как выходы микроконтроллера, а столбцы как входы.

Система безопасности должна работать следующим образом. Секретный пароль из 5 цифр хранится в программе микроконтроллера. На дисплее выведено слово «Password:». Пользователь вводит 5 произвольных цифр пароля и получает ответ на дисплее: «OK!» или «ERROR!». После неудачи выводится число попыток: «TIMES: 1». После пяти попыток ввод прекращается.

Библиотека для работы с клавиатурой

MikroC PRO for PIC предоставляет библиотеку для работы с клавиатурой 4x4. Подпрограммы библиотеки также могут быть использованы для клавиатуры 4x1, 4x2, 4x3. Подключение клавиатуры 4x3 показано на рис.9.1.

Во всех проектах, использующих библиотеку клавиатуры, должна быть определена внешняя переменная `extern sfr char keypadPort` и установлен порт, к которому подключена клавиатура, например:

```
// Keypad module connections
char  keypadPort at PORTB;
// End Keypad module connections
```

Подпрограмма `Keypad_Init`

Прототип	`void Keypad_Init(void);`
Результат	Нет
Описание	Инициализирует порт для работы с клавиатурой.
Требования	Глобальная переменная `KeypadPort` должна быть определена перед использованием этой подпрограммы

Пример	`// Keypad module connections`
	`char keypadPort at PORTD;`
	`// End of keypad module connections`
	`...`
	`Keypad_Init();`

Прототип функции позволяет контролировать количество и соответствие типов аргументов и производить приведение типов. Имена аргументов в прототипе функции имеют ограниченную прототипом область действия. В функции `Keypad_Init` аргументов нет.

Подпрограмма `Keypad_Key_Press`

Прототип	`char Keypad_Key_Press(void);`
Результат	Код нажатой клавиши (1….16) типа `char`. Если клавиши не нажаты, возвращает 0.
Описание	Считывает значение нажатой клавиши, когда клавиша нажата.
Требования	Порт должен быть предварительно инициализирован для работы с клавиатурой подпрограммой `Keypad_Init()`.

Пример	`char kp;`
	`...`
	`kp = Keypad_Key_Press();`

Подпрограмма `Keypad_Key_Click`

Прототип	`char Keypad_Key_Click(void);`
Результат	Код щелчка клавиши (1….16) типа `char`. Если щелчка клавиши не было, возвращает 0.
Описание	Функция ждет, пока клавиша не будет нажата и отпущена. При отпускании функция возвращает (1…16), в зависимости от клавиши. Если одновременно нажаты несколько клавиш, функции будет ждать, пока все нажатые клавиши не будут освобождены. После этого функция возвращает код первой нажатой клавиши.
Требования	Порт должен быть предварительно

	инициализирован для работы с клавиатурой подпрограммой `Keypad_Init()`.

Пример	`char kp;`
	`...`
	`kp = Keypad_Key_Click();`

Библиотека mikroC для работы с ЖК-дисплеем

MikroC PRO for PIC предоставляет библиотеку для подключения ЖК-дисплеев, совместимых с контроллером HD44780 через 4-битный интерфейс.

Подпрограмма `Lcd_Init()`

В нашем примере дисплей мы будем подключать к порту D.

Прототип	`void Lcd_Init();`
Результат	Нет
Описание	Инициализирует LCD-модуль
Требования	Глобальные переменные: - `LCD_D7`: бит 7 данных - `LCD_D6`: бит 6 данных - `LCD_D5`: бит 5 данных - `LCD_D4`: бит 4 данных - `LCD_RS`: пин сигнала Register Select (data/instruction) - `LCD_EN`: пин сигнала Enable - `LCD_D7_Direction`: направление пина Data 7 - `LCD_D6_Direction`: направление пина Data 6 - `LCD_D5_Direction`: направление пина Data 5 - `LCD_D4_Direction`: направление пина Data 4 - `LCD_RS_Direction`: направление пина Register Select - `LCD_EN_Direction`: направление пина Enable signal Все переменные должны быть определены перед использованием этой функции.
Пример	// Подключение Lcd к порту B

	sbit LCD_RS at RD4_bit; sbit LCD_EN at RD5_bit; sbit LCD_D7 at RD3_bit; sbit LCD_D6 at RD2_bit; sbit LCD_D5 at RD1_bit; sbit LCD_D4 at RD0_bit; // Направление выводов sbit LCD_RS_Direction at TRISD4_bit; sbit LCD_EN_Direction at TRISD5_bit; sbit LCD_D7_Direction at TRISD3_bit; sbit LCD_D6_Direction at TRISD2_bit; sbit LCD_D5_Direction at TRISD1_bit; sbit LCD_D4_Direction at TRISD0_bit; ... Lcd_Init();

Подпрограмма `Lcd_Out`

Прототип	`void Lcd_Out(char row, char column, char *text);`
Результат	Нет
Описание	Печать текста на ЖК, начиная с указанной позиции. Как строковые переменные, так и литералы могут быть приняты в качестве текста. Параметры: - строка: исходное положение (номер строки); - колонка: исходное положение (номер столбца); - текст: текст, который будет написан.
Требования	Модуль ЖК необходимо инициализировать процедурой Lcd_Init.
Пример	`/* Write text "Hello!" on Lcd starting from row 1, column 3: */` `Lcd_Out(1, 3, "Hello!");`

Подпрограмма `Lcd_Out_CP`

Прототип	`void Lcd_Out_CP(char *text);`
Результат	Нет
Описание	Печать текста на ЖК-дисплее в текущей позиции

	курсора. Строковые переменные и литералы могут быть приняты в качестве текста. Параметры: - text: текст, который будет написан.
Требования	Модуль ЖК необходимо инициализировать процедурой `Lcd_Init`.
Пример	`/* Write text "Here!" at current cursor position:*/` `Lcd_Out_CP("Here!");`

Подпрограмма `Lcd_Chr`

Прототип	`void Lcd_Chr(char row, char column, char out_char);`
Результат	Нет
Описание	Печать символа на ЖК-дисплее в указанной позиции. Переменные и литералы могут быть приняты как символ. Параметры: - строка: номер строки позиции записи; - Колонка: номер столбца позиции записи; `Out_char`: символ для записи.
Требования	Модуль ЖК необходимо инициализировать процедурой Lcd_Init.
Пример	`// Write character "i" at row 2, column 3:` `Lcd_Chr(2, 3, 'i');`

Подпрограмма `Lcd_Chr_Cp`

Прототип	`void Lcd_Chr_CP(char out_char);`
Результат	Нет
Описание	Печать символа на ЖК-дисплее в текущей позиции курсора. Переменные и литералы могут быть приняты как символ. Параметры: `Out_char`: символ для записи.
Требования	Модуль ЖК необходимо инициализировать процедурой Lcd_Init.

Пример	`/* Write character "e" at current cursor position:*/` `Lcd_Chr_CP('e');`

Подпрограмма `Lcd_Cmd`

Прототип	`void Lcd_Cmd(char out_char);`
Результат	Нет
Описание	Посылает команду на ЖК-дисплей. Параметры: `Out_char`: команда для отправки. Примечание: Предопределенные константы (доступеные LCD –команды) могут быть переданы в функцию.
Требования	Модуль ЖК необходимо инициализировать процедурой `Lcd_Init`.
Пример	`// Clear Lcd display:` `Lcd_Cmd( LCD_CLEAR);`

Полный список доступных LCD – команд

`_LCD_FIRST_ROW`	Перемещает курсор в 1-ю строку
`_LCD_SECOND_ROW`	Перемещает курсор в 2-ю строку
`_LCD_THIRD_ROW`	Перемещает курсор в 3-ю строку
`_LCD_FOURTH_ROW`	Перемещает курсор в 4-ю строку
`_LCD_CLEAR`	Стирает дисплей
`_LCD_RETURN_HOME`	Возвращает курсор "домой" и возвращает сдвинутый экран в начальную позицию. На память дисплея не влияет.
`_LCD_CURSOR_OFF`	Выключает изображение курсора.
`_LCD_UNDERLINE_ON`	Включает изображение курсора в виде подчеркивания.
`_LCD_BLINK_CURSOR_ON`	Включает мерцание курсора.
`_LCD_MOVE_CURSOR_LEFT`	Перемещает курсор влево без изме-

	нения данных в памяти дисплея.
_LCD_MOVE_CURSOR_RIGHT	Перемещает курсор вправо без изменения данных в памяти дисплея.
_LCD_TURN_ON	Включение дисплея
_LCD_TURN_OFF	Выключение дисплея
_LCD_SHIFT_LEFT	Сдвиг экрана дисплея влево без изменения данных в памяти дисплея
_LCD_SHIFT_RIGHT	Сдвиг экрана дисплея вправо без изменения данных в памяти дисплея

9.2.Лабораторная работа №12

Исследование модели системы безопасности с клавиатурой и дисплеем

*Цель работы:*Изучение программирования и моделирования ввода данных с клавиатуры и вывода информации на дисплей в средах mikroC и Proteus.

Лабораторное задание

1. Создать в программе Proteus проект LAB12. Разместить проект в папке PRO-LAB12. Собрать схему модели системы безопасности (рис.9.1) с использованием микроконтроллера PIC16F877A.

2. Создать в программе mikroC проект с тем же названием LAB12. Разместить проект в папке MKC-LAB12.

3. В окне Code Editor среды mikroC набрать текст программы системы безопасности. Возможный вариант программы показан в листинге 9.1. Детально проанализировать программу и записать подробные комментарии.

4. Выполнить компиляцию программы, выбрав в главном меню *Build.* Если в программе имеются ошибки, найти их причину и устранить. В случае успешной компиляции, в папке MKC-LAB12 появится в числе прочих исполняемый HEX- файл. Обратите внимание на время создания этого файла.

5. Откройте в программе Proteus схему модели системы безопасности. Загрузите в микроконтроллер из папки МКС-LAB12 исполняемый файл программы, соответствующий Вашему времени создания. Выполните пуск и проверьте правильность функционирования системы безопасности. Сделайте скриншоты табло дисплея для правильного и неправильного ввода пароля.

6. Измените в программе «секретный пароль» так, чтобы он включал номер бригады, дату и месяц выполнения работы (например: 30810 – бригада 3, 8 октября). Повторите отладку, компиляцию программы и моделирование устройства. Результаты поместите в отчет.

Листинг 9.1

Программа системы безопасности

```
unsigned short kp, cnt;
int res;

char txt[6]="12345"; //Секретный пароль
char txt_pass[6];
char txt_cnt[6];
char cln, n;

char  keypadPort at PORTB; /*Модуль подключения
клавиатуры*/

// Модуль подключения ЖК-дисплея

sbit LCD_RS at RD2_bit;
sbit LCD_EN at RD3_bit;
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;


sbit LCD_RS_Direction at TRISD2_bit;
sbit LCD_EN_Direction at TRISD3_bit;
```

```
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;
// Конец модуля подключения ЖК-дисплея

void main()
{
  cnt=0; //номер ошибки
  start: // метка повтора набора
  n=0;   // номер цифры в наборе кода
  cln = 10; // установка счетчика столбца в коде
  Keypad_Init(); // Функция инициализации клавиатуры

  Lcd_Init();   // Инициализация ЖК-дисплея
  Lcd_Cmd(_LCD_CLEAR); // Очистка ЖК - дисплея
  Lcd_Cmd(_LCD_CURSOR_OFF); // Отключение курсора
  Lcd_Out(1, 1, "Password:"); / Вывод текста на
дисплей
  TRISC=0;  // Выводы PORTC на выход
  PORTC=0x00; // Очистить PORTC
  do   //цикл ввода пароля
{
    kp = 0;  // Сброс переменной клавиатуры

    // Ожидание нажатия кнопки
    do
      // kp = Keypad_Key_Press(); /* Режим сохранения
кода кнопки в переменной kp при нажатии */
      kp = Keypad_Key_Click();/* Режим сохранения
кода кнопки в переменной kp при клике */
      while (!kp); //ожидание первого клика
   // Преобразование введенного значения в код ASCII
    switch (kp)
    {
      case  1: kp = 49; break; // 1
      case  2: kp = 50; break; // 2
      case  3: kp = 51; break; // 3

      case  5: kp = 52; break; // 4
      case  6: kp = 53; break; // 5
```

```
      case  7: kp = 54; break; // 6

      case  9: kp = 55; break; // 7
      case 10: kp = 56;  break; // 8
      case 11: kp = 57; break; // 9

      case 13: kp = 42; break; // *
      case 14: kp = 48; break; // 0
      case 15: kp = 35; break; // #

    }
     Lcd_Chr(1, cln, kp); /* Вывод символа ASCII на
дисплей*/
     txt_pass[n]=kp; //запись символа в строку
     n++; //инкремент номера символа кода и столбца
     cln++;

   }
     while (cln<15); //цикл ввода пяти символов
     res = strcmp(txt,txt_pass); /*сравнение строки
секретного кода и введенного кода*/
     if(res==0)
        {Lcd_Out(2, 1, "Ok!"); //код верный
        PORTC=0x40;
        }
     else
     {Lcd_Out(2, 1, "Error!"); // ошибка кода
     Delay_ms(1000);
     cnt++;      //счет ошибок
     WordToStr(cnt,txt_cnt); /*преобразование числа в
строку*/
     {Lcd_Out(2, 1, "Times:");
     Lcd_Out(2, 8, txt_cnt);
     Delay_ms(1000);
     if(cnt<5) /* если ошибок меньше 5, повторить
               попытки*/
     goto start;
     }
    while (1);
   }
  }
```

7. Измените программу так, чтобы после пятикратного ввода неправильного пароля включился мерцающий курсор, а текст на дисплее стал периодически перемещаться вправо и влево на две позиции курсора. Зарегистрируйте полученные результаты.

