МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. АБУ РАЙХАНА БЕРУНИ

Факультет «Электроника и автоматика»

Кафедра «Приборостроение»

На правах рукописи

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

для получения степени бакалавра

по направлению 5521500 «Приборостроение»

Валетов Ильдар Ильдусович

Тема: “Разработка физического и виртуального лабораторных стендов для изучения работы микроконтроллеров”.

Заведующий кафедры: доц. С.А. Васильева

Руководитель ст.преп. А.Х. Хайдаров

Ташкент – 2015

АННОТАЦИЯ

В данной выпускной квалификационной работе рассмотрено создание физического и виртуального лабораторных стендов на базе микроконтроллеров PIC. Физический стенд представляет собой печатную плату с наличием портов ввода/вывода и элементами индикации. Виртуальный лабораторный стенд реализован в программе PROTEUS VSM и идентичен физическому стенду. Для детального изучения устройства микроконтроллера и основ его программирования представлено 5 лабораторных работ.

Выпускная квалификационная работа состоит из общей и основной частей, а также, экономической части, безопасности жизнедеятельности, заключения и списка используемой литературы.

В первой части рассмотрены схемотехника микроконтроллеров PIC, их взаимодействие с периферией, а также основные команды, используемые при программировании. Представлены теоретические сведения по работе с такими программами, как PROTEUS VSM, MPLAB и PicKit 2 Programmer.

Во второй части описывается устройство лабораторного стенда, приводится схема электрическая принципиальная и представляется 5 лабораторных работ.

Следующие части выпускной квалификационной работы посвящены безопасности жизнедеятельности и организационно-экономическим вопросам.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Для решения задач ускоренного развития экономики Республики Узбекистан необходим научный и инновационный прорыв во всех основных отраслях экономики. Такой прорыв невозможен без широкого внедрения микропроцессоров и микроконтроллеров. Важнейшее значение имеет также процесс обучения специалистов навыкам работы с микроконтроллерной техникой.

Цель выпускной работы: разработка и изготовление лабораторного стенда для отладки периферийных схем и микроконтроллеров серии PIC предназначенной для ознакомления студентов с типовыми схемными решениями микроконтроллерных устройств различного назначения.

Лабораторный стенд для микроконтроллеров серии PIC типа 16F873A/876A должен обеспечивать индикацию состояния всех выводов портов и работу с типовыми периферийными устройствами.

Технико-экономическая эффективность разработки определяется необходимостью широкого развития микропроцессорной техники, необходимой для успешного инновационного развития экономики Республики.

1. ОБЗОРНАЯ ЧАСТЬ

1.1.Микроконтроллер PIC

1.1.1 Схемотехника применения PIC-контроллеров

Что бы изделие на основе PIC-контроллеров правильно работало, нужно соблюдать некоторые правила: обеспечение качественным питанием, уменьшение электромагнитные помехи на его выводах, не допускать перегрузок и т.д. Остановимся подробнее на некоторых моментах. Необходимо ставить керамический конденсатор 0,1 мкФ вблизи выводов питания (рисунок 1.1.1.1), добавлять последовательно резистор для защиты от перегрузки резонаторов с низким уровнем возбуждения[3].

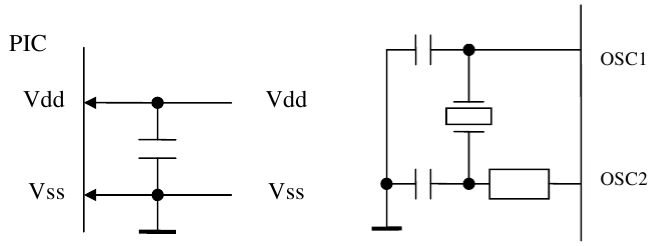

Рис 1.1.1.1. Схема подключения резонатора

На рисунке 1.1.1.2 приведена типовая схема внешнего сброса. Данная схема необходима, если скорость нарастания напряжения питания недостаточна (менее 0,05 В/мс). Диод используется для разряда конденсатора при выключении питания. Резистор может иметь значение от 100 до 1000 Ом и предназначен для ограничения тока по выводу MCLR' при перезаряде конденсатора.

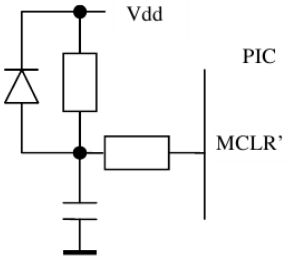

Рис.1.1.1.2. Типовая схема внешнего сброса

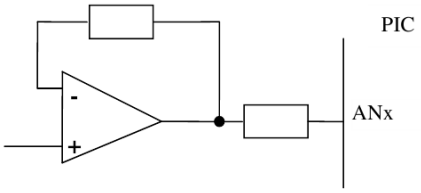

Рис.1.1.1.3. Схема подключения буферного усилителя

На рисунке 1.1.1.3 показано подключение буферного усилителя для аналоговых измерений. Операционный усилитель нужно брать с однополярным питанием и работой входа/выхода от потенциала земли до потенциала питания[5].

1.1.2. Команды микроконтроллеров PIC

Каждая команда состоит из двух компонентов: кода операции и аргументов. Аргументы могут быть константами, адресами или прочими параметрами. Оба компонента заключены в одном слове команды, разряды которого используются оптимально, поскольку отсутствует выделение строго определенного количества разрядов под код операции и аргументы. Так, в некоторых командах аргументы очень длинные, в то время как в других командах их вообще нет. В качестве примера рассмотрим команды, выполняющие логические и арифметические операции с аргументом данных и рабочим регистром W. Этот класс команд имеет следующую структуру:

Таблица 1.1.2.1.

12 разрядов	00000,D,FFFF
14 разрядов	00000,D,FFFFFF
16 разрядов	000000, D,FFFFFFFF

При этом «0» обозначает код операции, «D» - селектор регистра назначения, а «F» - разряды указателя на аргумент данных.

Очень удобно в этих командах то, что пунктом назначения для результата операции может быть, на выбор, регистр W или указанный в

аргументе файловый регистр. В языке ассемблера, поддерживаемом компанией Microchip, такая команда записывается следующим образом[4]:

ADDWF WERT,W

Этой командой содержимое ячейки памяти с именем WERT суммируется с рабочим регистром W и результат помещается в рабочий регистр W.

Всего в системе команд среднего подсемейства PIC-контроллеров 35 кодов операции, есть все, что требуется для программирования[5]:

Таблица 1.1.2.2. Команды с файловым регистром F в качестве аргумента

Команды+ операнды	***Операция***	***Флаги состояния***
ADDWF F,D	Add (W) and (F) (D)=(W) + (F)	C, DC, Z
ANDWF F,D	AND W with F (D)=(W) AND (F)	z
CLRF F	Clear (F) (F)=0	z
CLRW	Clear (W) (W)=0	z
COMF F,D	Complement (F) (D)=NOT (F)	z
DECF F,D	Decrement (F) (D)=(F) - 1	z
DECFSZ F,D	Decrement (F), Skip if 0 (D)=(F) - 1, пропускаем следующую команду, если Z= 1	не изменяются
INCF F,D	Increment (F) (D)=(F) + 1	Z

INCFSZ F,D	Increment (F), Skip if 0 (D)=(F) + 1, пропускаем следующую команду, если Z=l	не изменяются
IORWF F,D	Inclusive OR (W) with (F) (D)=(W) OR (F)	Z
MOVF F,D	Move f (D)=(F)	Z
MOVWF F,D	Move W to f (F)=(W)	не изменяются
NOP	No Operation Нет операции	не изменяются
RLF F,D	Rotate Left (F) through Carry Циклический сдвиг влево через C	C
RRF F,D	Rotate Right (F) through Carry Циклический сдвиг вправо через C	C
SUBWF F,D	Subtract (W) from (F) (D)=(F) - (W)	C, DC, Z
SWAPF F,D	Swap nibbles in (F) Переставить полубайты b(F) (D)=(F)	не изменяются
XORWF F,D	Exclusive OR W with f (D)=(W) XOR (F)	Z

Таблица 1.1.2.3. Команды с адресами разрядов в качестве аргументов; B=0...7

Команды+ операнды	***Операция***	***Флаги состояния***

BCF F, B	Bit Clear (F) Очистить бит B в (F)	не изменяются
BSF F, B	Bit Set (F) Установить бит B в (F)	не изменяются
BTFSC F,B	Bit Test (F), Skip if Clear Пропустить следующую команду, если бит B=() в (F)	не изменяются
BTFSS F,B	Bit Test (F), Skip if Set Пропустить следующую команду, если бит B=1 в (F)	не изменяются

Таблица 1.1.2.4. Команды с константами, команды перехода и другие

Команды+ операнды	***Операция***	***Флаги состояния***
ADDLW K	Add literal and W (W)=(W)+K	C, DC, Z
ANDLW K	AND literal with W (W)=(W) AND K	Z
CALL ADDR	Call subroutine Вызов подпрограммы	не изменяются
CLRWDT	Clear Watchdog Timer Сброс сторожевого таймера	TO, PD
GOTO ADDR	Go to address Переход по адресу	не изменяются
IORLW K	Inclusive OR literal with W (W)=(W) OR K	Z
MOVLW K	Move literal to W (W)=K	не изменяются
RETFIE	Return from interrupt Выход из подпрограммы обработки	GIE

	прерывания	
RETLW K	Return with literal in W Возврат с (W)=K	не изменяются
RETURN	Return from Subroutine Возврат	не изменяются
SLEEP	Go into standby mode Переход в спящий режим	TO, PD
SUBLW K	Subtract W from literal (W)=K - (W)	C, DC, Z
XORLW K	Exclusive OR literal with W (W)=(W) XOR K	Z

Примечание: скобки означают содержимое того регистра, имя которого заключено в скобки

Команды очень хорошо документированы в фирменных технических описаниях, где можно найти даже коды операций.

В описании команд следует всегда обращать внимание на колонку, в которой отмечены флаги, изменяемые в результате выполнения той или иной команды.

Флаги находятся в регистре состояния STATUS.

1.1.3. Области памяти в МК.

Каждый МК, который мы будем рассматривать на практике, имеет три области памяти[6]:

1. память программ или флеш-память, область, куда записываются строчки текста программы в момент «прошивания» МК;
2. оперативная память или регистровая память, область, в которую во время работы МК записываются, хранятся и изменяются байты, пока на МК подается питание. Сброс питания приводит к сбросу оперативной памяти;

3. энергонезависимая память ПЗУ или EEPROM, область, в которую в момент прошивания и/или во время работы МК записываются, хранятся и изменяются данные. Сброс питания не влияет на содержимое этой памяти.
4. Области памяти состоят из т.н. ячеек, в которых хранятся байты. Байты и биты.

Байт - это восемь бит. Бит - это минимальный элемент информации, который может быть равен нулю или единице. Комбинация из восьми битов составляет байт, т.е. восьмиразрядное бинарное число, например 10001101. Всего на восьми разрядах из нулей и единиц можно составить 256 комбинаций. Числа в МК хранятся в ячейках, размерностью один байт. МК оперирует байтами и поэтому называется восьмиразрядный. Закономерен вопрос - а если нам нужно больше чем 256 комбинаций? В таком случае используется несколько байтов. Например, набор единиц и нулей в двух байтах даст уже 256*256=65536 комбинаций. И т.д.

1.1.4. МК и текст программы. Типы данных. Переменные и константы.

Мы знаем, что текст программы записывается во флэш-область во время «прошивания» кристалла. Необходимо понимать, что текст программы не меняется и не изменится, пока мы во флэш не запишем новый текст. И именно в тексте программы мы говорим, что нужно делать контроллеру - установить на ножке исходящий сигнал или принять внешний сигнал. Работа с сигналами это лишь небольшая часть работы МК, но самая важная. Другая часть работы заключается в математическом обсчете принятых данных или математическом обсчете исходящих данных. Вот мы и подошли к ключевому термину – «Д А Н Н Ы Е ».

Типы данных - это не что иное, как число. Ну, например, число входящих сигналов за определенное время. Или число, характеризующее количество времени между исходящими сигналами. Данные это числа и

комбинации чисел. А какие числа/данные можно на Си обсчитывать? Отвечаем - определенной размерности или определенных типов.

Таблица 1.1.4.1. Типы данных

Тип	***Размер байт***	***Диапазон***
bit (бит)	1/8	0,1
char (символ)	1	-128 ... 127
unsigned shar (символ без знака)	1	0 ... 255
int (целое)	2	-32768 ... 32767
unsigned int (целое без знака)	2	0 ... 65535
long int (длинное целое)	4	-2147483648 ... 2147483647
unsigned long int (длинное целое без знака)	4	0 ... 4294967295
float (с плавающей точкой)	4	±1,175e-38... ±3,402e38

Т.е. бит может принимать одно из двух значений 1 или 0. Символ - это байт с соответствующим диапазоном значений. Под «целое» выделяется два байта. «Длинному целому» предоставляется четыре байта.

Переменные и константы.

Мы говорили о числах, которые могут записываться, храниться и изменяться. Числа, которые могут изменяться называют переменными. ***А что такое константы?*** Это жестко прописанные числа. А где мы жестко прописываем информацию, т.е. не можем её изменить? Правильно во флеш-памяти, а иначе говоря, в памяти программ. Чтобы проще усвоить понятия о переменных и константах рассмотрим выражение

x+y=25, где x и y это переменные, а 25 - это константа. Лучше это так понимать.

1.1.5. Регистры портов. Определение направлений работы линий.

Что такое регистр? Регистр - это ячейка памяти в МК (в области ОЗУ). Если это ячейка памяти, то в неё можно записать один байт информации. А байт это некое число, которое лежит в десятичном диапазоне 0...255. Некоторые регистры/ячейки имеются определенные имена и отвечают за тот или иной режим работы МК, что очень подробно расписывается в даташите.

1.1.6. Команда TRIS

Рассмотрим строчку TRISB = 0Ь00010000;

Мы видим, что регистр TRISB приравнивается чему-то. Вы наверняка слышали о некоторых системах счисления: двоичная, десятичная, шестнадцатеричная. И чтобы компилятор понимал, какое число в данном случае употребляем, существуют определенные форматы записи[7].

127 - обычное десятичное число 127;

0x7F - число 127 в 16-ричной системе счисления;

0b01111111 - число 127 в двоичной системе счисления.

Если нет приставки - то это десятичное число, если префикс

0x - то это 16-ричное число, если префикс

0b - то это двоичное число, либо воспользуйтесь стандартным калькулятором Windows (вид - инженерный).

Почему мы TRISB приравниваем некому двоичному числу? Для ответа на этот вопрос сначала заучим правило: биты в байте нумеруются справа налево от нуля до семи. Каждый бит в регистре/ячейке TRISB отвечает за направление работы соответствующей ножки на порту B. Например, нулевой бит отвечает за направление работы ножки RB0.

Фраза TRISB = 0Ь00010000; говорит нам, что четвертый бит в этом регистре будет равен 1. Использование двоичной записи гораздо нагляднее показывает направление работы каждой ножки.

А как запомнить - какой бит (0 или 1) нужен для работы на вход или для работы на выход? Мы говорили, что ножки работают на вход или на выход (вход/выход - на англ. Input/Output или 10 - наверняка вы это слышали в аббревиатуре BIOS- базовая система ввода/вывода). Внешне буквы I/O похожи на цифры 1/0. Цифра 1 - это вход (Input), а цифра 0 - это выход (Output).

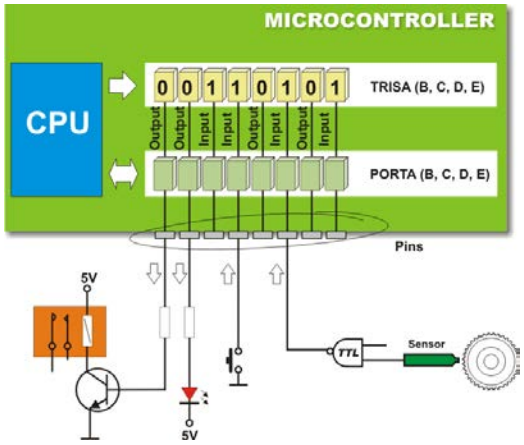

Рис.1.1.6.1. Установка портов на ввод/вывод

Итак, гладя на фразу TRISB = 0Ь00010000; мы видим, что 4й бит равен единице. А это значит, что RB4 будет input, т.е. цифровой линией, работающей на вход. Если бы мы записали TRISB=0Ь00100010; то этим мы скажем, что RB0-RB2-RB3-RB4-RB6-RB7 работают на выход, RB1 и RB5 на вход.

```
TRISA = 0Ь00000000; // направление работы ножек порта А
TRISB = 0Ь00010000;   // направление работы ножек порта В
```

Далее у нас две строчки

```
PORTA =0; // очищаем порт А
PORTB =0;   //  очищаем порт Б
```

Для чего нужна очистка? Представим ситуацию - мы подаем питание на МК. Начинается выполнение программы. Выполнились строчки определения направления. И что после этого? На ножках, которые работают на выход, может быть любое состояние сигнала. Признаком хорошего тона является именно в этом месте программы, в самом начале установить на ножках НЛУ (если обратного не требует логика работы устройства). Это как минимум экономит энергию, а как максимум даёт вам уверенность в определенности состояния сигналов на ножках.

Что такое очистка? Это прописывание нуля в регистр или установка конкретного бита в ноль. Эти две строчки можно было бы написать и в двоичном формате

PORTA =0Ь00000000; // очищаем порт А

PORTB = 0Ь00000000; // очищаем порт Б

Это означает, на всех цифровых линиях, которые работают на выход, будет установлен НЛУ. Если бы мы записали PORTB = 0b100l000l то на RBO, RB4 и RB7 будет установлен ВЛУ (при условии, если эти ножки будут настроены на выход).

1.1.7. Команда ADCON

Синтаксис

ADCIN *Channel, Var*

По этой команде считывается значение потенциала на входе выбранного канала АЦП, преобразуется в цифровой код, и результат сохраняется в указанной переменной. И хотя доступ к регистрам аналого-цифрового преобразователя можно получить непосредственно, команда ADCIN делает этот процесс несколько проще[8].

Прежде, чем Вы будете использовать команду ADCIN, необходимо соответствующий разряд регистра TRIS установить так, чтобы он стал входом. В регистре ADCON1 также необходимо установить

соответствующий разряд так чтобы вывод канала АЦП перевести в режим аналогового входа, а в некоторых случаях установить формат результата и источник синхронизации.

1.1.8. Команда ADCIN

Синтаксис

ADCIN *Channel,Var*

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

Прежде, чем Вы будете использовать команду **ADCIN**, необходимо соответствующий разряд регистра **TRIS** установить так, чтобы он стал входом. В регистре **ADCON1** также необходимо установить соответствующий разряд так чтобы вывод канала АЦП перевести в режим аналогового входа, а в некоторых случаях установить формат результата и источник синхронизации. См. справочные листы данных фирмы Microchip для получения дополнительной информации об этих регистрах и настройке их для определенного устройства. Для этих же целей могут использоваться несколько команд **DEFINE**[9]**.**

Пример:

```
DEFINE ADC_BITS 8        /определяем разрядность преобразования
DEFINE ADC_CLOCK 3       /устанавливаем источник синхронизации (rc = 3)
DEFINE ADC_SAMPLEUS 50   /установим время выборки в микросекундах
TRISA = 255      / установим все разряды PORTA на вход
```

ADCON1 = 2 /определяем режим PORTA как аналоговый вход

ADCIN 0, B0 /читаем канал 0, результат записываем в переменную B0

1.1.9. Команда CLEAR

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

Пример:

CLEAR *'установить все переменные в О*

1.1.10. Команда CLEARWDT

Обнулить сторожевой таймер.

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

Пример:

CLEARWDT ' *сбросить сторожевой таймер*

1.1.11. Команда PAUSE

Синтаксис: *PAUSE Period*

Эта команда приостанавливает работу программы на время — Period. Параметр Period задается в миллисекундах[10]. Значение периода - 16-разрядное число; таким образом задержки могут быть до 65 535 миллисекунд (немногим более, чем минута). В отличие от других

функций задержки (NAP и SLEEP), PAUSE не переводит микроконтроллер в режим малого энергопотребления. Таким образом, в режиме PAUSE микроконтроллер потребляет больше мощности, но этот режим также намного более точен. Он работает с той же точностью, что и задающий генератор системы. По умолчанию команда PAUSE предполагает частоту задающего генератора 4 МГц. Если используется другой генератор, то необходимо компилятору PicBasicPro сообщить об этом в определении DEFINE.

Пример:

PAUSE 1000 /*Задержка на 1 сек*

1.1.12. Команда LCDIN

Синтаксис:

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

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

ЖК мониторы имеют встроенное ОЗУ, которое используется для хранения выводимых знаков. Большинство ЖК мониторов имеет большое доступное ОЗУ, которое является необходимым для визуализируемой области. Данные в это ОЗУ могут быть записаны, используя команду **LCDOUT**. Команда же **LCDIN** позволяет считывать эту информацию из ОЗУ[5].

Знакогенератор ОЗУ работает с адресами от $40 до $7f. Стартовый адрес данных ОЗУ дисплея начинается с адреса $80. Смотрите справочные листы данных для определенного типа ЖК монитора относительно его функций и адресов.

Для работы этой команды необходимо соединить цепь чтение/запись (R/W) ЖК монитора с соответствующим выводом PIC контроллера.

Состояние ее, О или 1, определяет, то какая команда в данный момент используется — чтения **(LCDIN)** или записи **(LCDOUT).** А в начале программы записать два **DEFINE** указывающие на адрес вывода:

DEFINE LCD_RWREG PORTE /Подключить цепь R/W ЖК монитора к

PORTE

DEFINE LCDRWBIT 2 /Подключить цепь R/W ЖК монитора к выводу 2

1.1.13. Команда LCDOUT монитора.

Синтаксис:

LCDOUT *Item {,Item...}*

Эта команда выводит информацию на экран алфавитно-цифрового ЖК монитора. Компилятор PicBasicPro поддерживает модули ЖК мониторов с контроллерами типа Hitachi 44780 или эквивалентными ему. Эти мониторы обычно имеют 14 или 16 контактов - или двухрядный разъем[5].

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

Может потребоваться относительно длительное время для запуска ЖК монитора после включения питания. Поэтому программа должна выждать, по крайней мере, 0.5 секунды перед посылкой первой команды на ЖК монитор. В таблице 1.1.13.1 перечислены некоторые полезные команды, в которых присутствует код $FE.

Таблица 1.1.13.1. Команды для ЖК-дисплея.

Команда	***Операция***
$FE, 1	Очистить дисплей
$FE, 2	Вернутся домой (идти на первую строку и первое знакоместо)
$FE, $0C	Выключить курсор
$FE, $0E	Курсор в форме подчеркивания
$FE, $0F	Включить мигающий курсор
$FE,$10	Сдвинуть курсор влево на одну позицию
$FE,$14	Сдвинуть курсор вправо на одну позицию
$FE,$C0	Переместить курсор на начало второй строки
$FE, $94	Переместить курсор на начало третьей строки
$FE, $D4	Переместить курсор на начало четвертой строки

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

LCDOUT $FE, $C0

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

Экраны с 4 строками также имеют перепутанную карту памяти. Поэтому смотрите справочные листы данных для каждого конкретного

ЖК монитора относительно расположения в памяти знаков и дополнительных команд.

Пример:

LCDOUT $FE,1,"Hello" / Очистить экран и вывести "Hello"

LCDOUT BO, #B1

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

/Определим порт данных

DEFINE LCD DREG PORTB / Установим, стартовый информационный разряд (0 или 4), если 4-разрядная шина

DEFINE LCDDBIT 4 / Установим порт к которому подключается цепь Register Select

DEFINE LCD RSREG PORTB /Установим разряд порта к которому подключается цепь Register Select

DEFINE LCDRSBIT 1 / Установим порт к которому подключается цепь Enable **DEFINE LCDEREG PORTB /**Установим разряд порта к которому подключается цепь Enable

DEFINE LCDEBIT O /Установим тип шины (4 или 8 разрядов)

DEFINE LCDBITS 4 /Установил! тип LCD (количество строк на экране)

DEFINE LCDLINES 2 / Установить команду задержки вмкс

DEFINE LCDCOMMANDUS 2000 / Установить время задержки данных в мкс

DEFINE LCD DATAUS 50

Эти установки сообщат PicBasicPro, что используется ЖК монитор с 2 строками, связанный 4-разрядной шиной данных подключенной к старшим 4 разрядам PORTB, цепь Select Register подключена к **PORTB**. 1, а цепь Enable к **PORTB.O** (см. рисунок 1.1.13.1)[1].

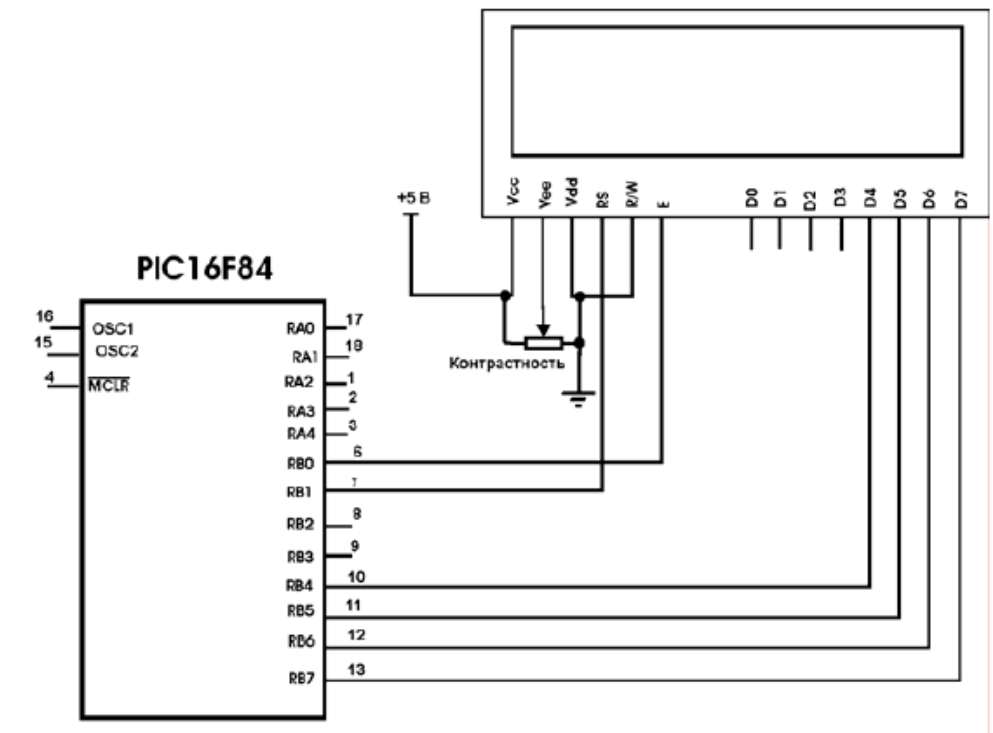

Рис.1.1.13.1 Вариант схемы подключения алфавитно-цифрового ЖКИ к PIC16F84 с использованием 4-проводной шины.

1.1.14. Управление ЖКИ

ЖКИ или ЖК монитор часто используются для вывода информации в удобной для восприятия человеком форме. На рисунке 1.1.14.1 представлен один из таких индикаторов.

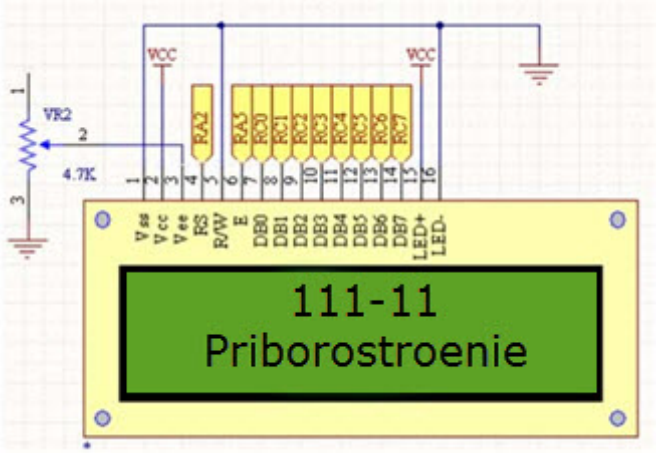

Рис.1.1.14.1. Подключение ЖК-монитора 1602

Существует большое количество различных типов ЖКИ, но в основном (в 99%) распространены индикаторы с интерфейсом Hitachi 44780. Что же представляют собой эти индикаторы? Большинство таких индикаторов имеет 14 выводов. Назначение этих выводов приведено в таблице[1].

Таблица 1.1.14.1. Назначение выводов ЖК-дисплея

Вывод	***Обозначение***	***Назначение***
1	Vdd	Общий
2	Vcc	+ Питание
3	Contrast (Vee)	Регулировка контрастности
4	R/S	Команда/Выбор регистра
5	R/W	Чтение/ Запись
6	E	Тактовые импульсы
7-14	Data	Данные: DO -7,...,D7 -14

Запись информации происходит в параллельном коде по фронту тактовых импульсов **E**. Можно не только записывать данные в регистры ЖКИ, но и читать их. Для этого используется цепь **R/W**. Для подключения шины данных можно использовать 2 режима:

Это режим 8-разрядной шины, когда данные в контроллер ЖКИ передаются байтами, или режим 4-разрядной шины, когда вначале передаются старшие 4 разряда байта, а затем младшие 4 разряда. Хотя понятно, что при 4-разрядной шине количество соединений меньше и загруженность портов меньше, но при этом и скорость обмена меньше.

Цепь **R/S** предназначена для указания типа информации, которая в данный момент подается на выводы данных. Если на линии **R/S** установлена логическая 1, то на шине данных можно устанавливать ASCII код, символ которого будет отображаться в текущей позиции курсора. Если же на линии R/S будет установлен логический 0, а в цепи R/W логическая 1, то можно считать код символа, отображаемого в данный момент на экране. Хотя в большинстве конструкций вывод R/W подключается к общей шине питания, так как нет необходимости использовать режим чтения. На выполнение команд ЖКИ затрачивается разное время. Например, для стирания экрана или на перемещение курсора в начальную позицию требуется около 4,1 мс, а на другие команды достаточно 160 мкс. Запись данных производится так же, как и запись команд, только в первом случае по цепи R/S должна быть подана 1.

1.2. PROTEUS VSM

Система виртуального моделирования

Proteus VSM, созданная фирмой Labcenter Electronics на основе ядра SPICE3F5 университета Berkeley, является так называемой средой сквозного проектирования. Это означает создание устройства, начиная с его графического изображения (принципиальной схемы) и заканчивая

изготовления печатной платы устройства, с возможностью контроля на каждом этапе производства[11].

В «сферу-влияний» PROTEUS VSM входят как простейшие аналоговые устройства так и сложные системы созданные на популярных ныне микроконтроллерах. Доступна огромная библиотека моделей элементов, пополнять которую может сам пользователь. Возможность анимации схем позволяет программе стать прекрасным учебным пособием на уроках в школе и ВУЗ. 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. ARES это трассировщик печатных плат с возможностью создания своих библиотек корпусов.

Основной программой является ISIS, в ней предусмотрена горячая связь с ARES для передачи проекта для разводки платы.

При запуске программы появляется основное окно.