8. Измените программу так, чтобы после выполнения п.7 на экране дисплея трижды перемещался справа налево текст «END». Зарегистрируйте результаты.

9. Измените программу так, чтобы после выполнения п.8 сдвинутый экран вернулся в начальную позицию, курсор включился в виде подчеркивания и перемещался справа налево до конца экрана.

Домашнее задание

Оформите отчет по работе. Отчет должен содержать программу в mikroC, скриншоты моделирования в Proteus, обсуждение результатов и выводы.

Контрольные вопросы

1. Как работает в Proteus матричная телефонная клавиатура?
2. Какую подпрограмму используют для инициализации клавиатуры в mikroC ?
3. Чем отличается работа подпрограмм клавиатуры для нажатия и для щелчка ?
4. Как в программе mikroC выполняется инициализация LCD-дисплея?
5. Как работает подпрограмма печати текста на ЖК, начиная с указанной позиции ?
6. Какие подпрограммы используют для печати символа в указанной и текущей позиции курсора?
7. Как послать на дисплей LCD-команду ?
8. Как в программе системы безопасности формируется код ASCII нажатой кнопки клавиатуры ?
9. Как в программе системы безопасности выполняется проверка правильности введенного кода ?

10. Как в программе выполняется преобразование числа в строку для печати на LCD-дисплее ?

9.3. Аналого-цифровое преобразование

Во многих цифровых системах существует необходимость преобразования аналоговых сигналов в цифровые. Дело в том, что большинство сигналов, которые создаются различными периферийными устройствами (микрофонами, телевизионными камерами, датчиками температуры, влажности и т.п.) являются аналоговыми. Для хранения, обработки и передачи таких сигналов их преобразуют в цифровую форму.

Различные типы аналого-цифровых преобразователей (АЦП) отличаются по точности, быстродействию, экономичности, выпускаются как отдельные микросхемы и, как правило, управляются процессором. Процессор выдает сигнал для начала преобразования, принимает от АЦП цифровое значение величины и сигнал об окончании преобразования.

Многие современные микроконтроллеры (например, PIC16F877A) имеют встроенные модули АЦП.

Важной характеристкой АЦП является разрядность. Разрядность показывает на сколько дискретных значений или уровней разделен диапозон изменения входного аналогового сигнала. Дискретные значения хранятся обычно в бинарном коде, их количество является степенью 2 и поэтому разрешение выражают в битах. Так АЦП с разрядностью 8 бит может кодировать аналоговый сигнал на 2^8=256 уровней. Значения могут быть в диапазоне от 0 до 255 (т.е. целое без знака) или от -128 до 127.

Разрешение АЦП может быть выражено через электрические величины. Разрешение в вольтах, которое называют шаг квантования, равно полному диапозону измерения напряжения, деленному на количество дискретных интервалов:

$$h = \frac{U_{max} - U_{min}}{2^m} = \frac{U_{max} - U_{min}}{N},$$

где: h – шаг квантования (дискретное значение напряжения в вольтах, приходящееся на единичное изменение выходного кода);

$(U_{max} - U_{min})$ – диапазон изменения входного напряжения;

m – разрядность АЦП;

N – количество интервалов (число градаций), на которые делится входной сигнал.

Например, используя АЦП с разрядностью 12 бит входное напряжение в диапозоне 0 – 10В можно преобразовать на 2^{12}=4096 квантованных уровня. Разрешение АЦП в вольтах составит:

$$h = \frac{U_{max} - U_{min}}{2^m} = \frac{10 - 0}{4096} = 2{,}44 мВ .$$

Модуль АЦП в микроконтроллере PIC16F877A

Микроконтроллер PIC16F877A имеет встроенный модуль аналого-цифрового преобразования (АЦП), который имеет восемь входных каналов. Входной аналоговый сигнал через коммутатор каналов заряжает внутренний конденсатор АЦП C_{HOLD}. Модуль АЦП преобразует напряжение, удерживаемое на конденсаторе C_{HOLD}, в соответствующий 10-разрядный цифровой код методом последовательного приближения. Источник верхнего и нижнего опорного напряжения может быть программно выбран с выводов V_{DD}, V_{SS}, RA2 или RA3.

Для управления АЦП в микроконтроллере используются 4 регистра:

- регистр результата ADRESH (старший байт);
- регистр результата ADRESL (младший байт);
- регистр управления ADCON0;
- регистр управления ADCON1.

Порядок функционирования этих регистров и назначение управляющих битов описан в главе 6 и перед выполнением данной работы надо внимательно прочитать раздел «Регистры ADCON0 (адрес 1Fh) и ADCON1 (адрес 9Fh)».

В регистрах ADRESH и ADRESL сохраняется 10-разрядный результат аналого-цифрового преобразования. Когда преобразование завершено, сбрасывается флаг GO/-DONE (ADCON0<2>) и устанавливается флаг прерывания ADIF.

После включения и конфигурации АЦП выбирают рабочий аналоговый вход. Соответствующие биты TRIS аналоговых каналов должны настраивать порт ввода/вывода на вход. Перед началом преобразования необходимо выдержать некоторую временную паузу.

Рекомендуемая последовательность действий для работы с АЦП следующая:

1. Настроить модуль АЦП:
   - настроить выводы как аналоговые входы, входы V_{REF} или цифровые каналы ввода/вывода (ADCON1);
   - выбрать входной канал АЦП (ADCON0);
   - выбрать источник тактовых импульсов для АЦП (ADCON0);
   - включить модуль АЦП (ADCON0).
2. Настроить прерывание от модуля АЦП (если необходимо):
   - сбросить бит ADIF в `0`;
   - установить бит ADIF в `1`;
   - установть бит PEIE в `1`;
   - установть бит GIE в `1`.
3. Выдержать паузу, необходимую для зарядки конденсатора $C_{HOLD.}$
4. Начать аналого-цифровое преобразование:
   - установить бит GO/-DONE в `1` (ADCON0).
5. Ожидать окончания преобразования:
   - ожидать пока бит GO/-DONE не будет сброшен в `0` или ожидать прерывание по окончанию преобразования;

6. считать результат преобразования из регистров ADRESH:ADRESL, сбросить бит ADIF в `0`, если это необходимо;
7. для следующего преобразования необходимо выполнить шаги с пункта 1 или 2, время преобразования одного бита определяется как время T_{AD}, минимальное время ожидания перед следующим преобразованием должно быть не менее $2T_{AD}$.

Для 10-разрядного результата требуется как минимум $12T_{AD}$. Параметры тактового сигнала для АЦП определяются программно, T_{AD} может принимать значения 2Tosc, 8Tosc, 32Tosc и (2-6 мкс) от внутреннего RC генератора.

Для получения корректного результата преобразования необходимо выбрать источник тактового сигнала АЦП, обеспечивающий время T_{AD} не менее 1,6 мкс.

10 - разрядный результат преобразования сохраняется в спаренном 16- разрядном регистре ADRESH:ADRESL. Запись результата преобразования может выполняться с правым или левым выравниванием в зависимости от бита ADFM. Незадействованные биты спаренного регистра читаются как `0`. Если модуль АЦП выключен, то 8-разрядные регистры ADRESH и ADRESL могут использоваться как регистры общего назначения.

Библиотека mikroC для работы с АЦП

Программирование работы АЦП существенно упрощается при использовании библиотеки mikroC. Библиотека содержит три подпрограммы.

Подпрограмма `ADC_Init`

Прототип	`void ADC_Init();`
Результат	Нет
Описание	Эта процедура инициализирует внутренний модуль АЦП PIC микроконтроллера для работы с RC-тактовым генератором. Такт определяет период

	времени, необходимый для выполнения преобразования АЦП (мин. 12TAD).
Требования	Микроконтроллер со встроенным АЦП-модулем.
Пример	`ADC_Init(); /* Initialize ADC module with default settings*/`

Подпрограмма `ADC_Get_Sample`

Прототип	`unsigned ADC_Get_Sample(unsigned short channel);`
Результат	Считать 10 или 12-битное значение из указанного канала (зависит от микроконтроллера).
Описание	Функция использует аналоговое значение из указанного канала. Параметр «канал» представляет собой канал, из которого должно быть взято аналоговое значение. (Вывод следует определить по спецификации). Примечание: Эта функция не работает с внешним опорным напряжением источника, а только с внутренним опорным напряжения.
Требования	Микроконтроллер со встроенным модулем АЦП. Перед использованием этой подпрограммы, модуль АЦП необходимо инициализировать. Перед использованием функции, убедитесь, что для настройка соответствующих битов TRISx назначает выводы АЦП в качестве входных.

Пример	`unsigned adc_value;`
	`...`
	`adc_value = ADC_Get_Sample(2); /* read analog value from ADC module channel 2*/`

Подпрограмма `ADC_Read`

Прототип	`unsigned ADC_Read(unsigned short channel);`
Результат	Считать 10 или 12-битным значение из указанного канала (зависит от микроконтроллера).
Описание	Инициализирует внутренний модуль АЦП для работы с RC-генератором. Такт определяет период

	времени, необходимый для выполнения преобразования AD (мин. 12TAD). Параметр канал представляет собой канал, из которого должно быть взято аналоговое значение. (Смотреть в соответствующей спецификации для входов АЦП микроконтроллера). Примечание: Эта функция не работает с внешним опорным напряжением источника, а только с внутренним опорным напряжением.
Требования	Микроконтроллер со встроенным модулем АЦП. Перед использованием функции, убедитесь, что для настройка соответствующих битов TRISx назначает выводы АЦП в качестве входных.

Пример	`unsigned tmp;`
	`...`
	`tmp = ADC_Read(2);  /* Read analog value from channel 2*/`

9.4. Лабораторная работа №13

Изучение модели аналого-цифрового преобразования

*Цель работы:*изучение программирования и моделирования аналого-цифрового преобразования измеряемого напряжения в бинарный код и десятичное число.

Лабораторное задание

1. Создать в программе Proteus проект LAB13. Разместить проект в папке PRO-LAB13. Собрать схему модели АЦП с индикаторными светодиодами (рис.9.2). Входное напряжение регулируется потенциометром RV1, изменяется в проделах 0 – 5В и измеряется вольтметром. Микроконтроллер с АЦП преобразует аналоговое напряжение в цифровой 10-разрядный бинарный код. Результат преобразования отображается напряжением на светодиодах.

2. Создать в программе mikroC проект с тем же названием LAB13 и разместить его в папке MKC-LAB13.

3. В окне Code Editor среды mikroC набрать текст программы аналого-цифрового преобразования напряжения (оцифровки напряжения). Возможный вариант программы показан в листинге 9.2. Детально проанализировать программу, дать пояснения содержания регистров ADCON0 и ADCON1.

4. Рассчитать для схемы рис.9.2 шаг квантования *h*. Для нескольких произвольных значений входного напряжения в диапозоне от 0 до 5 В проверить соответствие аналоговых и цифровых отсчетов. Сделать скриншоты схемы для исследованных значений напряжений.

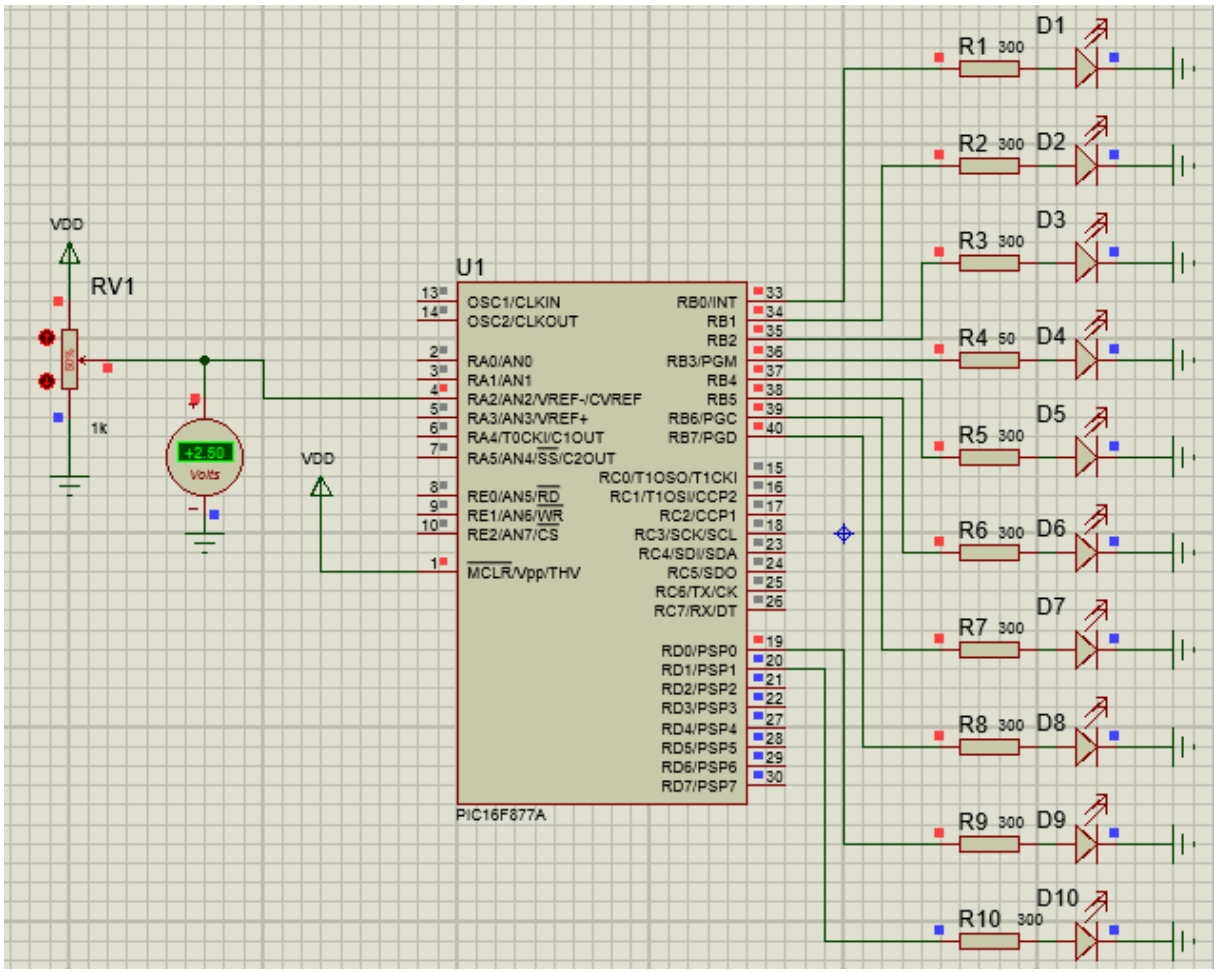


Рис.9.2. Схема модели АЦП

Листинг 9.2

Программа аналого-цифрового преобразования напряжения

```
unsigned int temp_res;
```

```
void main()
{
ADC_Init();
ADCON0=0x85;
ADCON1 = 0x80; /* Конфигурирование аналоговых входов
и Vref*/
TRISA = 0xFF; // Все выводы PORTA –входы
TRISB = 0; // Выводы PORTB –выходы
TRISD = 0; // Все выводы PORTD –выходы
do {
temp_res = Adc_Read(2);  /*получить результат
преобразования */
PORTB = temp_res; // Вывести младшие 8 битов в PORTB
PORTD = temp_res >> 8; /* Вывести старшие 2 бита на
RD7, RD6 */
} while(1);
}
```

5. Подключите к порту D микроконтроллера ЖК-дисплей, аналогично тому, как это было сделано в лабораторной работе №12. Отредактируйте программу так, чтобы измеренное напряжение в цифровой форме выводилось на ЖК-дисплей. Выполните инициализацию дисплея, подключенного к порту D. Затем вычислите шаг квантования h и умножьте его на бинарный код `temp_res` с выхода АЦП. Вы получите простейший цифровой вольтметр.

Рекомендуем посмотреть и использовать библиотеку преобразования mikroC, и, в частности, функцию `FloatToStr`. Эта функция создает выходную строку `output` для числа с плавающей точкой `number`. Выходная строка содержит нормализованный формат числа `number` (мантисса в диапазоне от 0 до 1) со знаком в крайней левой позиции. Мантисса имеет фиксированный формат длиной 6 цифр (`0.ddddd`), т.е. после точки всегда следуют 5 десятичных цифр. Выходная строка `output` должна иметь длину не менее 13 символов.

Пример:

```
float Volt = 0,567;
char txt[13];
//...
```

```
FloatToStr(volt, txt); // txt будет "5,67e-1"
```

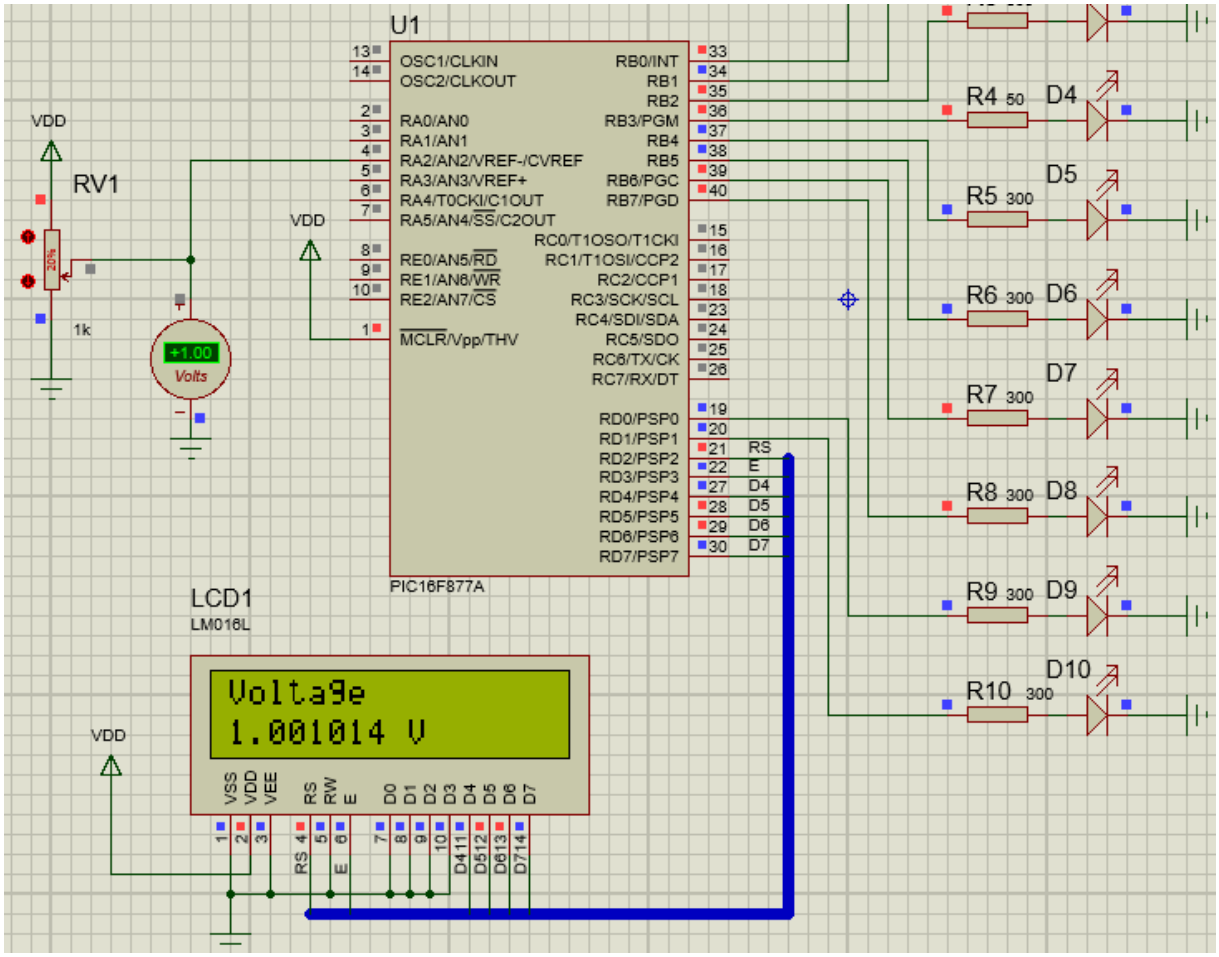


Рис.9.3. Схема цифрового вольтметра

Возможный вариант текста программы для схемы с цифровым вольтметром приведен на листинге 9.3.

Листинг 9.3

```
unsigned int temp_res;
float volt;
char txt_volt[16];

// Модуль подключения ЖК-дисплея

sbit LCD_RS at RD2_bit;
sbit LCD_EN at RD3_bit;
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
```

```
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;

sbit LCD_RS_Direction at TRISD2_bit;
sbit LCD_EN_Direction at TRISD3_bit;
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;
// Конец модуля подключения ЖК-дисплея

void main()
{
  Lcd_Init();   // Инициализация ЖК-дисплея
  Lcd_Cmd(_LCD_CLEAR); // Очистка ЖК - дисплея
  Lcd_Cmd(_LCD_CURSOR_OFF); // Отключение курсора
  Lcd_Out(1, 1, "Voltage"); /* Вывод текста на
                               дисплей */

ADC_Init();   //Инициализация АЦП
ADCON0=0x85;
ADCON1 = 0x80; /* Конфигурирование аналоговых входов
                  и Vref*/
TRISA = 0xFF; // Все выводы PORTA –входы
TRISB = 0; // Выводы PORTB –выходы
TRISD = 0; // Все выводы PORTD –выходы
do {
temp_res = Adc_Read(2);  /*подучить результат
                           преобразования */
PORTB = temp_res; // Вывести младшие 8 битов в PORTB
PORTD = temp_res >> 8; /* Вывести старшие 2 бита на
                          RD0, RD1*/
volt=temp_res*(0.004883); /*Вычисление десятичного
                            значения напряжения */
FloatToStr(volt, txt_volt);  /* преобразование
                         десятичного числа в текст*/
Lcd_Out(2, 1, txt_volt); //Вывод числа на дисплей
Lcd_Cmd(_LCD_MOVE_CURSOR_RIGHT);  /*Сдвиг курсора
                                    вправо*/
Lcd_Chr_Cp('V');  // Вывод символа
Delay_ms(200);
```

```
  } while(1);
}
```

Домашнее задание

Оформите отчет по работе. Отчет должен содержать программу для модели с LCD- дисплеем в mikroC, скриншоты моделирования в Proteus, обсуждение результатов и выводы.

Контрольные вопросы

1. Для чего применяют аналого-цифровые преобразователи?
2. Как рассчитать разрешение АЦП?
3. Какие регистры используют для управления АЦП в микроконтроллере PIC16F877?
4. Где сохраняется результат преобразования и как выполняют выравнивание результата?
5. Какие подпрограммы из библиотеки mikroC используют для работы с АЦП?
6. Объясните, как в программе выполняется преобразование бинарного кода в текст для отображения на LCD-дисплее десятичного значения напряжения?
7. Объясните работу программы из листинга 9.3.

9.5. Широтно-импульсная модуляция

Широтно-импульсная модуляция (ШИМ) сигналов является эффективным способом управления уровнем напряжения питания различных устройств. Период сигналов с ШИМ постоянный, а соотношение длительности импульса к периоду следования изменяется. Термин «коэффициент заполнения» (duty cycle) определяет отношение времени включенного состояния к периоду следования импульсов $D = \frac{t_{on}}{T}$ (рис.9.4).

Обратную величину называют скважностью импульсов $Q = \frac{1}{D} = \frac{T}{t_{on}}$. Широтно-импульсный сигнал управляет

ключевыми транзисторами, которые обеспечивают высокий коэффициент полезного действия, так как в закрытом состоянии при отсутствии тока и в полностью открытом состоянии при малом внутреннем сопротивлении хорошие ключевые транзисторы практически не потребляют энергию.

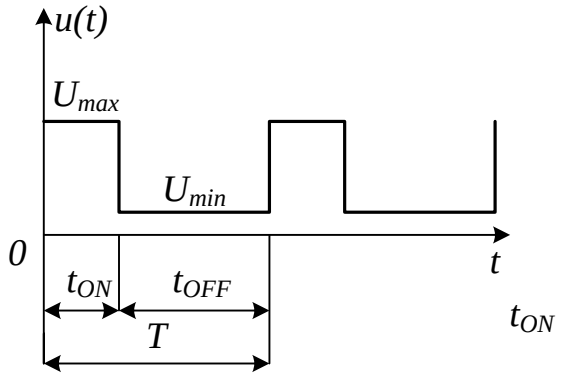


Рис.9.4. Сигнал с ШИМ

Если максимальное напряжение импульсов $U_{\max}$, а миниальное равно нулю, то среднее значение импульсной последовательности с ШИМ будет равно $U_{ср} = D \cdot U_{\max}$. Таким образом, регулируя коэффициент заполнения, можно управлять средним значением импульсного сигнала и, выполнив сглаживание LC- фильтром, получить управляемое напряжение питания. Этот метод успешно применяется в понижающих и повышающих *DC-DC* преобразователях постоянного напряжения.