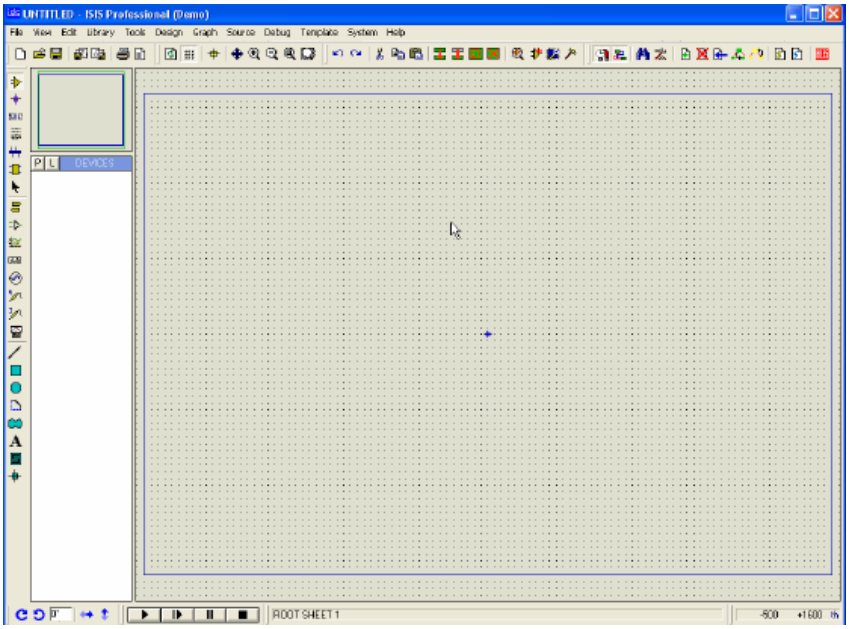

Рис. 1.2.1. Главное окно программы Proteus ISIS

Самое большое пространство отведено под окно редактирования EDIT WINDOW.

Именно в нем происходят все основные процессы создания, редактирования и отладки схемы устройства.

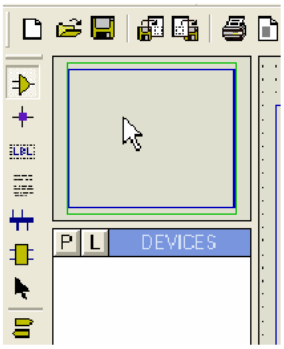

Рис. 1.2.2. Окно редактирования

Слева вверху маленькое окно предварительного просмотра Overview Window с ее помощью можно перемещаться по окну редактирования (щелкая левой кнопкой мыши по окну предварительного просмотра, мы перемещаем окно редактирования по схеме, если конечно схема не вмещается в окно).

Перемещать окно редактирования по схеме можно еще так - удерживая нажатой кнопку SHIFT двигать курсор мыши, не нажимая ее кнопок, по окну редактирования.

Приближать и отдалять схему в окне можно соответственно кнопками F6 и F7 или же колесом мыши, F5 центрирует схему в окне а нажатие F8 подгоняет размер схемы под окно редактирование.

Под окном предварительного просмотра находится Object Selector список выбранных в данный момент компонентов, символов и других элементов. Выделенный в списке объект отображается в окне предварительного просмотра.

Все возможные функции и инструменты Proteus VSM доступны через меню расположенное в самом верху основного окна программы, через

пиктограммы находящиеся под меню и в левом углу основного окна и через горячие клавиши, которые могут переназначаться пользователем.

Рис. 1.2.3.Строка состояния

В самом низу основного окна расположены: слева направо кнопки вращения и разворота объекта вокруг своей оси, панель управления интерактивной симуляцией (выглядит как магнитофонная и функции такие же: ПУСК-ПОШАГОВЫЙ РЕЖИМ ПАУЗА-СТОП) ,

Рис. 1.2.4. Элементы управления симуляцией

строка статуса (о ней отображаются: ошибки, подсказки, текущее состояние процесса симуляции и т.д.) и координаты курсора отображаемые в дюймах[16].

Для того чтобы манипулировать объектами их нужно сначала выделить, это можно сделать только на остановленном проекте. Для выделения одного объекта надо щелкнуть по нему правой кнопкой мыши. Див выделения группы можно либо удерживая CTRL последовательно щелкать правой кнопкой по всем объектам либо удерживая правую кнопку протащить область выделения по необходимым объектам. Выделять объекты надо осторожно, повторный щелчок правой кнопкой мыши по выделенному объекту удалит его, (удалить выделенные объекты можно еще, нажав кнопку DELETE) .

Но это не страшно отменить последние и все предыдущие действия по порядку можно с помощью кнопок отмены (UNDO, REDO).

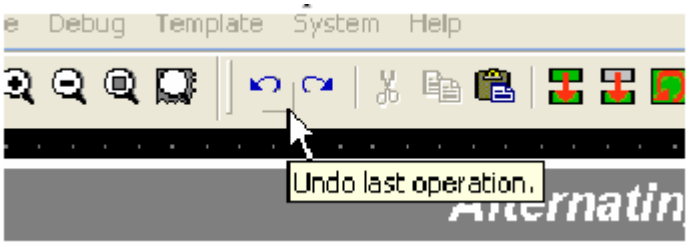

Рис. 1.2.5. Отмена последнего действия

Кнопки отмены действуют как назад, по хронологии так и вперед.

Выделенные объекты можно перемещать по схеме, ухватив их левой кнопкой мыши передвинув в нужное место отпустить кнопку. А с помощью этих кнопок производятся групповые операции с выделенными объектами. По порядку: копирование, перемещение, поворот и удаление.

Рис. 1.2.6. Операции над объектами

Мы овладели необходимым минимумом, настало время создания своих проектов. Создайте новый проект, используя меню FILE > NEW DESIGN. Это можно не делать, если вы только что открыли программу, ток как при запуске PROTEUS автоматически создаст новый проект с именем UNTITLED.DSN - безымянный.

Установим для удобства свои размеры листа схемы, Откроем меню SYSTEM > SET SHEET SIZE (Установить размеры листа). Выберем вариант USER пользовательский, в окошках введем 6 in 4 in (высота и ширина в дюймах).

После этого нажмите F8 , что бы подогнать размер листа схемы под окно редактирования.

Соберем схему согласно рисунку[2].

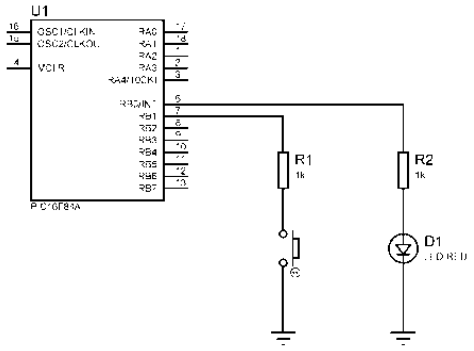

Рис. 1.2.7. Схема с управляющим светодиодом.

Для начала определимся со списком деталей.

Итак, нам нужны: микроконтроллер PIC16F84A 1 штука, светодиод красный 1 штука, кнопка и два резистора по 1 кОм каждый.

Откройте библиотеку компонентов. Чтобы вам долго не искать подскажу, наберите в окне KEYWORDS pic16f84a. Теперь либо дважды нажав ENTER, но тогда закроется библиотека и придется заново ее открывать, либо дважды щелкнув левой кнопкой по строке с описанием компонента, появившейся в окне RESULTS результат), вы переместите выбранный вами компонент в список Object Selector. Выберите, таким образом, резистор, набрав RES ,кнопку BUTTON и светодиод LED-RED .

Компоненты набираются по одному экземпляру, размножить их можно уже потом просто выбирая в списке Object Selector. Закройте библиотеку, нажав OK или же закрыв окно. Со временем к вам придет опыт, и вы будете, сами определять, какие нужны компоненты и где они находятся.

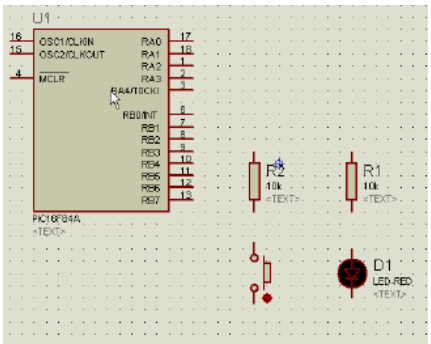

Рис. 1.2.8. Расположение элементов в Proteus ISIS

Разместим элементы на схеме, щелкнув левой кнопкой сначала по названию компонента в списке, а затем в нужном нам месте на пустой пока еще схеме. Разместите и разверните, если это необходимо все компоненты. В итоге получится, что-то вроде этого.

Нам не хватает еще одного важного элемента - «земли» или «корпуса». Элементы такого типа (терминалы) выбираются в режиме INTER SHEET TERMINAL .

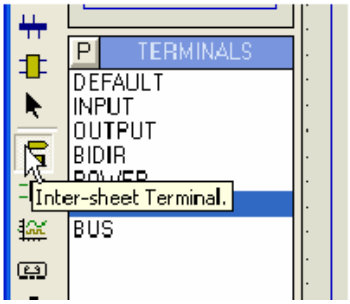

Рис. 1.2.9. Режим Inter-sheet Terminal.

Выберите элемент GROUND (земля) и поместите его на схеме под кнопкой и светодиодом. Теперь соедините компоненты меж собой как показано на схеме.

Измените сопротивление резисторов на 1 кОм. Также измените, тип модели резисторов на digital (цифровой), это необходимо, часы симулятор не тратил время на обсчет аналоговых свойств резисторов. Нам нужно только, горит или нет светодиод и нажата кнопка или нет, то есть чисто логические уровни. Схема готова.

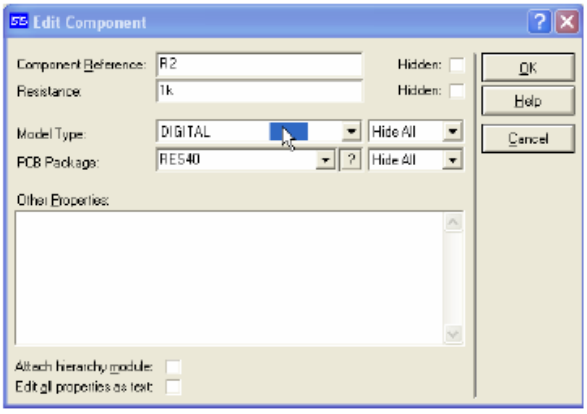

Рис. 1.2.10. Изменение параметров компонента.

Сохраните проект, в свою папку, чтобы не путаться, под именем LED.DSN. Если вы собираете схему на микроконтроллерах и хотите только отладить программу, не старайтесь точно воспроизвести схему источник, насколько это возможно меньше применяйте аналоговые устройства или же старайтесь заменить их цифровыми примитивами. Многие модели имеют два варианта аналоговый и цифровой, например тот же резистор. Транзисторы, работающие ключами можно заменить либо инверторами, либо буферами исходя из их проводимости.

Попробуем заставить схему заработать. Для этого нам необходим файл исходник. Среда PROTEUS поддерживает много средств разработки, среди них и HI-TECH Си компилятор и CROWHILL PIC BASIC и BASIC STAMP. И это только для микроконтроллеров фирмы MICROCHIP. Мы воспользуемся ассемблером MPASM.

Скопируйте ниже приведенный исходный код и сохраните файл, в папку с проектом, под именем LED.asm.

```
list p=16f84
#include <p16F84A.inc>
 CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON &_HS_OSC
call Delay500 bcf LED call Delay500 goto start
Delay500 clrf DelayL clrf DelayM movlw 3h movwf DelayH Wait1 decfsz DelayL goto Wait1
decfsz DelayM
goto Wait1 decfsz DelayH goto Wait1 return
end
```

Добавим наш исходник в проект. Для этого в меню SOURCE(исходный код) выберем ADD/REMOVE SOURCE FILE(добавить/удалить файл). В появившемся окне нажмем кнопку NEW (новый). В строке SOURCE CODE FILINAME выберем наш исходник, с помощью кнопки CHANGE (сменить), а в строке CODE GENERATION TOOLS компилятор MPASM. Подтвердим наш выбор, нажав OK .

Соберем проект, то есть создадим файл прошивки. Откроем меню SOURCE и нажмем BUILD ALL . Откроется лог компилятора, сообщая, что все в порядке, если вы не допустили ошибок, или же появятся строки с ошибками.

Закроем окно лога. «Прошьем» микроконтроллер. Для этого в окне свойств микроконтроллера, как открыть это окно рассказывалось выше, в строке PROGRAMM FILE выберем файл прошивки LED.HEX , который появился в нашей папке после компиляции.

В строке PROCESSOR CLOCK FREQUENCY (тактовая частота процессора) выставьте 4 МГЦ. В строке PROGRAMM CONFIGURATION WORD (слово конфигурации) выставлять ничего не надо, если соответствующие данные есть в файле-исходнике. Остальные установки также изменять пока нет необходимости. Сохраним еще раз проект.

Запустим проект, светодиод должен мигать.

Рассмотрим, что нам предлагает PROTEUS в качестве инструментов отладки программы и просмотра внутренностей микроконтроллера[16]. Поставьте проект на паузу. Появится окно отладчика, если этого не произошло, то в меню DEBUG отметьте пункт PIC CPU SOURCE CODE.

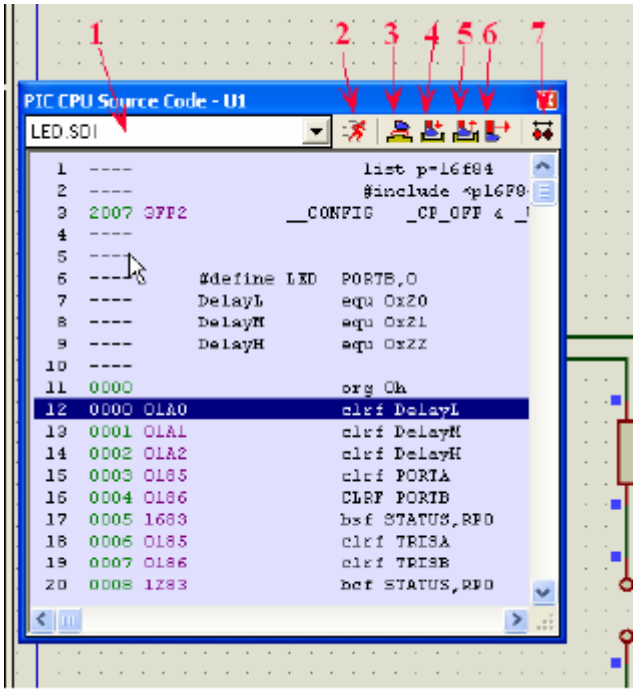

Рис. 1.2.11. Окно отладчика Proteus ISIS.

На рисунке 1.2.11 под цифрами:

1. Выбор файла отладки, если было два и более исходника, один, например, на СИ или бэйсике а другой на ассемблере.

2. Продолжить выполнение программы.

3. Шаг без входа в подпрограмму.

4. Шаг с входом в подпрограмму.

5. Исполнять код до возврата из подпрограммы. Фактически остановка происходит на следующей за RETURN командой.

6. Использовать эту опцию можно естественно, только находясь в подпрограмме.

7. Исполнять пока не будет достигнута выделенная полосой строка (курсор).

8. Триггер точек остановки (брейк пойнт), то есть вкл /выкл точку.

Щелкнув правой кнопкой по окну отладчика можно изменить его настройки.

В появившемся меню есть такие опции:

GOTO LINE перейти на линию GOTO ADDRESS перейти на адрес FIND найти

TOGGLE (SET/CLEAR) BREAK POINTустановить / очистить точку установки

ENABLE ALL BREAK POINT разрешить все точки остановки

DISABLE ALL BREAK POINT запретить все точки (не удалить !)

CLEAR ALL BREAK POINTудалить все точки

FIX-UP BREAKPOINTS ON LOAD зафиксировать (разрешить) точки при загрузке проекта

Отметьте эту опцию.

DISPLAY LINE NUMBERS показывать номера линий DISPLAY ADDRESSES показывать адреса команд

DISPLAY OPCODES показывать опкод команд

SET FONT выбор шрифта

Чтобы видеть символы кириллицы выберите шрифт COURIER NEW , ***а*** для того чтобы все убралось в окне, поставьте размер шрифта 8.

SET COLOR выбор цвета текста, фона и т.д.

Поставьте точку остановки на строку с номером 0. Как это сделать, думаю, вы уже сами поняли. У номера строки появится красная точка, индикатор того, что точка остановки активна. Если вместо точки окружность, то значит, установленная точка не активна. Выключите

проект и снова запустите. Программа прервалась именно там, где мы поставили точку. Красный треугольник (маркер) показывает на текущую строку. Поставьте свои точки. Потренируйтесь, запуская проект и используя опции пошаговой трассировки.

Но что это нам дает? Пока ничего, просто перемещаемся по коду, не видя ни спецрегистров, ни регистров пользователя.

- В том же меню DEBUG отметьте пункт PIC CPU REGISTER. Теперь при трассировке кода вы сможете наблюдать содержимое спецрегистров.