Модуль CCP в PIC контроллерах

В микроконтроллере PIC16F877A для реализации ШИМ применяют два специальных модуля CCP (Capture/Compare/PWM – Захват/Сравнение/ШИМ). Каждый модуль CCP содержит 16-ти битный регистр, который может действовать в качестве:

- 16-ти разрядного регистра захвата;
- 16-ти разрядного регистра сравнения;
- двух 8-ми разрядных (ведущий и ведомый) регистров ШИМ.

Оба модуля CCP идентичны в работе за исключением функционирования триггера специального события.

Модуль CCP1:

Регистр CCPR1 модуля CCP состоит из двух 8-разрядных регистров: CCPR1L (младший разряд), CCPR1H (старший разряд). В регистре CCP1CON находятся управляющие биты модуля CCP1, доступные для записи и чтения. В режиме сравнения триггер специального события сбрасывает таймер TMR1.

Модуль CCP2:

Регистр CCPR2 модуля CCP состоит из двух 8-разрядных регистров: CCPR2L (младший разряд), CCPR2H (старший разряд). В регистре CCP2CON находятся управляющие биты модуля CCP2, доступные для записи и чтения. В режиме сравнения триггер специального события сбрасывает таймер TMR1 и запускает преобразование АЦП (если АЦП включен). В режимах захвата и сравнения модули CCP используют таймер TMR1, а в режиме ШИМ – таймер TMR2.

Регистры CCP1CON/ CCP2CON Адрес 17h/1Dh

Регистры CCP1CON/ CCP2CON доступны для чтения и записи.

Биты 7 и 6		Не используются, читаются как `0`
Бит 5 и 4	CCPxX: CCPxY	Младшие биты скважности ШИМ. В режиме захвата и сравнения не используются. В режиме ШИМ - два младших бита скважности. Восемь старших находятся в CCPxL.
Бит 3-0	CCPxM3: CCPxM0	Режим работы модуля CCPx 0000=модуль CCPx выключен (сброс модуля CCPx); 0100=захват по каждому заднему фронту сигнала; 0101=захват по каждому переднему фронту

		сигнала; 0110=захват по каждому 4-му переднему фронту сигнала; 0111= захват по каждому 16-му переднему фронту сигнала; 1000= сравнение, устанавливает выходной сигнал (устанавливается флаг CCPxIF в `1`); 1001= сравнение, сбрасывает выходной сигнал (устанавливает флаг CCPxIF в `1`); 1010= сравнение, на выходной сигнал не влияет (устанавливает флаг CCPxIF в `1`); 1011= сравнение, триггер специального события (устанавливает флаг CCPxIF в `1`; на вывод CCPx не влияет). CCP1 – сброс таймера TMR1. CCP2- сброс таймера TMR1, запуск преобразования АЦП (если АЦП включено). 11xx = ШИМ режим.

Режим захвата

При возникновении события захвата 16-разрядное значение счетчика TMR1 переписывается в регистры CCPR1L:CCPR1H модуля CCP1. Событием захвата может быть:

- каждый задний фронт сигнала на входе RC2/CCP1;
- каждый передний фронт сигнала на входе RC2/CCP1;
- каждый 4-й передний фронт сигнала на входе RC2/CCP1;
- каждый 16-й передний фронт сигнала на входе RC2/CCP1.

Тип захвата устанавливается битами CCP1M3:CCP1M0 в регистре CCP1CON. Порт ввода/вывода RC2/CCP1 должен быть настроен на вход установкой бита TRISC<2> в `1`. Таймер TMR1 должен работать в синхронизированном режиме.

Режим сравнения

В этом режиме 16 – разрядный регистр CCPR1 сравнивается со значением TMR1. Как только значения в регистрах становятся одинаковыми, модуль CCP1 изменяет состояние вывода RC2/CCP1:

- устанавливает высокий уровень сигнала;
- устанавливает низкий уровень сигнала;
- на вывод не воздействует.

Действие при совпадении может быть выбрано битами CCP1M3:CCP1M0 в регистре CCP1CON. В момент изменения состояния вывода устанавливатся флаг прерывания CCP1IF в `1`.

Режим ШИМ

В режиме ШИМ модуля CCP1 вывод RC2/CCP1 используется в качестве выхода 10-разрядного ШИМ. При этом бит направления TRISC<2> должен быть сброшен в `0`. Структурная схема модуля CCP1 в режиме ШИМ показана на рис.9.5.

В структурной схеме выполняется два сравнения. Период ШИМ определяется значением в регистре PR2 периода таймера TMR2 и может быть вычислен по формуле:

$$T_{ШИМ} = \left[\left(PR2\right)+1\right]\cdot 4\cdot T_{osc}\cdot (коэффициент\ предделителя\ TMR2).$$

Когда значение TMR2 сравнивается с PR2, выполняются следующие действия:

- TMR2 сбрасывается в 00h;
- устанавливается высокий уровень сигнала на выводе CCP1;
- модуль ШИМ начинает новый цикл, загружая значение из регистра CCPR1L в CCPR1H.

Скважность ШИМ определяется битами в регистрах CCPR1L и CCP1CON<5:4>. Для 10-разрядного ШИМ старшие восемь бит сохраняются в регистре CCPR1L, а младшие два бита в регистре CCPCON<5:4>. Длительность сигнала высокого уровня можно рассчитать по формуле:

$$Длительность\ ШИМ = (CCPR1L : CCPCON < 5:4 >) \cdot T_{osc} \cdot$$
$$\cdot (коэффициент\ предделителя\ TMR2)\ .$$

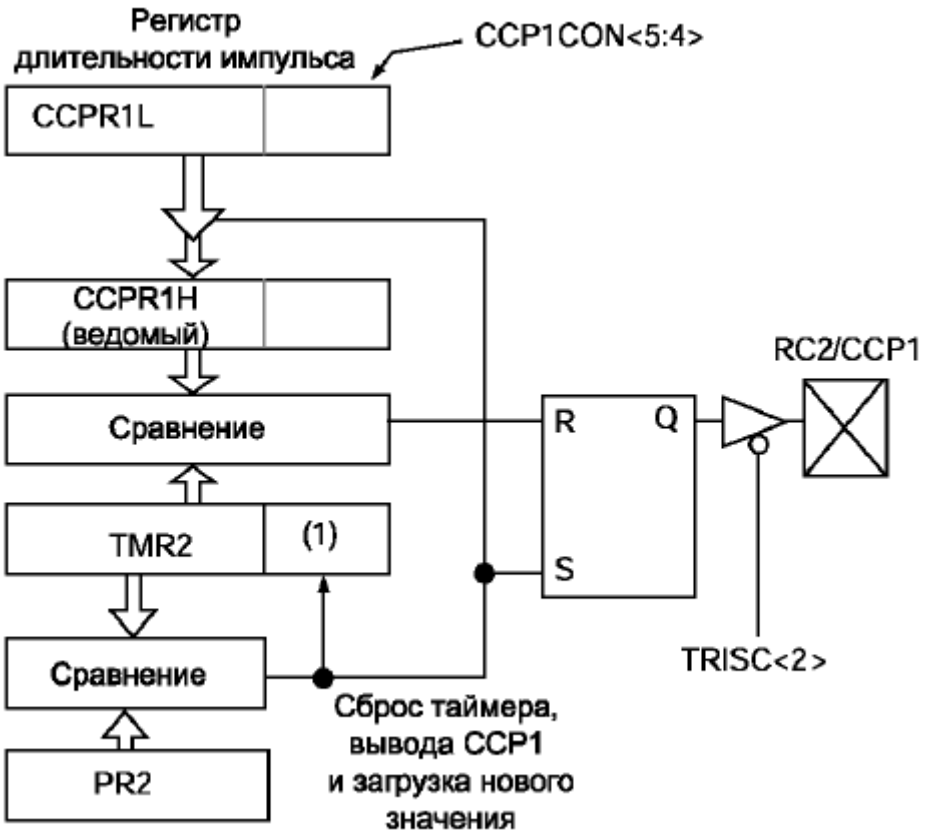


Рис.9.5. Структурная схема модуля CCP1 в режиме ШИМ

Биты в регистре CCPR1L и CCP1CON<5:4> могут быть изменены в любое время, но значение в регистре CCPR1H не изменяется, пока не произойдет соответствие PR2 и TMR2. В ШИМ режиме регистр CCPR1H доступен только для чтения и образует буфер ШИМ. Эффект буферизации необходим для записи нового значения длительности импульса ШИМ.

Библиотека mikroC для работы с ШИМ

MikroC предоставляет библиотеку, которая упрощает использование аппаратного модуля CCP в режиме PWM.

Отдельные микроконтроллеры PIC с двумя или более модулями CCP требуют предварительно определить модуль, который будет использован. Для этого достаточно просто

добавить номер `1` или `2` к PWM в имени функции. Например, `PWM2_Start();`. Также, с целью обратной совместимости с предыдущими версиями компилятора и облегчения управления кодами, микроконтроллеры с несколькими модулями CCP имеют PWM библиотеку, которая идентична PWM1 (т.е. можно использовать `PWM_Init(5000)` вместо `PWM1_Init(5000)` для инициализации модуля CCP1).

Подпрограмма `PWMx_Init`

Прототип	`void PWMx_Init(long freq);`
Результат	Нет
Описание	Инициализирует модуль PWM с коэффициентом заполнения 0. Параметр `freq` означает нужную частоту ШИМ в Гц, указанную в паспорте устройства и связанную с F_{OSC}. Эта процедура должна быть вызвана перед использованием других функций из библиотеки ШИМ.
Требования	Микроконтроллер должен иметь CCP модуль. Компилятор должен знать значение частоты ШИМ в момент компиляции. Поэтому этот параметр должен быть постоянным.
Пример	Инициализация PWM модуля на частоте 5 кГц: `PWM1_Init(5000);`

Подпрограмма `PWMx_Set_Duty`

Прототип	`void PWMx_Set_Duty(unsigned short duty_ratio);`
Результат	Нет
Описание	Устанавливает коэффициент заполнения ШИМ. Параметр `duty` принимает значения от 0 до 255, где 0 соответствует 0%, 127-50%, а 255-100%. Другие конкретные значения скважности могут быть рассчитаны.

Требования	Микроконтроллер должен иметь CCP модуль. Подпрограмма `PWMx_Init` должна быть вызвана перед использованием этой функции.
Пример	Установить коэффициент заполнения 75%: `PWM1_Set_Duty(192);`

Подпрограмма `PWMx_Start`

Прототип	`void PWMx_Start(void);`
Результат	Нет
Описание	Запуск ШИМ
Требования	Микроконтроллер должен иметь CCP модуль. Подпрограмма `PWMx_Init` должна быть вызвана перед использованием этой функции.
Пример	`PWM1_Start();`

Подпрограмма `PWMx_Stop`

Прототип	`void PWMx_Stop(void);`
Результат	Нет
Описание	Останов ШИМ
Требования	Микроконтроллер должен иметь CCP модуль. Подпрограмма `PWMx_Init` должна быть вызвана перед использованием этой функции. Подпрограмма `PWM1_Start()` должна быть вызвана перед этой подпрограммой.
Пример	`PWM1_Stop();`

9.6. Лабораторная работа №14

Изучение модели управления двигателем с использованием ШИМ

Цель работы: изучение программирования и моделирования микропроцессорного управления двигателем постоянного тока с использованием ШИМ, применение цифрового осциллографа.

Лабораторное задание

1. Создать в программе Proteus проект LAB14. Разместить проект в папке PRO-LAB14. Собрать схему модели управления двигателем с использованием ШИМ (рис.9.6). В микроконтроллере PIC16F877A работают два модуля ШИМ. Коэффициент заполнения PWM1 увеличивается при нажатии кнопки Up, подающий высокий уровень на вход RA0/AN0, и уменьшается при нажатии кнопки Down, соединенной с RA1/AN1. Коэффициент заполнения PWM2 увеличивается при нажатии кнопки Up, подающей высокий уровень на вход RA2/AN2, и уменьшается при нажатии кнопки Down, соединенной с RA3/AN3.

Выходной сигнал PWM1 с вывода RC2/CCP1 управляет полевым транзистором VN66 в стоке, которого включен двигатель постоянного тока с инерцией и нагрузкой. Вольтметр V1 измеряет среднее значение напряжения PWM1.

Среднее значение выходного сигнала PWM2 с вывода RC1/T1OSI/CCP2 измеряется вольтметром V2.

Сигналы PWM1, PWM2 и VN на обмотке двигателя контролируются осциллографом.

2. Создать в программе mikroC проект LAB14 и разместить его в папке MKC-LAB14. Используя библиотеку mikroC, составить программу управления двигателем с использованием ШИМ. Возможный вариант программы показан в листинге 9.4.

3. Детально проанализизовать программу. Выполнить отладку и компиляцию. Добиться успешного результата.