Есть два способа увидеть содержимое регистров пользователя и спецрегистров одновременно, который не очень удобный отметьте в меню DEBUG пункт PIC CPU DATA MEMORY. Не удобный, потому что отображаются они все сразу, в виде таблицы и только во время паузы или на точке остановки.

Второй более продвинутый способ это WATCH WINDOW. Отметьте соответствующий пункт в меню DEBUG . Напоминаю, что выбор всех этих окон, возможен только на, находящемся в паузе или работающем проекте. Щелкните правой кнопкой по появившемуся окну WATCH WINDOW.

Щелкните по ADD ITEMS (BY NAME) , в появившемся списке выберите PORTA дважды щелкнув левой кнопкой по нему. Элемент добавится в окно WATCH WINDOW. Добавьте также PORTB , TRISA , TRISB . Когда закончите, закройте окно, нажав DONE . Добавьте DELAYL с адресом 0x0C , DELAYM 0X0D и DELAYH 0X0E .

Для этого выберете ADD ITEMS (BY ADRESSES) , в поле NAME введите имя регистра, например, DELAYL , в поле ADDRESS естественно адрес и выберите формат USIGNED INTEGER (целое без знака). Нажмите ADD. Добавьте остальные регистры самостоятельно.

Теперь измените формат отображения данных регистров TRISA, TRISB, PORTA и PORTB на двоичный BINARY. Нам удобнее рассматривать эти регистры как отдельные биты (выводы).

Снимите все точки остановки с программы. Запустите проект.

Сейчас хорошо видно и изменение регистров в подпрограмме задержки и смена бита PORTB .

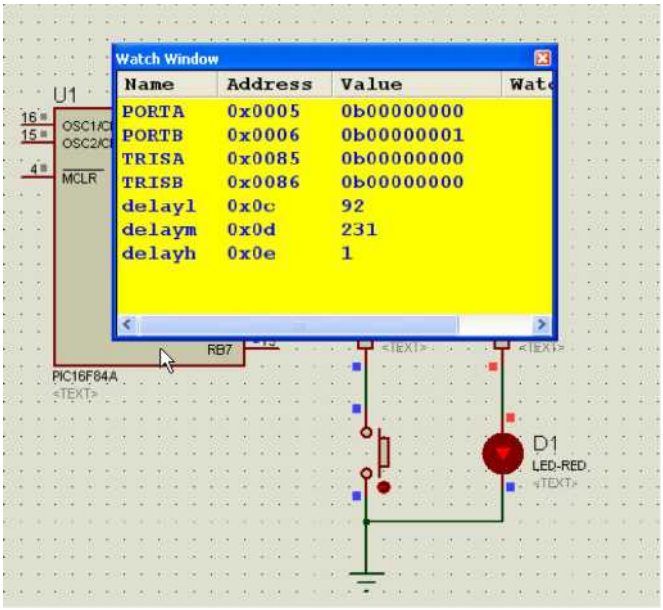

Рис. 1.2.12. Окно изменения значений регистров.

Добавим остановку по совпадению условия. Выберите WATCHPOINT CONDITION в меню WATCH WINDOW.

По порядку:

Turn off(disable) watch point - запретить остановку по условию

Suspend the simulation if any expression is true - остановить симулятор если любое условие соблюдено.

Stop the simulation only when all expression is true - остановить симулятор когда все условия соблюдены.

ITEM - выбор регистра для которого применяются условие

MASK - маска AND, OR, XOR - и значение маски

CONDITION - выбор условия (равно, больше чем, меньше чем и т.д.)

VALUE - значение с которым сравнивается регистр.

Введите в поле ITEM PORTB, в поле MASK AND 0X01 далее EQUALS(равно) 0X01. Таким образом, мы выставили условие симулятору остановиться, когда бит 0 PORTB буден равен 1. Также отметьте Suspend the simulation if any expression is true .

Запустите симулятор. Первая остановка внеплановая. Из-за того, что при старте в регистре был мусор. Продолжите симуляцию, теперь остановки будут проходить как нужно. И так, мы остановились. Посмотрите в окне отладчика, но какой команде встал симулятор.

Правильно, на CALL DELAY 500 , а предыдущая была BSF LED именно она и установила бит 0 PORTB в 1.

1.3.MPLAB

MPLAB — интегрированная среда разработки, представляющая собой набор программных продуктов, предназначенная для облегчения процесса создания, редактирования и отладки программ для микроконтроллеров семейства PIC, производимых компанией Microchip Technology. Среда разработки состоит из отдельных приложений, связанных друг с другом и включает в себя компилятор с языка ассемблер, текстовый редактор, программный симулятор и средства работы над проектами, также среда позволяет использовать компилятор с языка C. MPLAB работает под управлением операционных систем семейства Windows[17].

MPLAB состоит из следующих основных модулей:

- MPLAB Project Manager — средства работы на проектами;
- MPLAB-SIM Software Simulator — моделирование поведения программы с целью поиска и удаления ошибок в алгоритме;

- MPLAB Editor — полноценный текстовый редактор файлов ASM;
- MPASM Universal Macro Assembler — компилятор с ассемблера, компоновщик;
- MPLAB-ICE 2000 — моделирование поведения программы в реальном времени.

Создание нового проекта.

1) Запустите MPLAB и затем "мастер проекта"

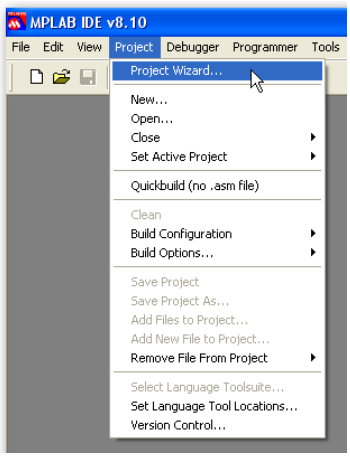

Рис. 1.3.1. Пункт меню Project

На появившейся заставке мастера нажмите "далее" и в следующем диалоге выберите модель МК который хотите использовать

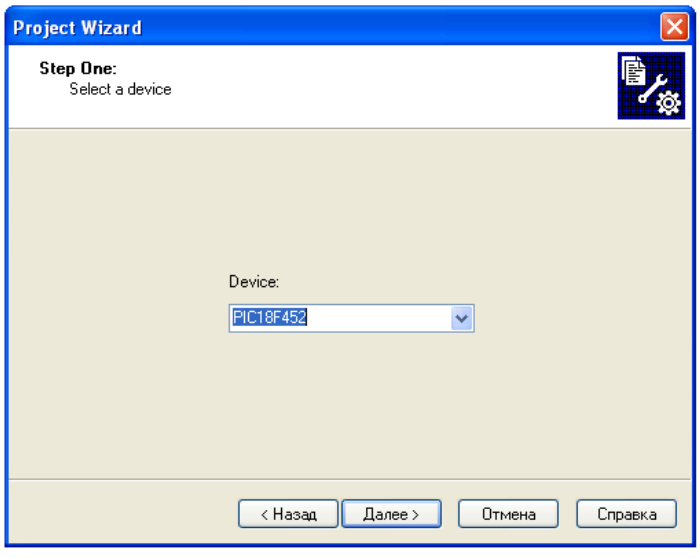

Рис. 1.3.2. Окно первого шага мастера проектов

Жмите "Далее" и во втором шаге укажите компилятор C18

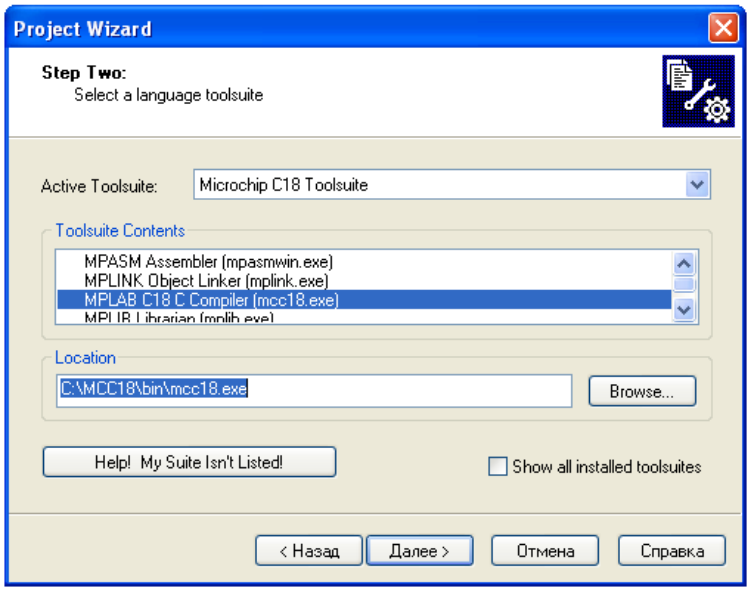

Рис.1.3.3. Окно второго шага мастера проектов

Если компилятора не окажется в списке, то найдите его вручную и укажите. Жмите "Далее" и в следующем шаге-окне нажмите "Browse" и создайте папку для проекта:

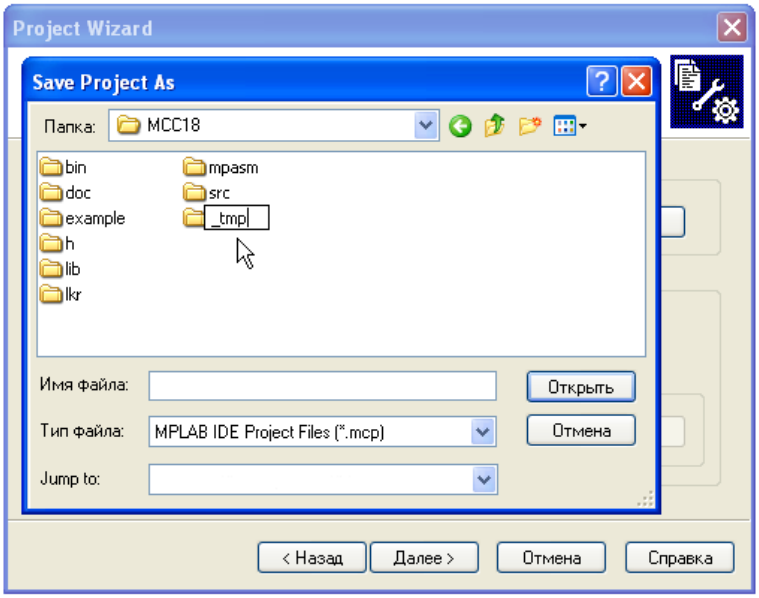

Рис. 1.3.4. Сохранение файла проекта

Откройте созданную папку и дайте название проекту и "Сохранить"

Должно получиться вот так

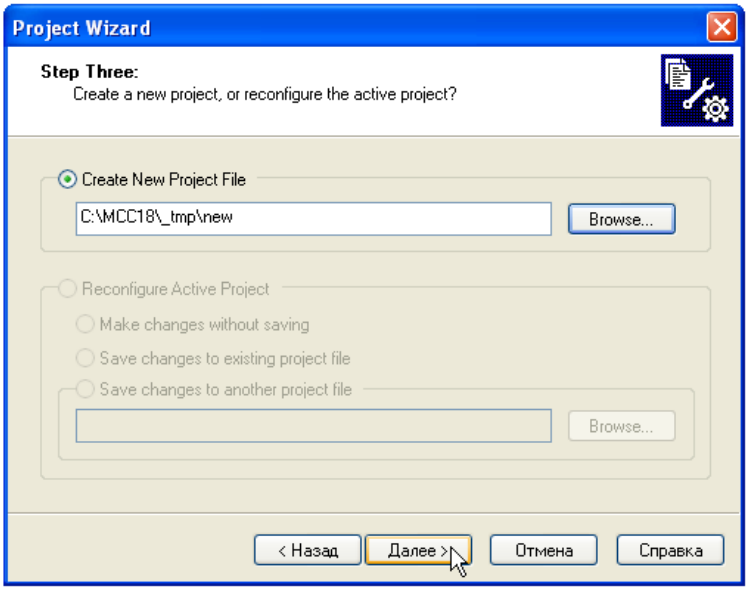

Рис. 1.3.5. Окно третьего шага мастера проектов

Жмите "Далее".

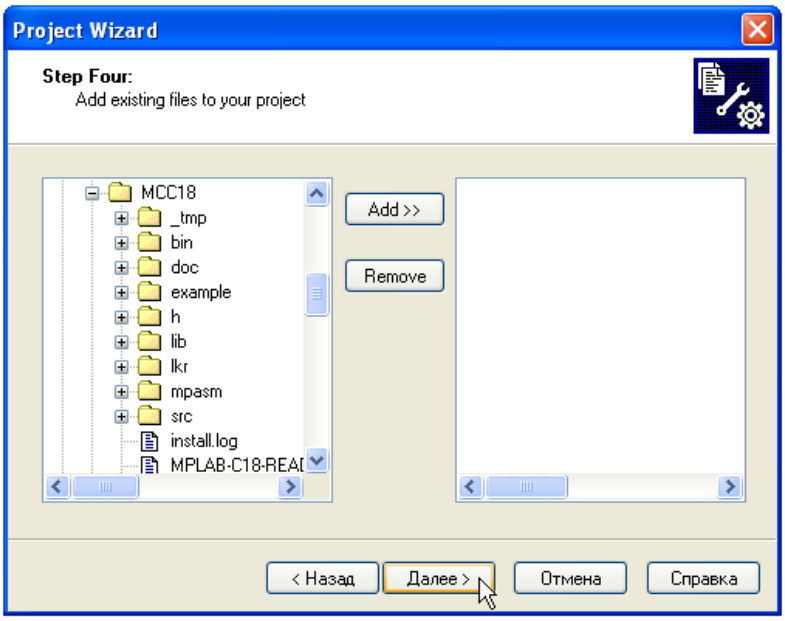

Рис. 1.3.6. Окно четвертого шага мастера проектов

Работа мастера завершена – итоговый отчет

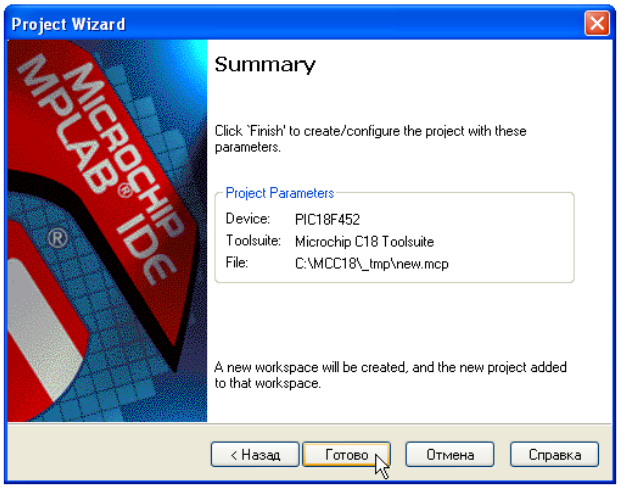

Рис. 1.3.7. Окно завершения мастера проектов

Нажмите "Готово".

Вот что теперь в папке проекта

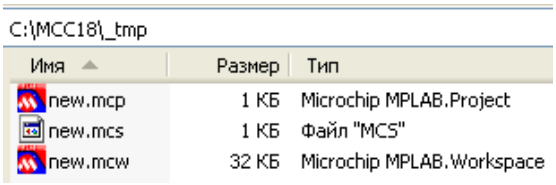

Рис. 1.3.8. Список файлов созданного проекта

Как создать файл с исходным текстом программы на Си.

В MPLAB создайте новый файл - меню "File" > "New" и сохраните его "File" > "Saveas" под именем **main.c** [14]

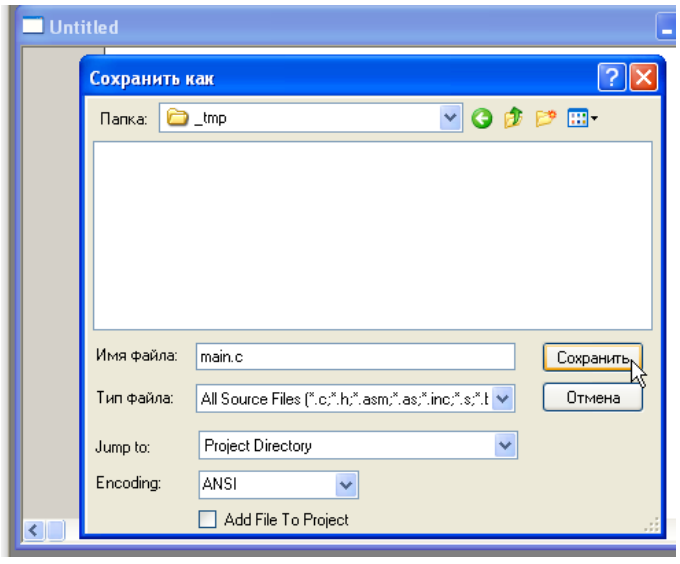

Рис. 1.3.9. Окно сохранения файла программы в C++.

Теперь нужно набрать в этом окне текст первой программы

```
#include <stdio.h> // библиотека ввод-вывод
#pragma config WDT = OFF // выкл. сторожевой таймер
void main (void)
{ // напечатать:  Hello!
printf ("Hello");
while (1); // бесконечный цикл
}
```

или скопировать и вставить в окно. Должно получиться вот так:

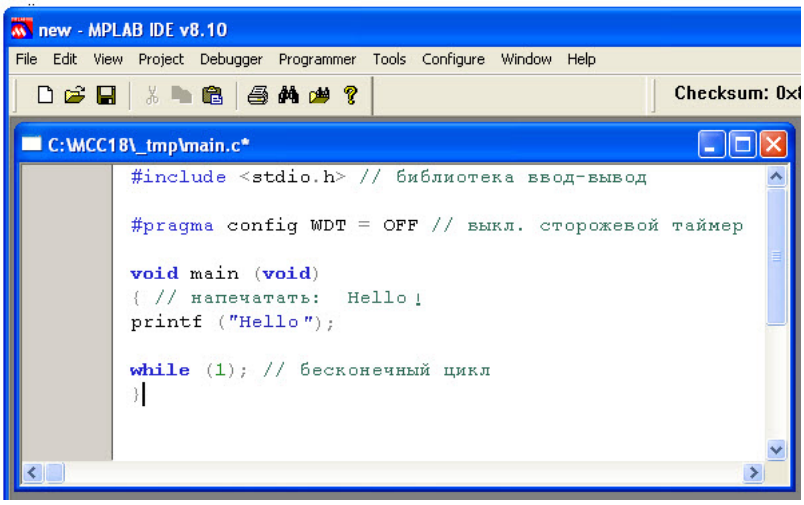

Рис.1.3.10. Окно рабочего поля ввода текста кода.

Кликните "дискетку" чтобы сохранить текст в файле.

Теперь нужно добавить **main.c** в проект.

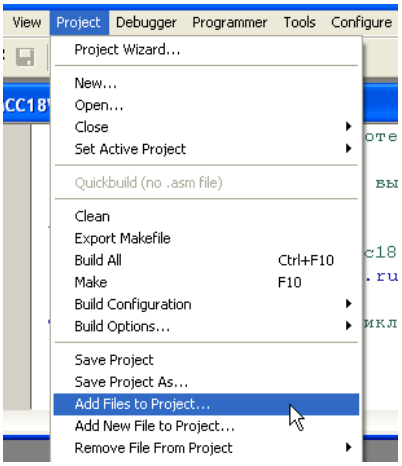

Рис. 1.3.11. Пункт меню Project-Add files to Project.

1.3.1. Компиляция и симуляция-отладка

Нужно выбрать инструмент в котором будет происходить проверка работы программы - это может быть реальное устройство с выбранным МК подключенное через интерфейс типа ICD2 к ПК, может быть симулятор PROTEUS - он позволяет симулировать работу не только МК PIC, но целого электронного устройства с несколькими МК и даже разных производителей в одной схеме[14].

Мы будем использовать симулятор MPLAB - выберите его

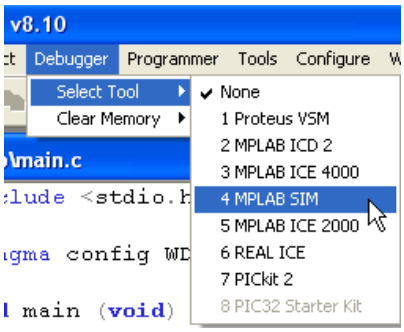

Рис. 1.3.1.1. Выбор MPLAB SIM.

Симулятор нужно настроить - меню: "Debugger" > "Settings"

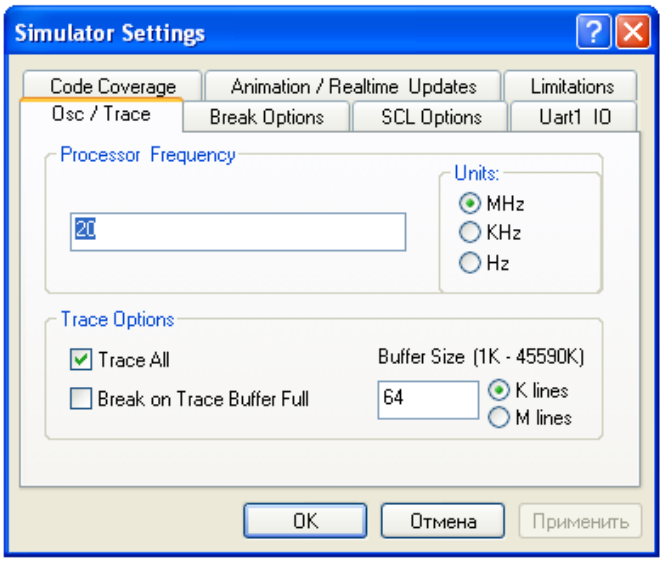

Рис. 1.3.1.2. Настройка отладчика MPLAB.

Здесь указывается частота тактирования микроконтроллера - 20 МГц в нашем примере.

Перейдите на закладку "Uart1 IO" и сделайте такие установки

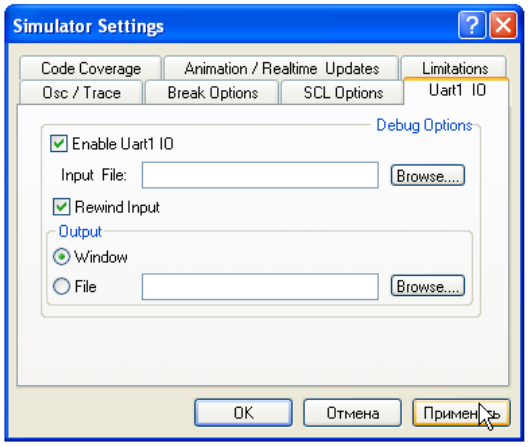

Рис. 1.3.1.3. Настройка симуляции – Uart IO.

Выход "Output" мы направим в "Window" - т.е. на экран, а можно было сохранять в файл. Если указать "Input File" то при симуляции его содержимое будет отправляться в UART МК. Нажмите "Применить" и "ОК". Выполните компиляцию проекта

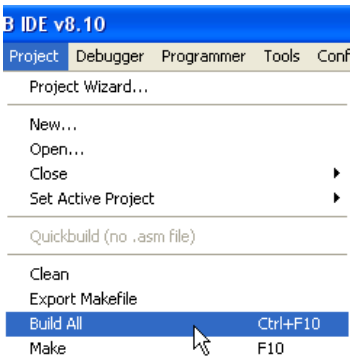

Рис. 1.3.1.4. Компиляция проекта.

В результате должно появится 2 новых окна:

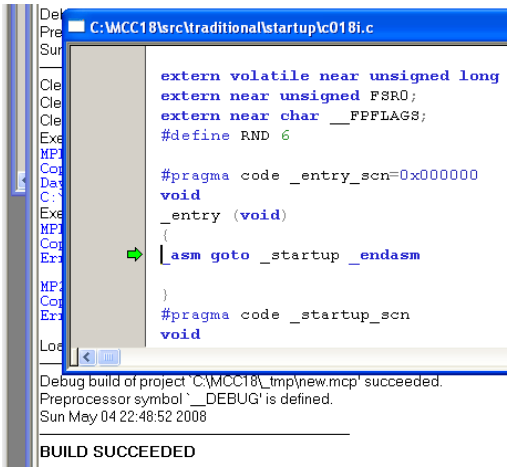

Рис. 1.3.1.5. Окно состояния компиляции.

Закройте окно **c018i.c**

Самое важное сейчас - это сообщение об успешной компиляции "**BUILD SUCCEEDED**".

В окно "**Output**" выводятся все сообщения о ходе работа инструментов пакета MPLAB - вы видите сейчас в нём подробности процесса компиляции. Если бы при компиляции возникли ошибки или предупреждения, то они тоже выводились бы в это окно.

Уменьшите окно вывода и разместите его удобно на экране, затем переключитесь на закладку "SIM Uart1" - сюда будет выводится информация из UART **PIC18F452** при симуляции.

А в папке проекта появились новые файлы:

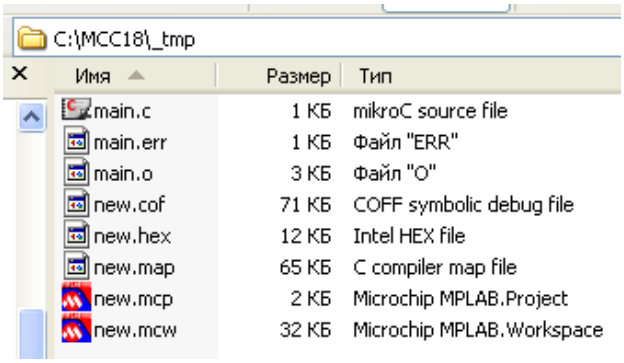

Рис. 1.3.1.6. Список файлов проекта.

main.hex - это файл "прошивка" для загрузки в реальный МК

main.cof - это файл с информацией для отладки в симуляторах по исходному тексту программы на Си.

Виртуальный микроконтроллер **PIC18F452** с нашей первой программой, откомпилированной и зашитой в него, готов начать работать.

Щелкните два раза мышкой на сером фоне, чуть левее строки

while (1); // бесконечный цикл

Появится красный кружок - это "ТОЧКА ОСТАНОВА" (BreakPoint) - перед выполнением строки кода перед которой стоит "ТО" симулятор остановится и будет ждать вашей команды. Включите секундомер - меню "Debugger" > "StopWatch" - он позволяет точно измерять временные промежутки при симуляции. Симуляцию запустите кнопкой "Пуск" - голубой треугольник в панели управления

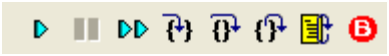

Рис. 1.3.1.7. Кнопки управления симуляцией в MPLAB SIM

Программа начнет выполняться и остановится на точке останова. Приветствие появилось в окне вывода

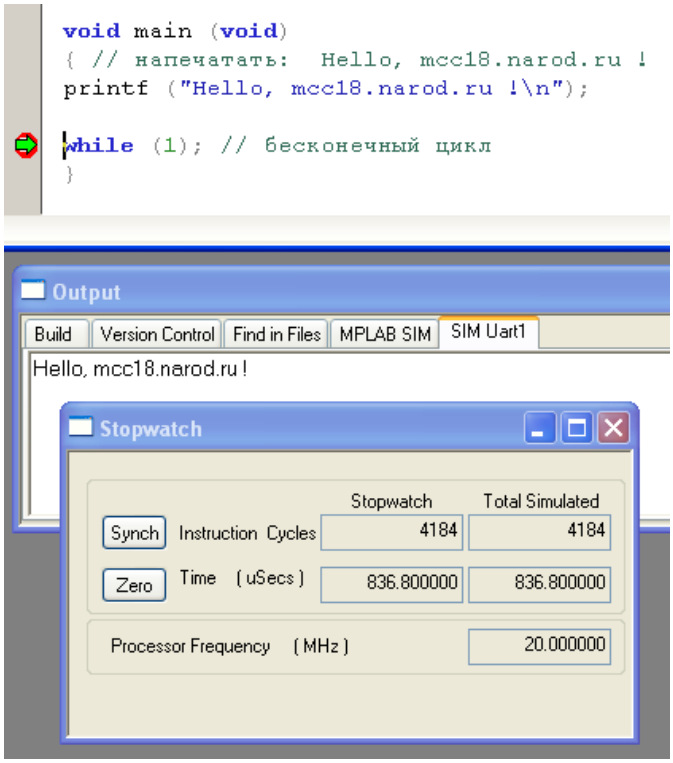

Рис. 1.3.1.8. Выполнение симуляции.

Секундомер показывает, что с запуска МК прошло 836,8 мкс это 4184 машинных цикла для микроконтроллеров PIC18 - у них один машинный цикл это 4 периода тактового сигнала.

Нажмите еще раз кнопку "Пуск" - цикл **while** (1); будет выполнен еще раз и в окне секундомера появится число 4186 - значит на

выполнение цикла тратится 2 машинных цикла PIC18F452 или 8 периодов частоты 20 МГц.

1.4. Программатор-Отладчик PICkit™ 2

1.4.1. Обзор Программатора-отладчика PICkit 2

Эта подчасть описывает свойства программатора - отладчика PICkit 2 и меню программного обеспечения PICkit 2 Programmer[15].

Комплект PICkit 2 содержит следующее:

1. Программатор/отладчик PICkit 2.
2. USB кабель
3. Диск с программным обеспечением PICkit Starter Kit и MPLAB IDE

Комплекты PICkit Starter Kit и PICkit 2 Debug Express дополнительно содержат демонстрационные платы с установленным PIC микроконтроллером.

1.4.2. Программатор-отладчик разработчика PICkit 2

Программатор-отладчик разработчика PICkit 2 это недорогое средство разработки, поддерживающее программирование большинства микроконтроллеров, микросхем памяти и KeeLOQ производства компании Microchip Technology Inc. Для получения полного списка поддерживаемых микросхем обратитесь к файлу README на диске PICkit 2 Starter Kit.

Поддержка новых микросхем может быть добавлена при выходе обновлений программного обеспечения PICkit 2. PICKit 2 так же может использоваться для внутрисхемной отладки некоторых микроконтроллеров.

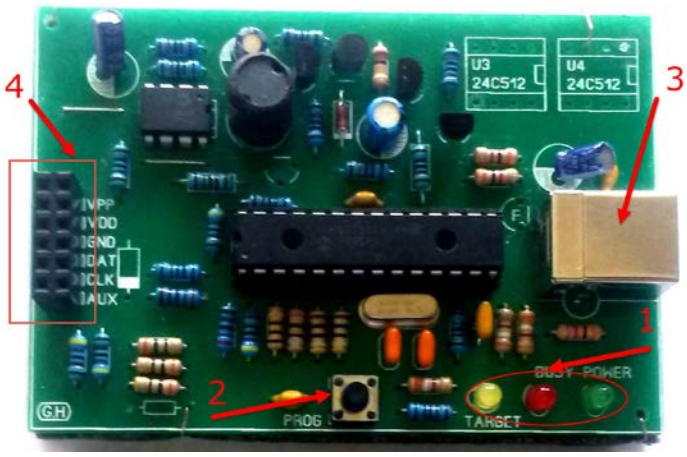

Рис. 1.4.2.1. Программатор PICkit 2.

1. Светодиоды состояния
2. Кнопка
3. Разъем для подключения USB кабеля
4. Разъем для подключения программируемого устройства

1.4.3. Подключение к USB порту

PICkit 2 имеет USB разъем типа mini-B. Подключите PICkit 2 к компьютеру используя кабель из комплекта поставки.

1.4.4. Светодиоды состояния

Светодиоды состояния отображают статус программатора/отладчика PICkit 2[15].

1. Power (зеленый светодиод) показывает, что PICkit 2 подключен к USB порту.
2. Target (желтый светодиод) показывает, что PICkit 2 выдает питание на целевое устройство
3. Busy (красный светодиод) показывает, что PICkit 2 занят и выполняет такие функции как программирование, проверку и т.п.

1.4.5. Кнопка

Кнопка может быть задействована для запуска программирования целевого устройства, для этого установите галочку на пункте *Programmer>Write on PICkit Button*.

Кнопка также может использоваться для ввода PICkit 2 в загрузочный режим, в этом режиме можно обновить программное обеспечение программатора PICkit 2.