Листинг 9.4

Программа управления двигателем с использованием ШИМ

```
unsigned short current_duty, old_duty, current_duty1,
old_duty1;

void InitMain() {
  ADCON0=0;  // отключение АЦП
```

```
  ADCON1 = 6; //установка цифровых вводов PORTA
  PORTA = 0;
  TRISA = 255; // все выводы PORTA - входы
  PORTB = 0;   // установка PORTB в `0`
  TRISB = 0;   // все выводы PORTB - выходы
  PORTC = 0;   // установка PORTC в `0`
  TRISC = 0;   // все выводы PORTC - выходы
  PWM1_Init(5000);  /* Инициализация модуля PWM1 на
частоте 5 кГц*/
  PWM2_Init(5000); /* Инициализация модуля PWM2 на
частоте 5 кГц*/
}

void main()
{
  InitMain();
  current_duty  = 128;  /* начальное значение для
коэффициента заполнения current_duty*/
  current_duty1 = 128;  /* начальное значение для
коэффициента заполнения current_duty1*/

  PWM1_Start();  // запуск PWM1
  PWM2_Start();  // запуск PWM2
  PWM1_Set_Duty(current_duty);  /* Установить
коэффициет заполнения для PWM1*/
  PWM2_Set_Duty(current_duty1); /* Установить
коэффициет заполнения для PWM2*/

  while (1)   // бесконечный цикл

    {if (RA0_bit) // нажата кнопка на RA0
      { Delay_ms(4);  //задержка
       current_duty++;  // инкремент current_duty
      PWM1_Set_Duty(current_duty);
     }
    if (RA1_bit) // нажата кнопка RA1
    { Delay_ms(4);
      current_duty--;  // декремент current_duty
      PWM1_Set_Duty(current_duty);
     }
    if (RA2_bit) // нажата кнопка RA2
```

```
    { Delay_ms(4);
      current_duty1++;  // инкремент current_duty1
      PWM2_Set_Duty(current_duty1);
     }
    if (RA3_bit) // нажата кнопка RA3
    { Delay_ms(4);
      current_duty1--; // декремент current_duty1
      PWM2_Set_Duty(current_duty1);
     }
    Delay_ms(5);  /* небольшое замедление темпов
                     изменения */
  }
}
```

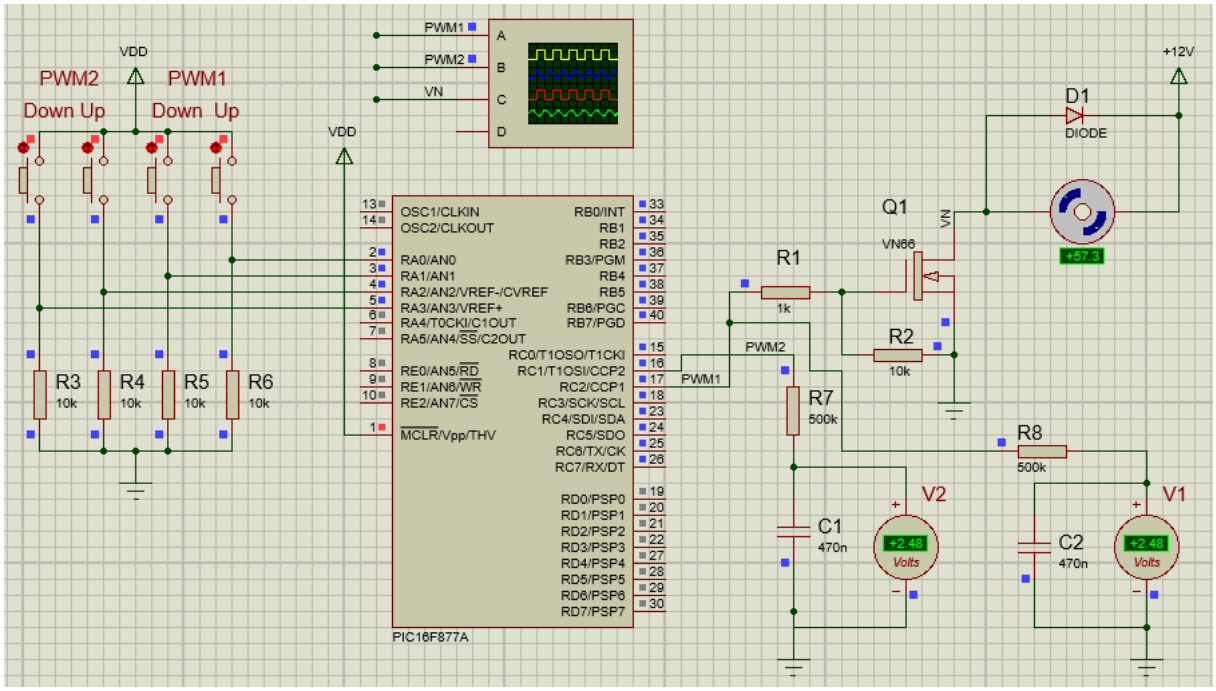


Рис.9.6. Модель управления двигателем с использованием ШИМ

4. Загрузить программу в микроконтроллер модели управления двигателем. Выполнить пуск.

5. Настроить цифровой осциллограф (рис.9.7):

- длительность горизонтальной развертки установить 0,2 мс/дел.;
- синхронизацию включить по каналу C;
- входы каналов A, B и C включить по постоянному току;

- усиление каналов A, B и C установить 2В/дел.;
- регулировкой *Position* сместить осциллограммы как показано на рис. 9.7.

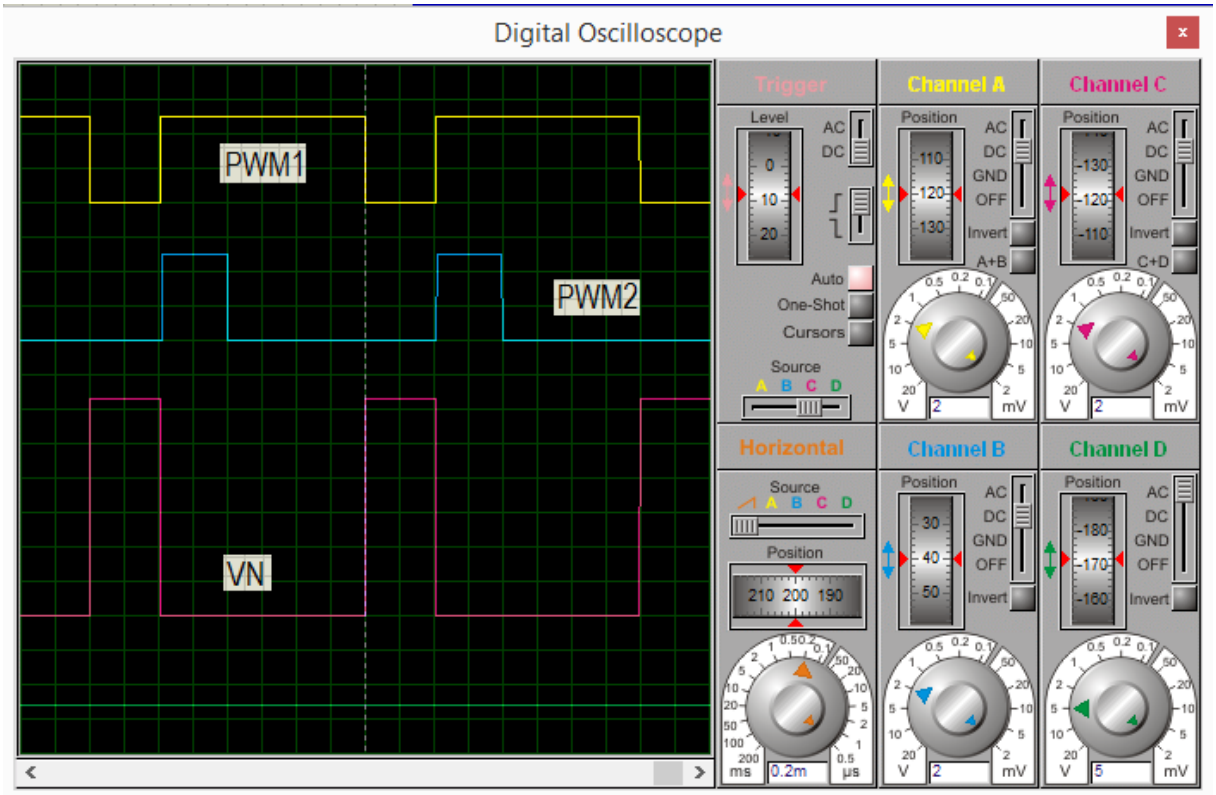


Рис.9.7. Осциллограммы сигналов

6. Определить по осциллограммам период ШИМ.

7. Используя кнопки управления, установить коэффициент заполнения для PWM1 равным 25%, а для PWM2 равным 75%. Записать показания вольтметров V1 и V2, а также скорость двигателя в установившемся режиме. Угловая скорость двигателя измерена в градусах в секунду. Зарегистрировать осциллограммы сигналов.

8. Определить регулировочную характеристику двигателя как зависимость скорости вращения от коэффициента заполнения ШИМ. Для этого последовательно устанавливать следующие значения коэффициента заполнения для PWM1: 12,5%, 25%, 37,5%, 50%, 62,5%, 75%, 87,5%. После окончания разгона двигателя записать в таблицу 9.1. угловую скорость двигателя и

показания вольтметра V1. Зарегистрировать осциллограммы сигналов для разных значений коэффициента заполнения.

Таблица 9.1

Коэффициент заполнения ШИМ	12,5%	25%	37,5%	50%	62,5%	75%	87,5%
Скорость двигателя (град/сек)							
Напряжение V1 (В)							

Домашнее задание

Оформите отчет по работе. Отчет должен содержать программу в mikroC, скриншоты моделирования в Proteus, результаты измерений, регулировочную характеристику двигателя, обсуждение результатов и выводы.

Контрольные вопросы

1. Что называют широтно-импульсной модуляцией сигналов?
2. Объясните значения терминов «коэффициент заполнения» и «скважность» для ШИМ?
3. Как используют ШИМ для управления напряжением питания ?
4. Расскажите о составе модулей CCP, которые реализуют режим ШИМ в микроконтроллера PIC16F877A.
5. Какие регистры управляют модулями CCP ?
6. Поясните назначение битов регистров, управляющих модулями CCP.
7. В каких режимах работают модули CCP микроконтроллера PIC16F877A ?
8. Поясните по структурной схеме работу моделя CCP в режиме ШИМ.
9. Какие подпрограммы из библиотеки mikroC используют для управления модулем CCP в режиме ШИМ ?

10. Как следует задавать нужный коэффициент заполнения ШИМ в подпрограмме mikroC ?

11. Как выполнить настройку осциллографа в программе Proteus ?

12. По описанию из «Help» расскажите о модели двигателя постоянного тока в программе Proteus ?

13. Поясните структуру и работу программы из листинга 9.4.

9.7. Изучение универсального синхронно-асинхронного приемопередатчика (USART)

USART – это модуль последовательного ввода/вывода, который может работать в полнодуплексном асинхронном режиме (одновременные передача и прием сообщений) для связи с терминалами, персональными компьютерами или в синхронном полудуплексном режиме (попеременная передача информации, когда источник и приемник последовательно меняются местами) для связи с микросхемами ЦАП, АЦП, последовательными EEPROM и т.д.

USART может работать в трех режимах:

- асинхронный, полный дуплекс;
- ведущий синхронный, полудуплекс;
- ведомый синхронный, полудуплекс.

Биты SPEN (RCSTA<7>) и TRISC<7:6> должны быть установлены в `1` для использования выводов RC6/TX/CK и RC7/RX/DT в качестве портов универсального синхронно-асинхронного приемопередатчика. В асинхронном режиме RC6 действует как выход передатчика данных TX, а RC7 используется как вход приемника данных RX. Данные обычно передаются 8-ми битными словами, начиная с младшего бита. Приемник начинает прием с той скоростью, с которой передаются данные. В примерах часто используют скорость 9600 бод (примерно 10 кбит/с). Так как используются отдельные линии передачи и приема, эти операции можно делать одновременно.

На рис.9.8 показана блок-схема соединения микроконтроллера с компьютером. Модуль USART работает с TTL – уровнями сигналов и нуждается в дополнительном линейном драйвере для преобразования сигналов в более высокое симметричное напряжение. Для интерфейса RS232 обычно используется напряжение +/- 12 В.

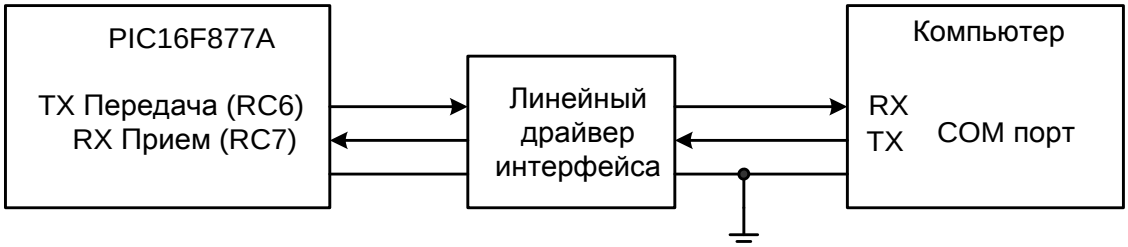


Рис.9.8. Блок-схема соединения микроконтроллера с компьютером

Для управления модулем USART используют два регистра: TXSTA и RCSTA.

Регистр управления и статуса передатчика TXSTA (адрес 98h)

Бит 7	CSRC	Выбор источника тактового сигнала Синхронный режим: 1=ведущий, внутренний тактовый сигнал от BRG; 0=ведомый, внешний тактовый сигнал с входа CK; Асинхронный режим: Не имеет значения.
Бит 6	TX9	Разрешение 9- разрядной передачи: 1= 9-разрядная передача; 0=8-разрядная передача.
Бит 5	TXEN	Разрешение передачи: 1= разрешена; 0=запрещена. *Примечание*. В синхронном режиме биты SREN/CREN отменяют действие бита

		TXEN.
Бит 4	SYNC	Режим работы USART: 1=синхронный; 0=асинхронный.
Бит 3		Не используется: читается как `0`.
Бит 2	BRGH	Выбор скоростного режима Синхронный режим: не имеет значения. Асинхронный режим: 1=высокоскоростной режим; 0=низкоскоростной режим.
Бит 1	TRMT	Флаг очистки сдвигового регистра передатчика TSR: 1=TSR пуст; 0=TSR полон.
Бит 0	TX9D	9-й бит передаваемых данных (может использоваться для проверки четности)

Регистр управления и статуса приемника RCSTA (адрес 18h)

Бит 7	SPEN	Разрешение работы последовательного порта: 1=модуль USART включен (выводы RC7/RX/DT, RC6/TX/CK подключены к USART); 0=модуль USART выключен.
Бит 6	RX9	Разрешение 9- разрядного приема: 1= 9-разрядный прием; 0=8-разрядный прием.
Бит 5	SREN	Разрешение одиночного приема Синхронный режим: 1= разрешен одиночный прием; 0=запрещен одиночный прием. Сбрасывается в `0` по завершению приема. *Примечание*. В режиме ведомого не имеет

		значения. Асинхронный режим: не имеет значения.
Бит 4	CREN	Разрешение приема Синхронный режим: 1= прием разрешен (CREN автоматически сбрасывает бит SREN); 0=прием запрещен. Асинхронный режим: 1= прием разрешен; 0=прием запрещен.
Бит 3	ADDEN	Разрешение детектирования адреса Асинхронный 9-разрядный прием (RX9=1): 1=детектирование адреса разрешено; 0=детектирование адреса запрещено. Асинхронный 8-разрядный прием (RX9=0): не имеет значения. Синхронный режим: не имеет значения.
Бит 2	FERR	Ошибка кадра, сбрасывается при чтении регистра RCREG: 1=произошла ошибка кадра; 0=ошибки кадра не было.
Бит 1	OERR	Ошибка переполнения внутреннего буфера, устанавливается в `0` при сбросе бита CREN: 1=произошла ошибка переполнения; 0=ошибки переполнения не было.
Бит 0	RX9D	9-й бит принятых данных (может использоваться для проверки четности).

Генератор частоты обмена USART BRG

В синхронном ведущем и асинхронном режимах используется отдельный 8-разрядный генератор скорости обмена BRG, период которого определяется значением в регистре

SPBRG (адрес 99h). В документации на микроконтроллер приведены таблицы для вычисления скорости обмена в бодах при различных режимах работы модуля USART (относительно внутреннего тактового генератора микроконтроллера).

Работа модуля USART

Рассмотрим кратко работу модуля USART. Сначала будем считать, что микроконтроллер передает данные (рис.9.9). Отправитель и получатель должны быть инициализированы, чтобы использовать одну и ту же скорость передачи данных, количество битов данных (по умолчанию 8) и количество стоповых битов (по умолчанию 1). Выход передатчика (TX), имеет высокий уровень во время простоя (внешняя линия на выходе линейного драйвера будет иметь отрицательный уровень). Когда в буфер последовательного регистра (TXREG) записаны данные, они автоматически передаются следующим образом. Начало передачи определяется стартовым битом низкого уровня. Затем из регистра передатчика выводятся следующие 8 бит с периодичностью, установленной выбранной скоростью передачи. Чередование высокого и низкого уровня бит определяется конкретным содержанием слова. После последнего бита данных передается стоповый бит высокого уровня. На линии остается высокий уровень, если нет новых данных для передачи. Следующее слово можно передать после некоторой задержки. Данные часто кодируются в ASCII, поэтому удобно передавать текстовые сообщения.

Временная диаграмма асинхронного приема данных показана на рис.9.10. Приемник должен быть инициализирован, чтобы читать данные со скоростью их передачи. На скорости 9600 бод период бита составляет около 100 мкс. Когда обнаруживается задний фронт стартового бита, приемник должен ждать 1,5 битовых периода, затем он считывает заданное количество бит данных до стопового бита. Стоповый бит подтверждает конец байта и то, что может начаться новая передача.

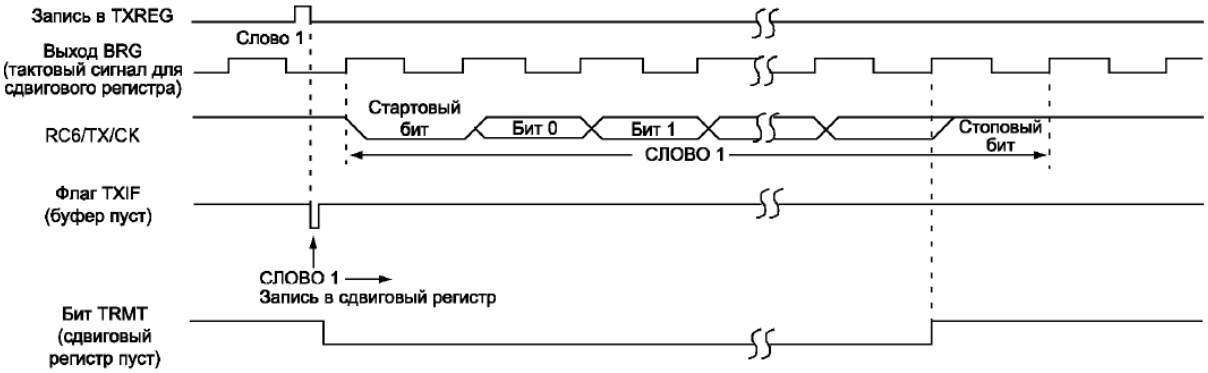


Рис.9.9. Временная диаграмма асинхронной передачи данных

Флаг прерывания RCIF используется, чтобы сигнализировать микроконтроллеру, что ожидается передача данных. После записи трех байтов в приемный регистр RCREG устанавливается бит переполнения применика OERR. Три байта данных считываются из приемного регистра RCREG, перед тем, как поступает следующий байт. Для разрешение приема бит CREN устанавливается в `1`.

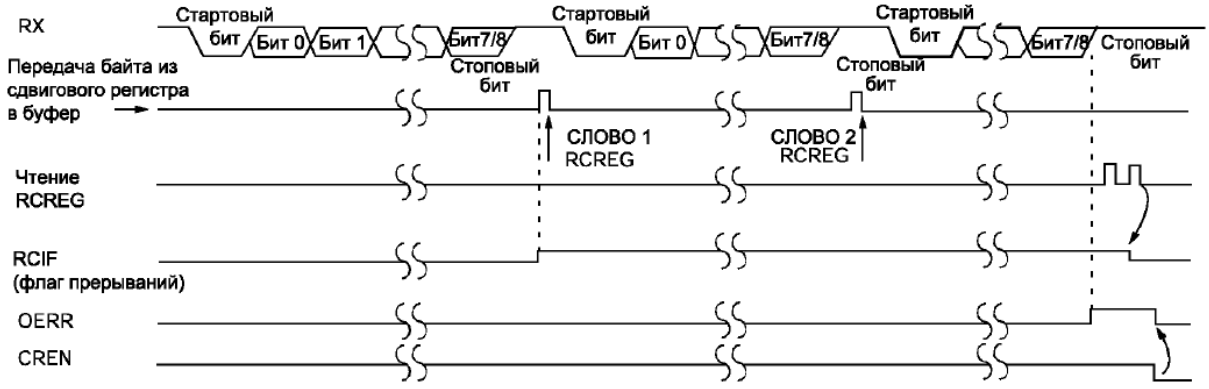


Рис.9.10. Временная диаграмма асинхронного приема данных

В документации микроконтроллера детально описаны и другие режимы работы модуля USART

Библиотека mikroC для работы с модулем USART

Модуль USART позволяет работать с рядом PIC-совместимых микроконтроллеров. В mikroC PRO библиотека обеспечивает удобную работу в *асинхронном режиме* (полный дуплекс). Поэтому соответствующая библиотека именуется UART Library, а модули USART называются модулями UART.

Можно легко общаться с другими устройствами с помощью RS-232 протокола (например, с персональным компьютером).

Так как в микроконтроллерах может быть несколько модулей USART, библиотечные подпрограммы требует указать модуль, который будет использован. Чтобы выбрать нужный модуль, надо изменить символ `x` в прототипе подпрограммы на цифру 1 или 2. Переключение модулей в библиотеке UART делается функцией `UART_Set_Active` (UART модуль должны быть предварительно инициализирован).

Для PIC16F877A можно использовать функции, перечисленные ниже.

Подпрограмма `UARTx_Init`

Прототип	`void UARTx_Init(const unsigned long baud_rate);`
Результат	Нет
Описание	Настраивает и инициализирует UART модуль. Внутренний UART модуль установлен в режим: • включен приемник; • включен передатчик; • размер кадра 8 бит; • 1 стоповый бит; • режим четности отключен; • асинхронная работа. Параметры: `baud_rate` – требуемая скорость передачи.
Требования	Микроконтроллер должен иметь UART модуль.
Пример	`/* Инициализировать модуль UART1 и установить связь на скорости 9600 бод*/` `UART1_Init(9600);`

Подпрограмма `UARTx_Data_Ready`

Прототип	`UARTx_Data_Ready;`
Результат	1, если данные готовы для чтения; 0, если в регистре приема нет данных
Описание	Используют функцию, чтобы проверить готовы ли данные в приемном буфере для чтения.

Требования	Микроконтроллер должен иметь UART модуль. UART модуль должен быть инициализирован.

Пример	`// Если данные готовы, считать их`
	`if (UART1_Data_Ready() == 1) {`
	`  receive = UART1_Read();`
	`  }`

Подпрограмма `UARTx_Tx_Idle`

Прототип	`char UARTx_Tx_Idle();`
Результат	1, если данные были переданы; 0, в противном случае.
Описание	Используют функцию, чтобы проверить сдвиговый регистр передачи пуст или нет.
Требования	Перед использованием этой функции модуль UART должен быть инициализирован и создана связь.

Пример	`/* Если предыдущие данные были сдвинуты, отправьте следующие данные*/`
	`if (UART1_Tx_Idle() == 1) {`
	`  UART1_Write(_data);`
	`  }`

Подпрограмма `UARTx_Read`

Прототип	`char UARTx_Read();`
Результат	Возвращает принятый байт.
Описание	Функция получает байт с помощью UART. Сначала надо использовать функцию `UARTx_Data_Ready`, чтобы проверить готовы ли данные.
Требования	Перед использованием этой функции модуль UART должен быть инициализирован.

Пример	`// If data is ready, read it:`
	`if (UART1_Data_Ready() == 1) {`
	`  receive = UART1_Read();`
	`  }`

Подпрограмма `UARTx_Read_Text`

Прототип	`void UARTx_Read_Text(char *Output, char *Delimiter, char Attempts);`
Результат	Нет.

Описание	Функция читает знаки, полученные через UART пока не будет обнаружен разделитель последовательности. Последовательность чтения хранится в параметре `Output`; последовательность разделителей хранится в параметре `Delimiter`. Это блокирующий вызов: ожидается последовательность разделителей. Иначе процедура прерывается. Параметры: Выход: полученный текст. Разделитель: последовательность символов, которая определяет конец полученной строки. Испытания: определяет количество полученных символов, в котором ожидается появление последовательности разделителей. Если установлено 255 испытаний, эта процедура будет постоянно пытаться обнаружить последовательность разделителя.
Требования	Перед использованием этой функции модуль UART должен быть инициализирован и создана связь.

Пример	Читать текст до получения последовательности "OK" и отправить обратно то, что было получено:
	`UART1_Init(4800);// initialize UART1 module`
	`Delay_ms(100);`
	
	`while (1) {`
	`  if (UART1_Data_Ready() == 1) /* если данные получены*/`
	`{`
	`  UART1_Read_Text(output, "OK", 10);`
	`// читать текст, пока не найден'OK'`
	`  UART1_Write_Text(output);`
	`// послать текст обратно`

Подпрограмма `UARTx_Write`

Прототип	`void UARTx_Write(char_data);`
Результат	Нет

Описание	Функция передает байт через модуль UART. Параметры: `_DATA`: данные, которые будут отправлены.
Требования	Перед использованием этой функции модуль UART должен быть инициализирован.