1.4.6. Разъем для подключения программируемого устройства

Программирующий разъем имеет 6 выводов для подключения целевого устройства. Назначение выводов указано на рис. 1.4.6.1.

Для получения подробной информации о том, как использовать PICkit 2 для внутрисхемного программирования обратитесь к главе 3 «Использование внутрисхемного программирования (ICSP)» данного руководства.

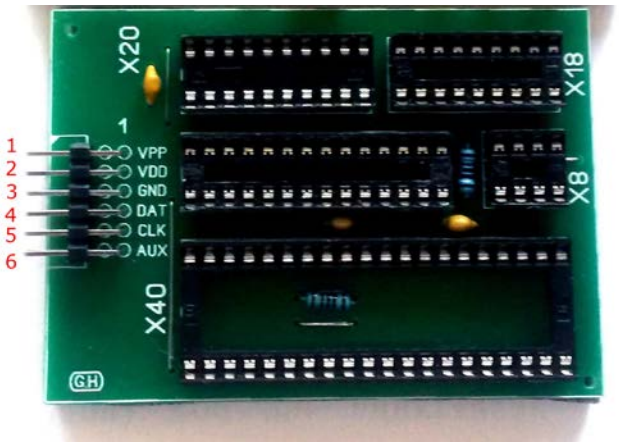

Рис. 1.4.6.1. Разъем программирования.

Назначение выводов:

1. VPP / MCLR
2. VDD напряжение питания целевого устройства
3. VSS земля
4. ICSPDAT / PGD

5. ICSPCLK / PGC

6. AUX

1.4.7. Программное обеспечение PICkit 2

Программное обеспечение PICkit 2 Programmer позволяет программировать все поддерживаемые программатором PICkit 2 микросхемы[4]. Интерфейс программы приведен на рис. 1.4.7.1.

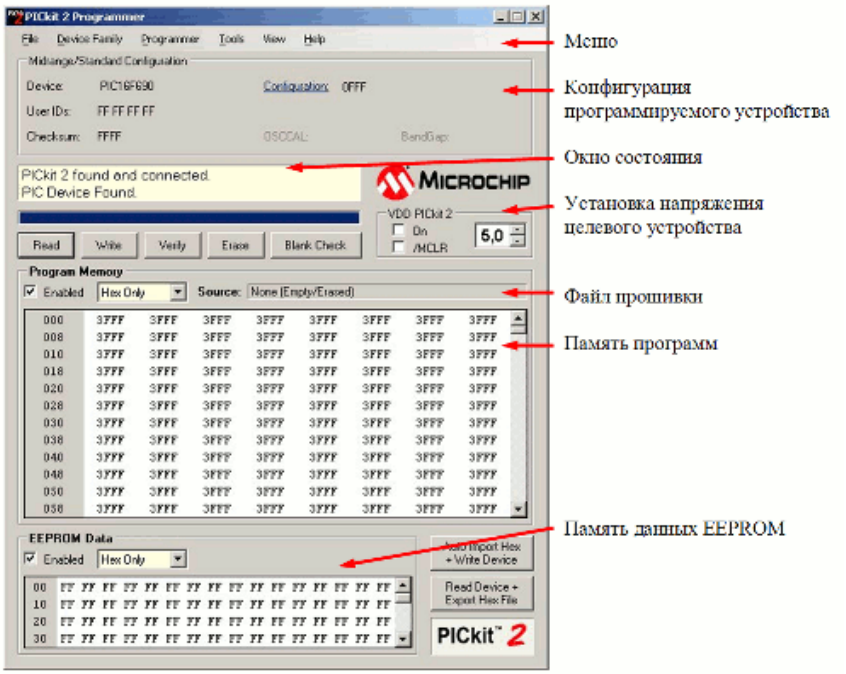

Рис. 1.4.7.1. Интерфейс программы PICkit 2 Programmer.

Подключение PICKit 2:

- Установка программного обеспечения
- Подключение к программируемой микросхеме
- Управление питанием
- Импорт файла .hex
- Программирование
- Верификация
- Чтение содержимого памяти микроконтроллера
- Защита кода

- Стирание памяти и проверка на чистоту
- Автоматическое программирование/считывание

Подключение PICKit 2

- Подключите ваш PICKit 2 к персональному компьютеру с помощью кабеля USB, входящего в комплект поставки
- Подключите PICKit 2 к целевой плате с помощью 6-контактного разъема
- Не подключайте программатор к целевой плате, имеющей внешнее питание, пока он не включен в работающий USB-порт
- Для подключения PICKit 2 к отладочным платам, имеющим разъем RJ-11 (как у ICD 2) используйте переходник AC164110
- При включении PICKit 2 в USB рекомендуется отключать его от целевой платы. Аналогичная рекомендация и при перезагрузке ПК.

Внешний вид оболочки приведен на рис.1.4.7.2.

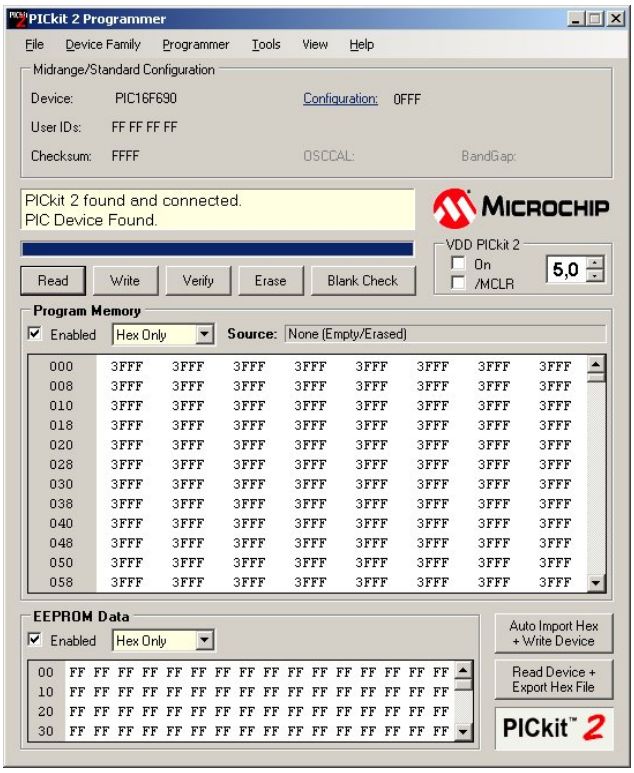

Рис. 1.4.7.2. Оболочка программы PICkit 2 Programmer.

1.4.8. Подключение к программируемой микросхеме

PICKit 2 поддерживает программирование множества микроконтроллеров Microchip PIC и микросхем памяти EEPROM[9].

При запуске программы производится автоматическое определение типа подключенного контроллера и его отображение в окне Configuration (рис.1.4.8.1).

Рис. 1.4.8.1. Определение подключенного контроллера.

Если устройство не определилось – проверьте подачу питающего напряжения и надежность подключения к целевой плате.

Можно в любой момент выбрать нужное вам семейство, воспользовавшись меню Device Family, при этом PICKit 2 попытается соединиться с целевым устройством (рис.1.4.8.2).

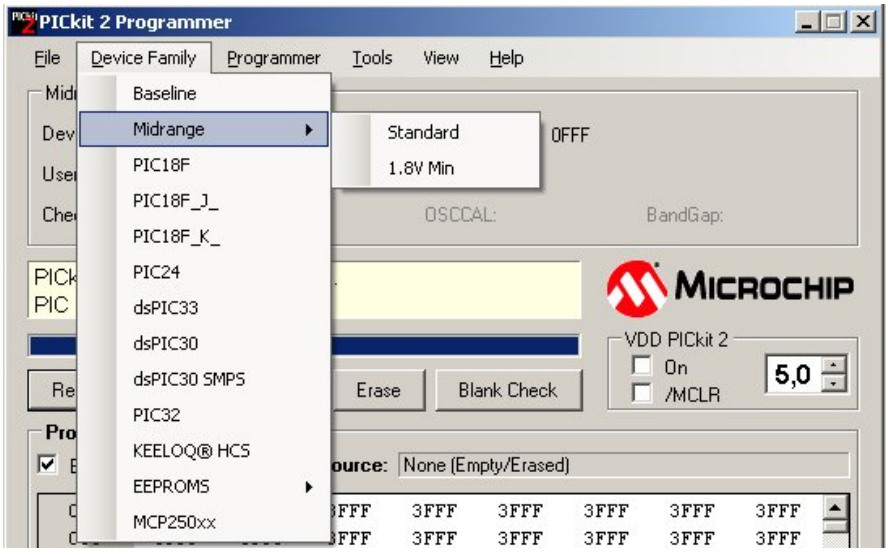

Рис. 1.4.8.2. Выбор программируемого семейства

При выборе семейства Baseline, а также микросхем KEELOQ® и EEPROM, необходимо также выбрать конкретное изделие из выпадающего списка (рис.1.4.8.3), т.к. в этих микросхемах нет идентификационных битов (device ID).

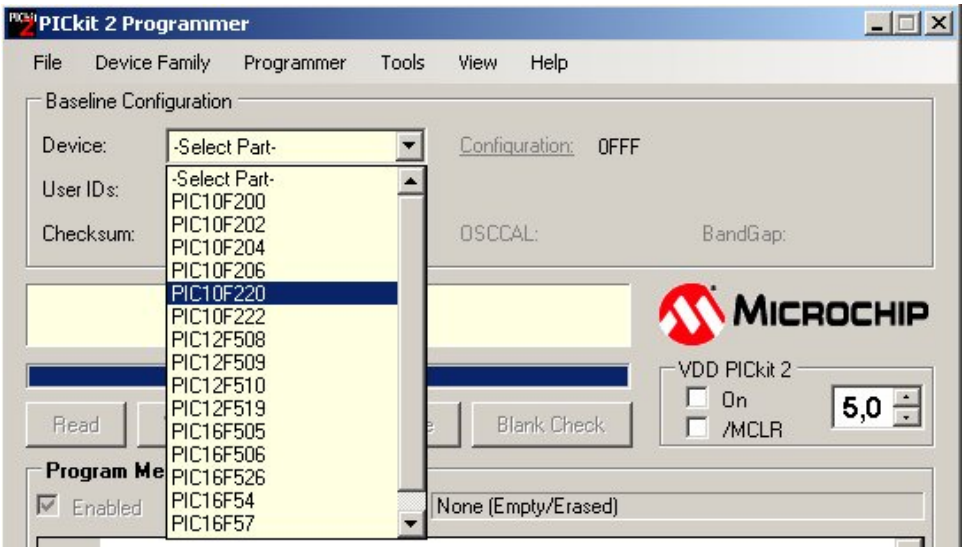

Рис. 1.4.8.3. Выбор контроллеров базового семейства.

1.4.9. Управление питанием

При работе с программатором PICKit 2 возможны два варианта питания целевой микросхемы: от PICKit 2 и внешнее питание.

Питание от PICKit 2

Если используется питание от PICKit 2, отдельно подавать питание на плату не нужно, т.к. программатор измерит его и не даст подать питание через себя. Если плата не запитана, то оболочка дает возможность установить значение питающего напряжения, подаваемого с PICKit 2 (рис. 1.4.9.1)[15].

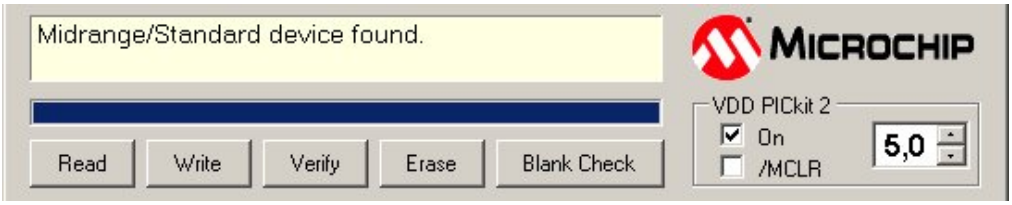

Рис. 1.4.9.1. Включение питания от PICkit 2.

Для подачи напряжения выберите значение On.

В случае короткого замыкания или превышении максимального тока запитки выдается сообщение об ошибке (рис. 1.4.9.2). Потребление целевой платы не должно превышать 25 мА, при этом время нарастания питающего напряжения при включении составляет не более 500 мкс.

Рис. 1.4.9.2. Ошибка VDD

Внешнее питание

Целевая плата может питаться от собственного источника питания. PICKit 2 автоматически детектирует наличие внешнего питания и, в случае его наличия, в оболочке меняется заголовок окна с VDD PICkit 2 на VDD Target, отключается возможность подачи питания и отображается значение внешнего питающего напряжения (рис. 1.4.9.3). Щелчок по галочке Check обновляет отображенное питающее напряжение. В случае пропадания внешнего питания оболочка переключиться в режим подачи питания от PICKit 2.

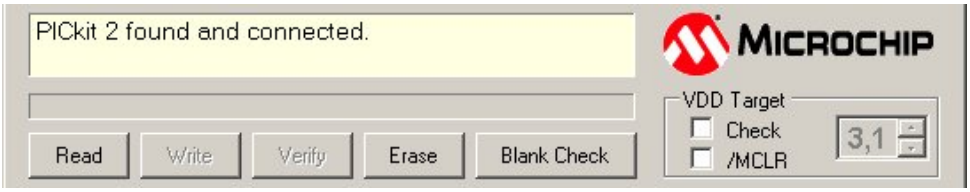

Рис. 1.4.9.3. Внешнее питание.

1.4.10. Импорт .hex файлов

Для импорта файла прошивки в формате .hex выберите пункт меню *File→Import HEX*. В случае, если в файле прошивки отсутствуют какие-либо конфигурационные биты, оболочка выдаст предупреждение. Для правильного сохранения текущей прошивки в файл .hex выберите *File→Export* в меню оболочки MPLAB IDE[6].

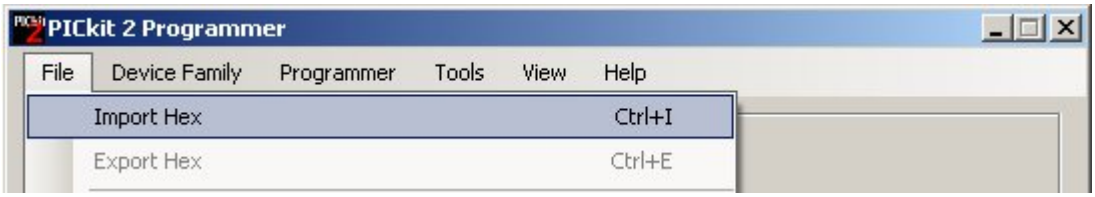

Рис. 1.4.10.1. Импорт hex файла.

1.4.11. Программирование микросхем

После правильного выбора семейства микросхем и импорта файла прошивки возможно программирование целевой микросхемы по кнопке Write (рис. 1.4.11.1).

Рис. 1.4.11.1. Кнопка Запись.

Микросхема будет стерта и запрограммирована загруженной прошивкой.

Большая часть микроконтроллеров поддерживает режим общего стирания (Bulk Erase), доступный при минимальном напряжении питания, часть контроллеров также поддерживают блочное стирание (Row Erase). Процедура блочного стирания занимает больше времени, нежели общее стирание, но доступно при пониженных напряжениях питания. PICKit 2 автоматически переключается на блочное стирание при невозможности выполнения общего стирания. Если микроконтроллер не поддерживает блочное стирание – выдается предупреждение. Список контроллеров, поддерживающих блочное стирание, доступен в файле readme.

Ход выполнения процедуры программирования отображается в строке статуса. В случае, если программирование прошло успешно, строка становиться зеленого цвета и на ней пишется Programming Successful (рис. 1.4.11.2).