Пример	`unsigned char _data = 0x1E;`
	`...`
	`UART1_Write(_data);`

Подпрограмма `UARTx_Write_Text`

Прототип	`void UARTx_Write_Text(char * UART_text);`
Результат	Нет
Описание	Посылает текст (параметр `UART_text`) через UART. Текст должен быть ограничен до 255 символов и нулем. Параметры: `UART_text`: текст, который будет отправлен
Требования	Перед использованием этой функции модуль UART должен быть инициализирован и создана связь.

Пример	`UART1_Init(4800);/*инициальзация модуля UART1*/`
	`Delay_ms(100);`
	
	`while (1) {`
	`  if (UART1_Data_Ready() == 1)`
	`// если данные получены`
	`  {  UART1_Read_Text(output, "OK", 10);`
	`// считывать текст пока не найден 'OK'`
	`    UART1_Write_Text(output);`
	`// послать текст обратно`
	`  }`
	`}`

Подпрограмма `UART_Set_Active`

Прототип	`void UART_Set_Active(char (*read_ptr)(),`
	`void (*write_ptr)(unsigned char data_),`
	`char (*ready_ptr)(), char (*tx_idle_ptr)())`

Результат	Нет

Описание	Устанавливает активный модуль UART, который будет использоваться подпрограммами библиотеки.
	Параметры:
	`read_ptr: UARTx_Read обработчик`
	`write_ptr: UARTx_Write обработчик`
	`ready_ptr: UARTx_Data_Ready обработчик`
	`tx_idle_ptr: UARTx_Tx_Idle обработчик`

Требования	Подпрограмма доступна только для микроконтроллеров с двумя модулями UART. Перед использованием этой подпрограммы модуль UART должен быть инициализирован.

Пример	`UART1_Init(9600); // инициализация UART1`
	`UART2_Init(9600); // инициализация UART2`
	`RS485Master_Init(); /* инициализация микроконтроллера как мастера*/`
	
	`UART_Set_Active(&UART1_Read, &UART1_Write, &UART1_Data_Ready, &UART1_Tx_Idle);`
	`// установить UART1 активным`
	`RS485Master_Send(dat,1,160);`
	`// послать сообщение через UART1`
	
	`UART_Set_Active(&UART2_Read, &UART2_Write, &UART2_Data_Ready, &UART2_Tx_Idle);`
	`// установить UART2 активным`
	`RS485Master_Send(dat,1,160);`
	`// послать сообщение через UART2`

9.8. Лабораторная работа №15

Программирование и моделирование передачи и приема данных с использованием модуля USART и виртуального терминала

Цель работы: изучение программирования и моделирования передачи и приема данных с использованием модуля USART и виртуального терминала.

Лабораторное задание

1. Создать в программе Proteus проект LAB15. Разместить проект в папке PRO-LAB15. Собрать схему модели связи микроконтроллета с виртуальным терминалом по интерфейсу UART (рис.9.11). Регистрация поступающих в микроконтроллер данных отображаеися на ЖК-дисплее. Виртуальный терминал выполняет функции клавиатуры и экрана вашего компьютера и позволяет посылать и принимать через интерфейс RS232 асинхронные последовательные данные в модели с микроконтроллером PIC16F877A. Это очень удобно в отладке программ, потому, что Вы сразу видите ссобщения, которые генерируются Вашей программой.

Виртуальный терминал имеет следующие характеристики:

- двунаправленные последовательные данные отображается как символы ASCII при нажатия клавиш клавиатуры и передаются как последовательные данные ASCII.
- простой последовательный интерфейс передачи данных имеет всего два провода: RXD для полученных данных и TXD для передаваемых данных.
- простой двухпроводный интерфейс аппаратного контроля: RTS является выходом и показывает, что терминал готов к приему данных; CTS является входом на который подается высокий уровень перед началом передачи терминала.
- скорость передачи от 300 до 57600 бод.
- 7 или 8 бит данных.
- возможен контроль четности при приеме данных, инвертирование полярности сигналов и т.д.

На рис.9.12 показано окно установки параметров виртуального терминала, с которыми следует выполнять лабораторное задание.

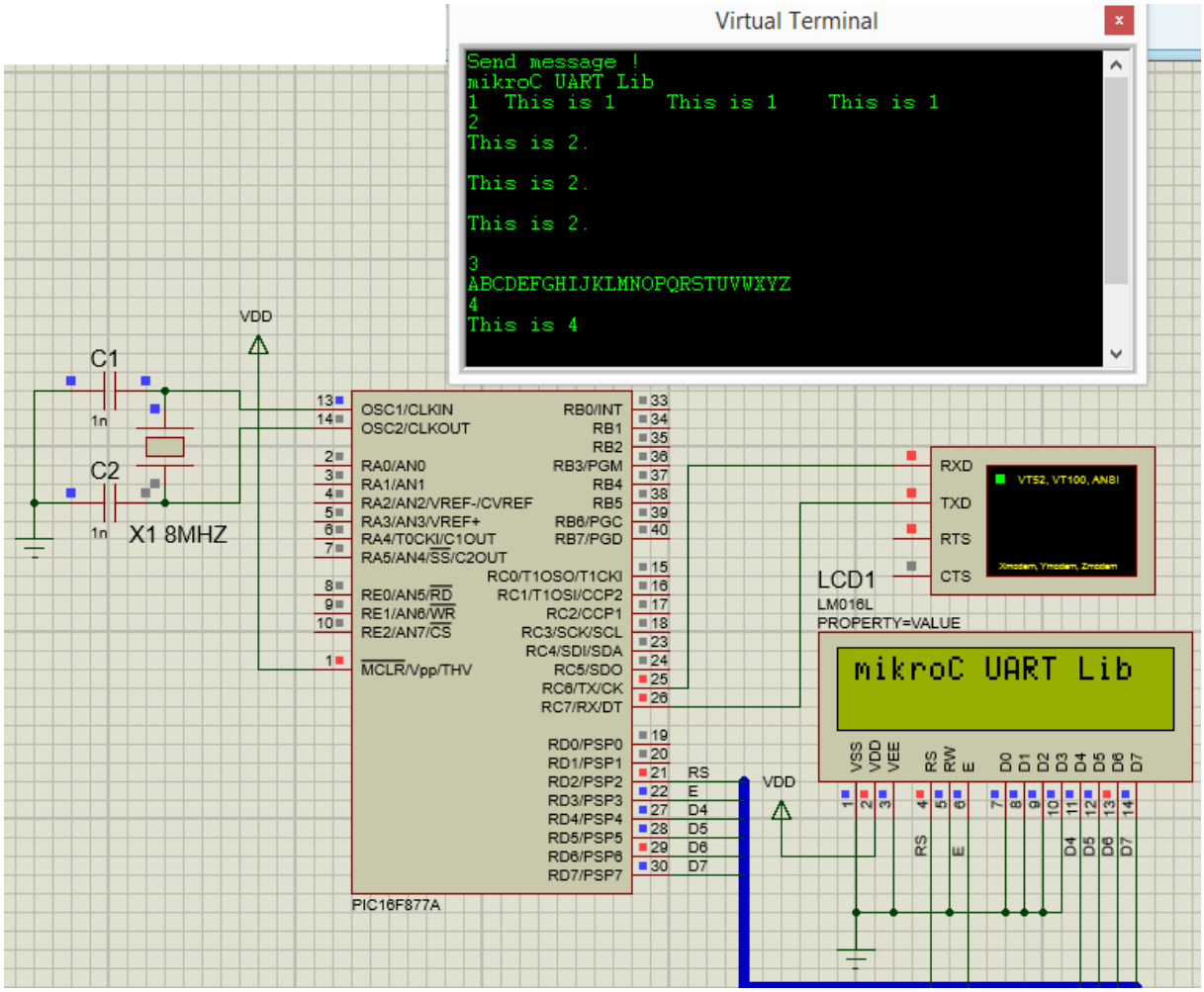


Рис.9.11. Модель интерфейса UART

2. Создать в программе mikroC проект с названием LAB15 и разместить его в папке MKC-LAB15.

3. В окне Code Editor среды mikroC набрать текст программы интерфейса UART. Возможный вариант программы показан в листинге 9.5. Детально проанализировать программу, разобраться в применении библиотечных подпрограмм и технике набора текста.

4. Выполнить компиляцию программы, сделать отладку и устранить ошибки, выявленные в mikroC.

5. Загрузить HEX-файл программы в микроконтроллер модели интерфейса UART в Proteus. Включить моделирование и проверить функционирование модели. Должен открыться экран виртуального терминала и появиться текст «Send message !».

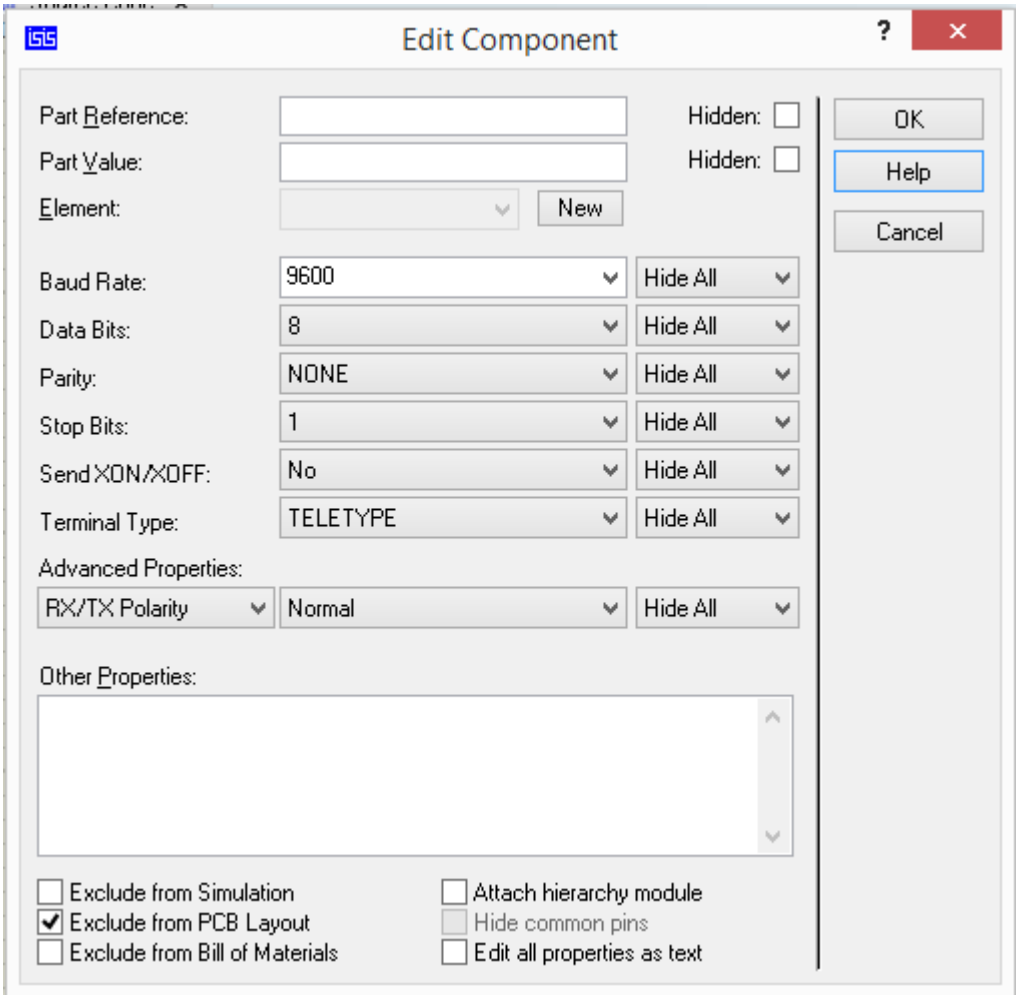


Рис. 9.12. Установка параметров виртуального терминала

Листинг 9.5

```
//USART-LCD.c
char x;
int n, i;
// Модуль подключения ЖК-дисплея

sbit LCD_RS at RD2_bit;
sbit LCD_EN at RD3_bit;
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;
sbit LCD_RS_Direction at TRISD2_bit;
sbit LCD_EN_Direction at TRISD3_bit;
```

```
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;
// Конец модуля подключения ЖК-дисплея

void main() {

  Lcd_Init();   // Инициализация ЖК-дисплея
  Delay_ms(1000);
  UART1_Init(9600); //Инициализация модуля UART
  Delay_ms(500);
  Start:   //Метка повтора программы
  Lcd_Cmd(_LCD_CLEAR); // Очистка ЖК - дисплея
  Lcd_Cmd(_LCD_CURSOR_OFF); // Отключение курсора

  UART1_Write_Text("Send message !\r\n"); /*Посылка
запроса на терминал */
  Delay_ms(100);
  Lcd_Cmd(_LCD_FIRST_ROW); /*Вывод на дисплей в 1-й
строке */
    while(1) {
   if (UART1_Data_Ready() ) /*Проверка готовности
данных к чтению */
   {x=UART1_Read(); //Получение байта через UART
   Delay_ms(100);
   Lcd_Chr_Cp(x); /*Отображение байта в текущей
позиции курсора*/
   Delay_ms(100);
   }
   switch(x)  // Оператор выбора по константе
   {
       case '1': //Первый блок
   {
   for(n=1; n<=3; n++)
      {UART1_Write_Text("  This is 1  "); /*Печатать
три раза в одной строке */
      }
   x=0;
     UART1_Write(13);  //Перевод на новую строку
   }
```

```
       break;

       case '2': /*Второй блок: печатать три раза с
переводом на новую строку с одним интервалом */
    {
       for(n=1; n<=3; n++)
       { UART1_Write(13);
       UART1_Write_Text("This is 2.");
         }
        x=0;
        UART1_Write(13);
        }
       break;

       case '3': /* Третий блок: печатать алфавит в
одной строке*/
       {UART1_Write(13);
        for(i='A'; i<='Z'; i++)
          {
       while(UART1_Tx_Idle()==0);
       UART1_Write(i);

          }
       x=0;
       UART1_Write(13);
       }
       break;

       case '4': /*Четвертый блок: печатать три раза
с переводом на новую строку и с двойный интервалом*/
      {
       for(n=1; n<=3; n++)
       {UART1_Write_Text("\r\n"); /* Перевод в начало
новой строки*/
       UART1_Write_Text("This is 4\r\n");
       }
        x=0;
        }
       break;

       case '5': /* Пятый блок: разное
```

```
        {
         x=0;
        UART1_Write(13);
        UART1_Write(77); /* Печать буквы ‘M’ по коду в
ASCII */
        UART1_Write(13);
        UART1_Write('M'); /* Печать символьной
константы*/
        UART1_Write(13);
        UART1_Write_Text("Goto Start\r\n"); /* Печать
с переводом в начало новый строки */
        goto start;  //Возврат в начало программы
      }
       break;
     }
}
}
```

6. Наберите в терминале английским шрифтом Вашу фамилию. Убедитесь, что текст передан в микроконтроллер и отображен на ЖК-дисплее. Сделайте скриншот модели.

7. Последовательно набирайте в терминале цифры 1, 2, 3, 4 и 5. Наблюдайте и регистрируйте данные, полученные терминалом от микроконтроллера.

8. В пятом блоке программы перед оператором печати «Goto start» введите английскими буквами текст, в котором укажите день, месяц, дату проведения лабораторной работы и фамилии всех членов бригады. Отформатируйте текст так, чтобы он красиво выглядел на терминале. Отошлите текст на терминал и сделайте скриншот.

Домашнее задание

Оформите отчет по работе. Отчет должен содержать программу в mikroC, скриншоты моделирования в Proteus, обсуждение результатов и выводы.

Контрольные вопросы

1. Что такое USART и для чего применяют этот модуль микроконтроллера ?

2. В каких режимах может работать модуль USART ? Поясните особенности режимов работы.

3. Как измеряют скорость передачи данных ?

4. Для чего применяют линейный драйвер при использовании интерфейса RS232 ?

5. Какие регистры используют для управления модулем USART ?

6. Объясните назначение битов регистра управления и статуса передатчика.

7. Объясните назначение битов регистра управления и статуса приемника.

8. Поясните работу модуля USART в режиме асинхронной передачи данных.

9. Поясните работу модуля USART в режиме асинхронного приема данных.

10. Какие подпрограммы из библиотеки mikroC используют для работы с модулем UART и почему изменено название USART ?

11. Поясните применение подпрограмм `UARTx_Init` и `UARTx_Data_Ready`.

12. Поясните применение подпрограмм `UARTx_Read` и `UARTx_Read_Text`.

13. Поясните применение подпрограмм `UARTx_Write` и `UARTx_Write_Text`.

14. Используя «Help» программы Proteus, расскажите о характеристиках и настройке виртуального терминала.

15. Прокомментируйте структуру и работу программы из листинга 9.5.

Библиографический список

1. MPLAB IDE. User's guide with MPLAB Editor and MPLAB SIM Simulator.- http://www.microchip.com/
2. PIC16F84A. Перевод описания. Microchip Technology Inc. 2001.- http://www.microchip.com/.
3. PIC16F87X. Однокристальные 8-разрядные FLASH CMOS микроконтроллеры компании Microchip Technology Incorporated. Перевод. - http://www.microchip.com/.
4. MPLAB XC8 C Compiler. User's Guide. –
- http://www.microchip.com/.
5. . MPLAB XC8 Getting Started Guide. –
- http://www.microchip.com/.
6. mikroC PRO for PIC. User manual.–http:// www.mikroe.com/.
7. TINA. Design Suite. The Complete Electronics Lab for Windows. Quick Start manual. - http://www.designsoftware.com/.
8. Алехин В.А. Электротехника и электроника. Компьютерный лабораторный практикум в программной среде TINA-8.- М.: Горячая линия-Телеком, 2014.-208 с.
9. Proteus Design Suite. ISIS Shematic Capture. Version 8.1.-
- http:// labcenter.com/.
10. Новиков Ю.В., Скоробогатов П.К. Основы микропроцессорной техники. Учебное пособие. –М.: Интернет-Университет информационных технологий. БИИОМ. Лаборатория знаний, 2009 г.-357с.
11. Магда Ю.С. Микроконтроллеры PIC: архитектура и программирование.- М.: ДМК Пресс. 2009.-240 с.
12. Шпак Ю.А. Программирование на языке С для AVR и PIC микроконтроллеров.-Киев.:МК-Пресс; СПб.: Корона-Век. 2011.-546 с.
13. Martin Bates. PIC Microcontrollers. An introduction to microelectronics. Second Edition. 2004. - http://www.general-ebooks.com/book/78249465
14. Martin Bates. Interfacing PIC Microcontrollers: Embedded Design by Interactive Simulation. 2006. - http://www.general-ebooks.com/book/36220807

15. PIC-контроллеры. [Электронный ресурс].- Режим доступа: http://www.technari.ru
16. Брайан Керниган, Деннис Ритчи. Язык программирования Си. -: Издательство: Вильямс. 2015 г.-253 с.
17. Полный справочник по Си. – http://lord-n.narod.ru/download/books/walla/programming/Spr_po_C/main.htm

СОДЕРЖАНИЕ

Приложение 1

Таблица чисел, отображаемых в одном байте

D	B	H	D	B	H	D	B	H	D	B	H
0	0000 0000	00	64	0100 0000	40	128	1000 0000	80	192	1100 0000	C0
1	0000 0001	01	65	0100 0001	41	129	1000 0001	81	193	1100 0001	C1
2	0000 0010	02	66	0100 0010	42	130	1000 0010	82	194	1100 0010	C2
3	0000 0011	03	67	0100 0011	43	131	1000 0011	83	195	1100 0011	C3
4	0000 0100	04	68	0100 0100	44	132	1000 0100	84	196	1100 0100	C4
5	0000 0101	05	69	0100 0101	45	133	1000 0101	85	197	1100 0101	C5
6	0000 0110	06	70	0100 0110	46	134	1000 0110	86	198	1100 0110	C6
7	0000 0111	07	71	0100 0111	47	135	1000 0111	87	199	1100 0111	C7
8	0000 1000	08	72	0100 1000	48	136	1000 1000	88	200	1100 1000	C8
9	0000 1001	09	73	0100 1001	49	137	1000 1001	89	201	1100 1001	C9
10	0000 1010	0A	74	0100 1010	4A	138	1000 1010	8A	202	1100 1010	CA
11	0000 1011	0B	75	0100 1011	4B	139	1000 1011	8B	203	1100 1011	CB
12	0000 1100	0C	76	0100 1100	4C	140	1000 1100	8C	204	1100 1100	CC
13	0000 1101	0D	77	0100 1101	4D	141	1000 1101	8D	205	1100 1101	CD
14	0000 1110	0E	78	0100 1110	4E	142	1000 1110	8E	206	1100 1110	CE
15	0000 1111	0F	79	0100 1111	4F	143	1000 1111	8F	207	1100 1111	CF
16	0001 0000	10	80	0101 0000	50	144	1001 0000	90	208	1101 0000	D0
17	0001 0001	11	81	0101 0001	51	145	1001 0001	91	209	1101 0001	D1
18	0001 0010	12	82	0101 0010	52	146	1001 0010	92	210	1101 0010	D2
19	0001 0011	13	83	0101 0011	53	147	1001 0011	93	211	1101 0011	D3
20	0001 0100	14	84	0101 0100	54	148	1001 0100	94	212	1101 0100	D4
21	0001 0101	15	85	0101 0101	55	149	1001 0101	95	213	1101 0101	D5
22	0001 0110	16	86	0101 0110	56	150	1001 0110	96	214	1101 0110	D6
23	0001 0111	17	87	0101 0111	57	151	1001 0111	97	215	1101 0111	D7
24	0001 1000	18	88	0101 1000	58	152	1001 1000	98	216	1101 1000	D8
25	0001 1001	19	89	0101 1001	59	153	1001 1001	99	217	1101 1001	D9
26	0001 1010	1A	90	0101 1010	5A	154	1001 1010	9A	218	1101 1010	DA
27	0001 1011	1B	91	0101 1011	5B	155	1001 1011	9B	219	1101 1011	DB
28	0001 1100	1C	92	0101 1100	5C	156	1001 1100	9C	220	1101 1100	DC
29	0001 1101	1D	93	0101 1101	5D	157	1001 1101	9D	221	1101 1101	DD
30	0001 1110	1E	94	0101 1110	5E	158	1001 1110	9E	222	1101 1110	DE
31	0001 1111	1F	95	0101 1111	5F	159	1001 1111	9F	223	1101 1111	DF
32	0010 0000	20	96	0110 0000	60	160	1010 0000	A0	224	1110 0000	E0
33	0010 0001	21	97	0110 0001	61	161	1010 0001	A1	225	1110 0001	E1
34	0010 0010	22	98	0110 0010	62	162	1010 0010	A2	226	1110 0010	E2
35	0010 0011	23	99	0110 0011	63	163	1010 0011	A3	227	1110 0011	E3
36	0010 0100	24	100	0110 0100	64	164	1010 0100	A4	228	1110 0100	E4
37	0010 0101	25	101	0110 0101	65	165	1010 0101	A5	229	1110 0101	E5
38	0010 0110	26	102	0110 0110	66	166	1010 0110	A6	230	1110 0110	E6
39	0010 0111	27	103	0110 0111	67	167	1010 0111	A7	231	1110 0111	E7
40	0010 1000	28	104	0110 1000	68	168	1010 1000	A8	232	1110 1000	E8
41	0010 1001	29	105	0110 1001	69	169	1010 1001	A9	233	1110 1001	E9
42	0010 1010	2A	106	0110 1010	6A	170	1010 1010	AA	234	1110 1010	EA
43	0010 1011	2B	107	0110 1011	6B	171	1010 1011	AB	235	1110 1011	EB
44	0010 1100	2C	108	0110 1100	6C	172	1010 1100	AC	236	1110 1100	EC
45	0010 1101	2D	109	0110 1101	6D	173	1010 1101	AD	237	1110 1101	ED
46	0010 1110	2E	110	0110 1110	6E	174	1010 1110	AE	238	1110 1110	EE
47	0010 1111	2F	111	0110 1111	6F	175	1010 1111	AF	239	1110 1111	EF
48	0011 0000	30	112	0111 0000	70	176	1011 0000	B0	240	1111 0000	F0
49	0011 0001	31	113	0111 0001	71	177	1011 0001	B1	241	1111 0001	F1
50	0011 0010	32	114	0111 0010	72	178	1011 0010	B2	242	1111 0010	F2
51	0011 0011	33	115	0111 0011	73	179	1011 0011	B3	243	1111 0011	F3
52	0011 0100	34	116	0111 0100	74	180	1011 0100	B4	244	1111 0100	F4
53	0011 0101	35	117	0111 0101	75	181	1011 0101	B5	245	1111 0101	F5
54	0011 0110	36	118	0111 0110	76	182	1011 0110	B6	246	1111 0110	F6
55	0011 0111	37	119	0111 0111	77	183	1011 0111	B7	247	1111 0111	F7
56	0011 1000	38	120	0111 1000	78	184	1011 1000	B8	248	1111 1000	F8
57	0011 1001	39	121	0111 1001	79	185	1011 1001	B9	249	1111 1001	F9
58	0011 1010	3A	122	0111 1010	7A	186	1011 1010	BA	250	1111 1010	FA
59	0011 1011	3B	123	0111 1011	7B	187	1011 1011	BB	251	1111 1011	FB
60	0011 1100	3C	124	0111 1100	7C	188	1011 1100	BC	252	1111 1100	FC
61	0011 1101	3D	125	0111 1101	7D	189	1011 1101	BD	253	1111 1101	FD
62	0011 1110	3E	126	0111 1110	7E	190	1011 1110	BE	254	1111 1110	FE
63	0011 1111	3F	127	0111 1111	7F	191	1011 1111	BF	255	1111 1111	FF