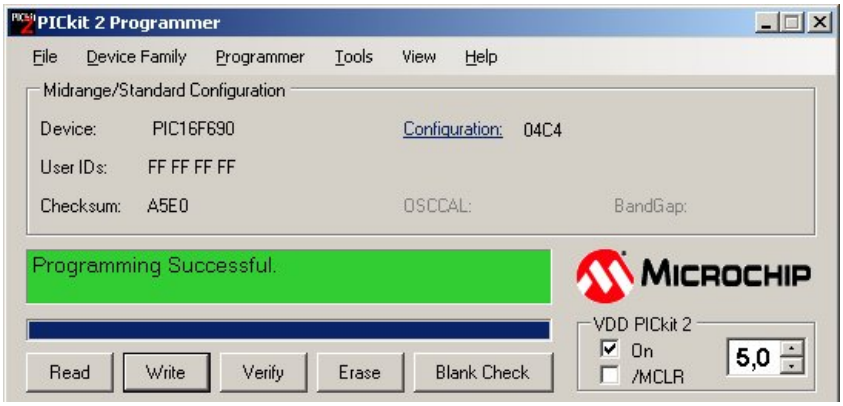

Рис. 1.4.11.2. Успешное завершение записи.

В случае ошибки строка становится красной и на ней пишется Programming Failed (рис. 1.4.11.3). В этом случае попробуйте повторить процедуру программирования.

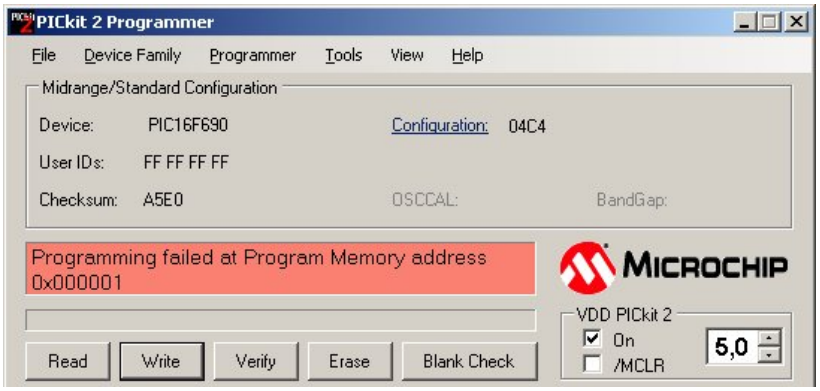

Рис. 1.4.11.3. Ошибка программирования.

В других случаях строка статуса становится желтой и на ней пишется причина предупреждения, например, нет соединения с целевым устройством (рис. 1.4.11.4).

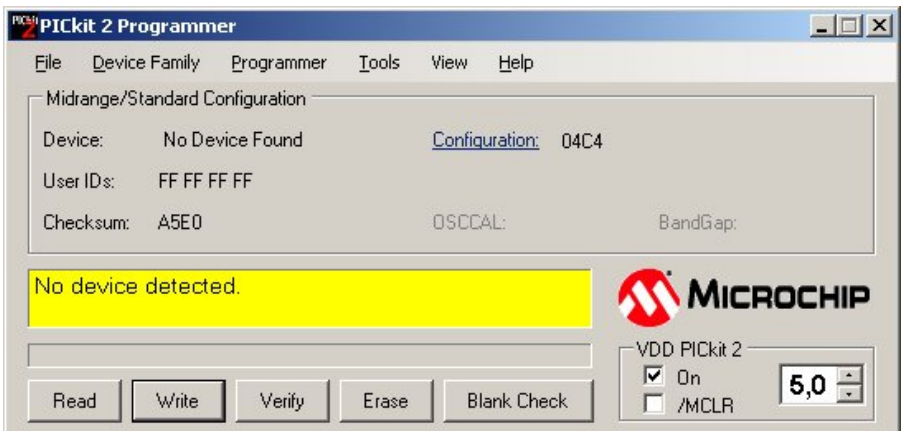

Рис. 1.4.11.4. Предупреждение при записи.

2. ОСНОВНАЯ ЧАСТЬ

2.1. Лабораторный стенд

Лабораторный стенд предназначен для работы с микроконтроллерами PIC серии 16F873A/876A. Внешний вид стенда приведен на рис. 2.1.1.

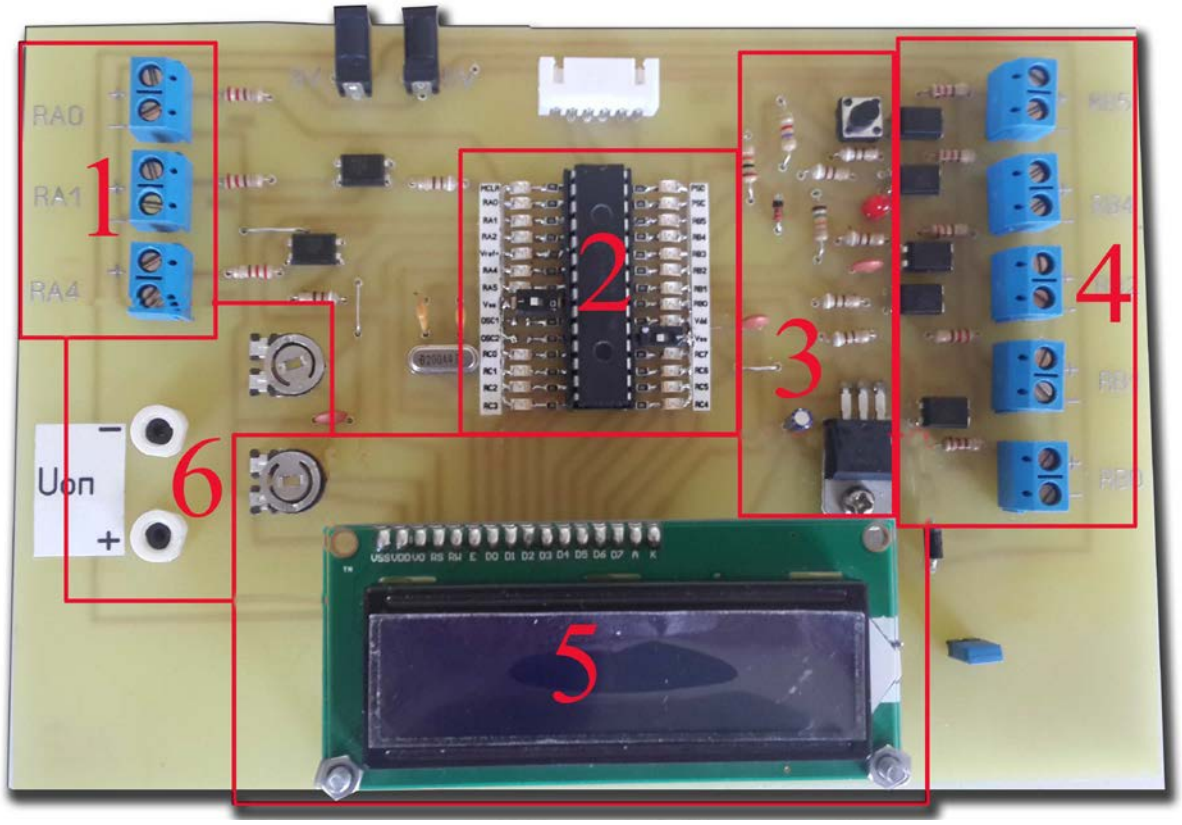

Рис.2.1.1. Лабораторный стенд на базе МК PIC 16F876A.

Цифрами обозначены основные блоки лабораторного стенда:

1. Порты ввода RA0, RA1 и RA4 предназначены для подачи сигналов на микроконтроллер.

2. Микроконтроллер PIC 16F876A со схемой индикации. Светодиоды сигнализируют наличие сигнала на соответствующем порте.

3. Цепь питания, состоящая из стабилизатора LM7805, который преобразует входное напряжение 9В в 5В; светодиода, сигнализирующего наличие тока в цепи и кнопки, для перезапуска микроконтроллера.

4. Порты вывода RB0, RB1, RB2, RB4 и RB5 предназначенные для выдачи сигналов от микроконтроллера. Все порты имеют гальваническую развязку, обеспечиваемую оптронами EL817, т.е. цепи

питания микроконтроллера и портов вывода независимы, что обеспечивает полную защищенность микроконтроллера от внешних воздействий.

5. Дисплей LCD 1602, представляет собой ЖК-дисплей с возможностью вывода текста в 2 строки по 16 символов каждая. Имеется возможность изменять контрастность текста при помощи подстроечного резистора.

6. Блок регулировки опорного напряжения, подаваемого на микроконтроллер, в основном применяется для изменения размера шага квантования. Также имеются два вывода для замера этого напряжения при помощи вольтметра.

Помимо этого, имеется интерфейс для программирования микроконтроллера через программатор PicKit.

Для лабораторного стенда необходимо два источника питания: +9В для основной схемы; +5В для обеспечения гальванической развязки.

Спецификация элементов для лабораторного стенда приведена ниже в таблице 2.1.1.

Таблица 2.1.1. Спецификация элементов для лабораторного стенда.

№	Наименование/тип элемента	Количество
1	Микроконтроллер PIC 16F876A	1 шт.
2	Дисплей LCD 1602	1 шт.
3	Кварц 20 МГц	1 шт.
4	Панелька для PIC 16F876A	1 шт.
5	Кнопка	1 шт.
6	Светодиоды smd (зеленые)	24 шт.
7	Светодиод (красный)	1 шт.
8	Конденсатор 30пФ	2 шт.
9	Конденсатор 0,1мкФ	2 шт.

10	Конденсатор 10мкФ х 16В	1 шт.
11	Резистор 200 Ом - 0,25Вт	8 шт.
12	Резистор 1 кОм - 0,25Вт	5 шт.
13	Резистор 15 кОм - 0,25 Вт	1 шт.
14	Резистор 4.7 кОм – 0,25Вт	1 шт.
15	Диод 1N4007	1 шт.
16	Диод 1N4148	1 шт.
17	Оптопара EL817	5 шт.
18	Подстроечный резистор 4.7 кОм	2 шт.
19	Разъемы DC006A	2 шт.
20	Разъемы KF301 -2P	8 шт.

Схема электрическая принципиальная изображена на рис. 2.1.2.

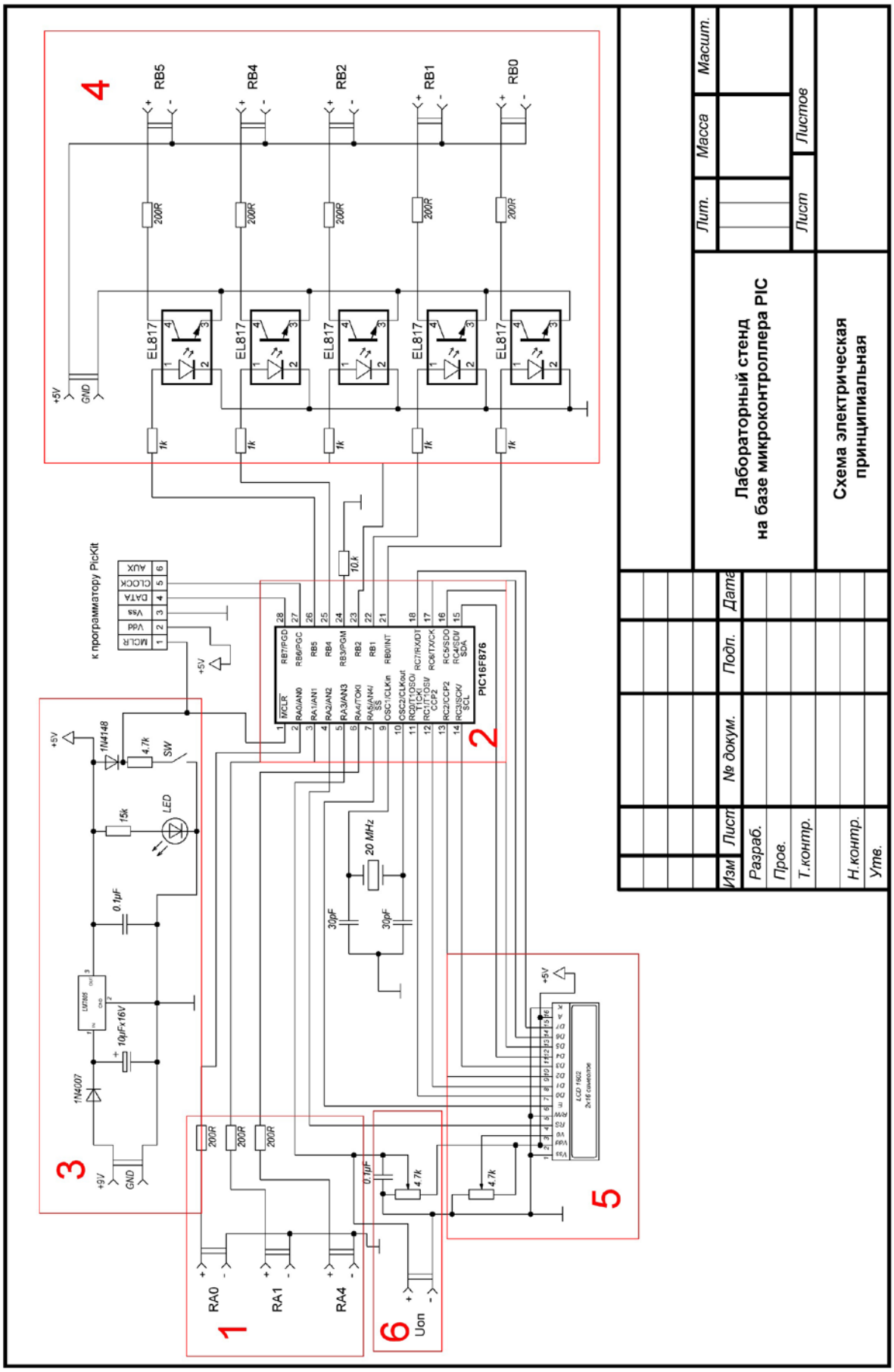

Рис.2.1.2.Схема электрическая принципиальная лабораторного стенда

2.2. Лабораторные работы

Для освоения материала по микроконтроллерам PIC ниже приведены 5 лабораторных работ. Они выполняются в программе PROTEUS VSM, но также рассчитаны для выполнения на физическом лабораторном стенде. Начиная с реализации простого мерцающего светодиода и постепенно усложняясь, в итоге, вы сможете собрать устройство для измерения температуры[1][2].

2.2.1. Лабораторная работа №1. Генератор импульсов

Цель работы: изучить устройство микроконтроллера PIC 16F876A и понять основы программирования микроконтроллеров на примере мерцающего светодиода.

Для начала работы необходимо запустить программу PROTEUS ISIS, открыть файл *generator.dsn*. На рабочем поле программы появится схема лабораторной работы, состоящая из микроконтроллера PIC 16F876A, цепи питания и подключенного к порту RB0 светодиода. Для запуска эмуляции работы схемы нажать на кнопку PLAY. В результате можно наблюдать мерцание светодиода с определенной частотой, которая задается в программе микроконтроллера. С помощью виртуального осциллографа можно наблюдать импульсы и произвести подсчет частоты их появления.

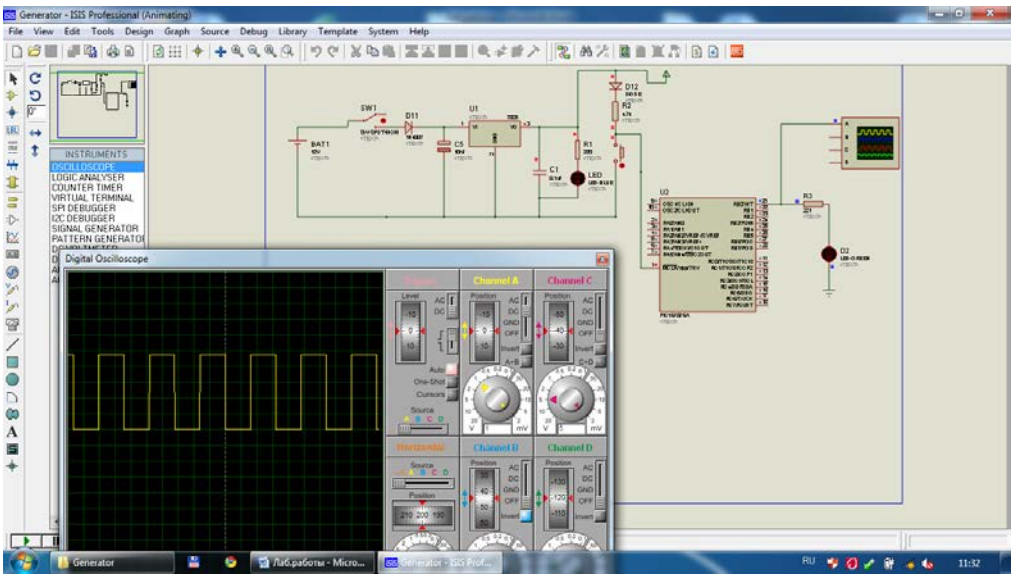

Рис. 2.2.1. Лабораторная работа №1

Код программы:

```
//====секция include====
#include<pic.h>
//====секция конфигурации====
__CONFIG (0x3F32);
//====определение портов ввода/вывода=====
#define         CHANNEL1 0b11111110  // RB0
#define         led0            RB0
//====прототип функции====
void delay(unsigned short i);
void main(void)
{
  TRISB=0b11111110;                 //конфигурирование PORTB как вывод
          while(1)                  //бесконечный цикл
          {
          led0=1;
          delay(10000);          // время включенного состояния светодиода
            led0=0;
          delay(10000);   // время выключенного состояния светодиода
  }
}
void delay(unsigned short i)
{
  for(;i>0;i--);
}
```

2.2.2. Лабораторная работа №2
8 светодиодный индикатор

Цель работы: изучить принцип преобразования входного аналогового сигнала в цифровой и вывести его на цепь светодиодов.

Для начала работы необходимо запустить программу PROTEUS ISIS, открыть файл *Indikator_diod_8.dsn.* На рабочем поле программы появится схема, состоящая из микроконтроллера PIC 16F876A, цепи питания, источника напряжения на 5В с подстроечным резистором на порте RA0 и подключенных к портам RB0-RB7 8 светодиодов. Для запуска эмуляции работы схемы нажать на кнопку PLAY. В зависимости от положения движка подстроечного резистора, можно изменять количество включенных светодиодов, т.е. положение движка изменяет напряжение,

прилагаемое на порт RA0, и число включенных светодиодов зависит от величины этого напряжения.

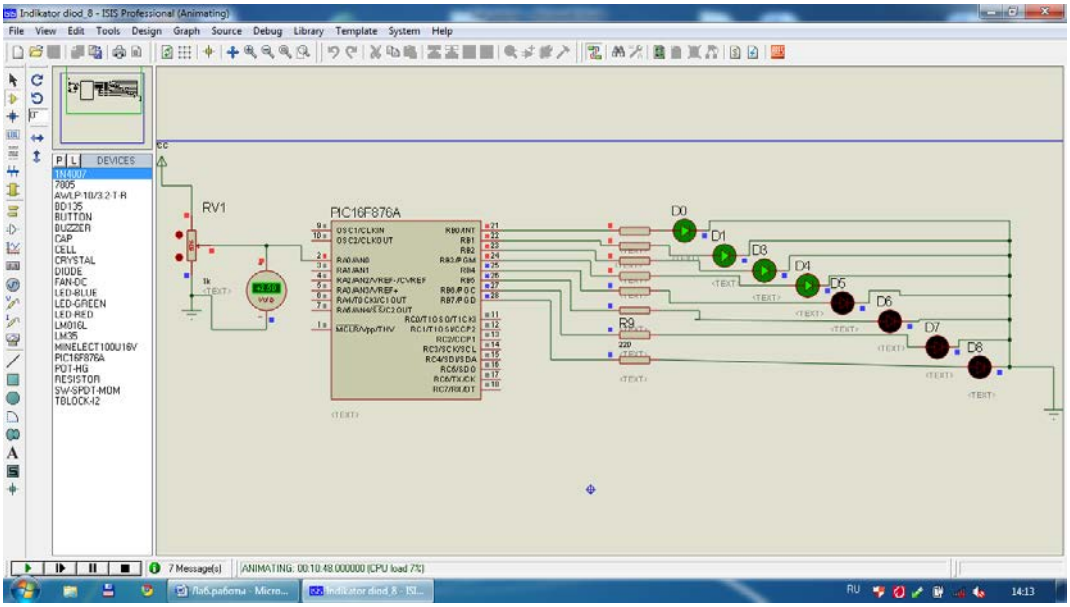

Рис. 2.2.2. Лабораторная работа №2

Код программы:

```
//====секция include====
#include<pic.h>
//====секция конфигурации====
__CONFIG (0x3F32);
//====определение портов ввода/вывода=====
#define         CHANNEL0 0b10000001  // AN0
#define         CHANNEL1 0b00000000  // PORTB
//====прототип функции====
void delay(unsigned short i);
void read_adc(void);
unsigned short read_temp(void);
//====основной блок (main)====
unsigned short result;
unsigned short temp;
void main(void)
{
  ADRESH=0;          //очистка A/D результата
  ADRESL=0;          //очистка A/D результата
  ADCON0=0b11000001;
  //установка регистра ADCON1
  ADCON1=0b10000010;  // A/D правосторонняя ориентация результата
  TRISA=0b11111111;        //конфигурация PORTA направления ввода/вывода
  TRISB=0b00000000;              //конфигурирование PORTB как вывод
  PORTA=0;
  PORTB=0;
  while(1)
```

```
  {
  read_adc();
 delay(500);
  temp=read_temp();
  if(temp>122)
  PORTB=0b00000001;
  if(temp<122)
  PORTB=0b00000000;
  if(temp>250)
  PORTB=0b00000011;
  if(temp>370)
  PORTB=0b00000111;
  if(temp>500)
  PORTB=0b00001111;
  if(temp>625)
  PORTB=0b00011111;
  if(temp>750)
  PORTB=0b00111111;
  if(temp>870)
  PORTB=0b01111111;
  if(temp>990)
  PORTB=0b11111111;
  delay(10);
  }
 }
 void read_adc(void)
 {
  unsigned short i;
  unsigned long result_temp=0;
  for(i=2000;i>0;i-=1)              //цикл 2000 раз для получения среднего
значения
  {
  ADGO = 1;                  //ADGO это бит 2 регистра ADCON0
 while(ADGO==1);   //запуск ADC, ADGO=0 после завершения прогресса ADC
  result=ADRESH;
        result=result<<8;           //сдвиг влево на 8 бит
        result=result|ADRESL;     //10 битный результат с ADC
        result_temp+=result;
  }
  result = result_temp/2000;         //получение среднего значения
 }
 unsigned short read_temp(void)
 {
  unsigned short temp_;
  temp_=result;
 //====задержка подпрограммы (DELAY)====
 void delay(unsigned short i)
 {
  for(;i>0;i--);
 }
  return temp_;
 }
```

2.2.3. Лабораторная работа №3
Вольтметр

Цель работы: изучить принцип аналого-цифрового преобразования входного напряжения и выдать результат на ЖК-дисплей.

Для начала работы необходимо запустить программу PROTEUS ISIS, открыть файл *voltmetr.dsn*. На рабочем поле программы появится схема, состоящая из микроконтроллера PIC 16F876A, цепи питания, источника напряжения на 5В с подстроечным резистором на порте RA0 и ЖК дисплея, подключенного к портам RC0-RC7. Для запуска эмуляции работы схемы нажать на кнопку PLAY. Подстроечный резистор на порте Vref+ изменяет опорное напряжение на микроконтроллере и влияет на размер квантового шага согласно формуле: $\Delta = \frac{U_{опор}}{2^n}$. Таким образом, на дисплее отображается напряжение, подаваемое на вход порта RA0.

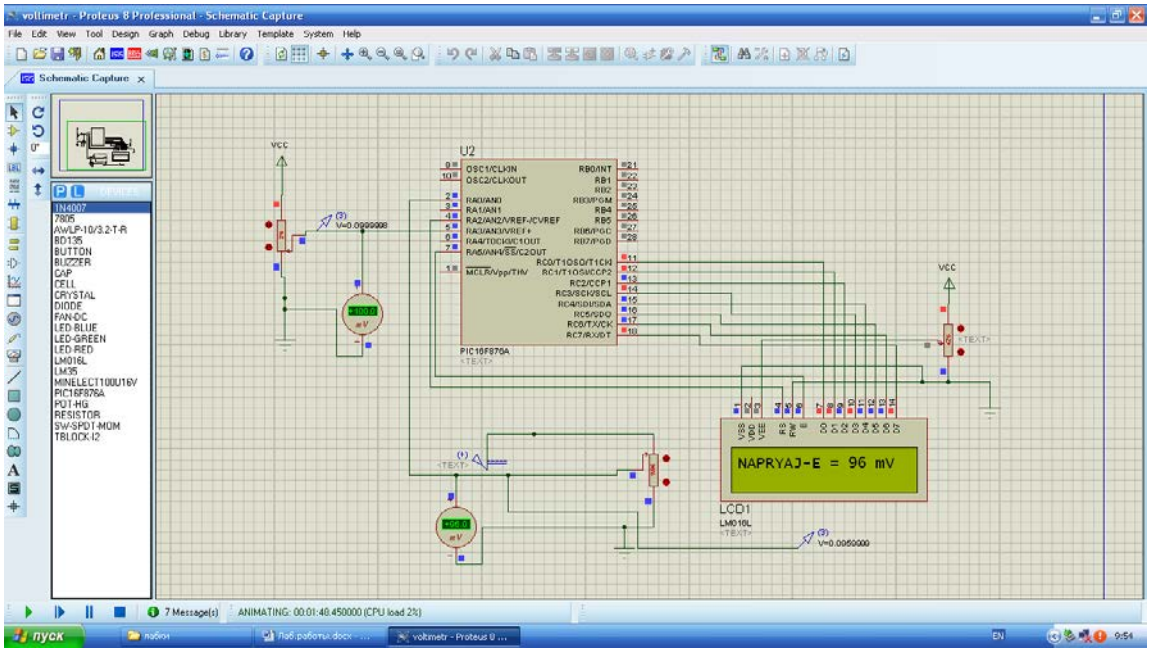

Рис. 2.2.3. Лабораторная работа №3

Код программы:

```
//====секция include====
#include<pic.h>
//====секция конфигурации====
__CONFIG (0x3F32);
//====определение портов ввода/вывода=====
#define          lcd                    PORTC
```

```
#define     RS              RA2
#define     E               RA5
#define     CHANNEL0 0b10000001  // AN0
//====прототип функции====
void e_pulse(void);
void delay(unsigned short i);
void send_char(unsigned char data);
void send_config(unsigned char data);
void lcd_goto(unsigned char data);
void lcd_clr(void);
void dis_num(unsigned long data);
//void increment(unsigned long data);
void read_adc(void);
//====основной блок (main)====
unsigned short result;
unsigned short temp;
float hm=0;
void main(void)
{
  ADRESH=0;      //очистка A/D результата
  ADRESL=0;      //очистка A/D результата
  //установка регистра ADCON1
  ADCON1=0b10000101;          // A/D правосторонняя ориентация результата конфигурирование RA2 и RA5 как цифровой ввод/вывод
  TRISA=0b01001001;           //конфигурация PORTA направления ввода/вывода
  TRISC=0b00000000;           //конфигурирование PORTC как вывод
  PORTA=0;                 // выставление порта в нулевое значение
  while(1)            //  начало 1-го цикла
  {
      send_config(0b00000001);      //очистка display at lcd
      send_config(0b00000010);      //возврат LCD в начало
      send_config(0b00000110);      //режим ввода курсора увеличивается на 1
      send_config(0b00001100);      //вкл-е дисплея, выкл-е курсора выкл-е мигания курсора
      send_config(0b00111000);      //установка функции
      lcd_goto(0);                 // запуск курсора с начала
   // отображение символов на LCD
      send_char('N');
      send_char('a');
      send_char('p');
      send_char('r');
      send_char('-');
      send_char('y');
      send_char('e');
      send_char('=');
      while(1)                     //  начало 2-го цикла
      {
      ADCON0=CHANNEL0;
      lcd_goto(8);              // вывод на дисплей с 8-ой позиции
      read_adc();           // чтение порта RA0 и передача в подпрограмму ADC
```

```
            temp=result;            // запись значения в temp
            dis_num(temp/1);     // целая часть
            send_char('.');
            dis_num(temp%1);     // дробная часть
            send_char('m ');
            send_char('V');
            send_char(' ');
            send_char(' ');
            delay(500);     // задержка 500
  }
  }
}
//====Настройка подпрограммы LCD ====
void send_config(unsigned char data)
{
  RS=0;
  lcd=data;
  delay(10);
  e_pulse();
}
void e_pulse(void)
{
  E=1;
  delay(100);
  E=0;
  delay(100);
}
void send_char(unsigned char data)
{
  RS=1;
  lcd=data;
  delay(100);
  e_pulse();
}
void lcd_goto(unsigned char data)
{
  if(data<16)
  {
            send_config(0x80+data);
  }
  else
  {
            data=data-20;
            send_config(0xc0+data);
  }
}
void lcd_clr(void)
{
  RS=0;
  send_config(0x01);
  delay(100);   // 600 Эди
}
```

```
//====Подпрограмма отображения на ЖК====
void dis_num(unsigned long data)                    //
{
  unsigned char hundred_thousand;
  unsigned char ten_thousand;
  unsigned char thousand;
  unsigned char hundred;
  unsigned char tenth;

  hundred_thousand = data/100000;
  data = data % 100000;
  ten_thousand = data/10000;
  data = data % 10000;
  thousand = data / 1000;
  data = data % 1000;
  hundred = data / 100;
  data = data % 100;
  tenth = data / 10;
  data = data % 10;
  if(hundred_thousand>0)
  {
        send_char(hundred_thousand + 0x30); //0x30 добавляется чтобы стать
ASCII кодом
        send_char(ten_thousand + 0x30);
        send_char(thousand + 0x30);
        send_char(hundred + 0x30);
        send_char(tenth + 0x30);
        send_char(data + 0x30);
  }
  else if(ten_thousand>0)
  {
        send_char(ten_thousand + 0x30);//0x30 добавляется чтобы стать ASCII
кодом
        send_char(thousand + 0x30);
        send_char(hundred + 0x30);
        send_char(tenth + 0x30);
        send_char(data + 0x30);
  }
  else if(thousand>0)
  {
        send_char(thousand + 0x30);     //0x30 добавляется чтобы стать ASCII
кодом
        send_char(hundred + 0x30);
        send_char(tenth + 0x30);
        send_char(data + 0x30);
  }
  else if(hundred>0)
  {
        send_char(hundred + 0x30);      //0x30 добавляется чтобы стать ASCII
кодом
        send_char(tenth + 0x30);
        send_char(data + 0x30);
```

```
  }
  else if(tenth>0)
  {
        send_char(tenth + 0x30);  //0x30 добавляется чтобы стать ASCII кодом
        send_char(data + 0x30);
  }
  else send_char(data + 0x30);      //0x30 добавляется чтобы стать ASCII кодом
}
//====подпрограмма ADC====
void read_adc(void)
{
  unsigned short i;
  unsigned long result_temp=0;
                                          //  чтение значений порта RA0
  for(i=2000;i>0;i-=1)              //цикл 2000 раз для получения среднего
значения
  {
        ADGO = 1;               //ADGO это бит 2 регистра ADCON0
        while(ADGO==1);         //запуск ADC, ADGO=0 после завершения
прогресса ADC
        result=ADRESH;
        result=result<<8;       //сдвиг влево на 8 бит
        result=result|ADRESL;   //10 битный результат с ADC
        result_temp+=result;
  }
  result = result_temp/2000;        //запись среднего значения в result
}
void delay(unsigned short i)
{
  for(;i>0;i--);
}
```

2.2.4. Лабораторная работа №4
2-х кнопочное управление светодиодом

Цель работы: изучить способы управления состоянием светодиода.

Для начала работы необходимо запустить программу PROTEUS ISIS, открыть файл *2-knopki-diod.dsn.* На рабочем поле программы появится схема, состоящая из микроконтроллера PIC 16F876A, цепи питания, 2-х кнопок, подключенных к портам RA0 и RA1 соответственно и светодиода, подключенного к порту RB0. Для запуска эмуляции работы схемы нажать на кнопку PLAY. При нажатии на кнопку, подключенную к порту RA0, светодиод включается и работает до тех пор, пока не будет

нажата кнопка на порте RA1. Таким образом, происходит управление состоянием светодиода.

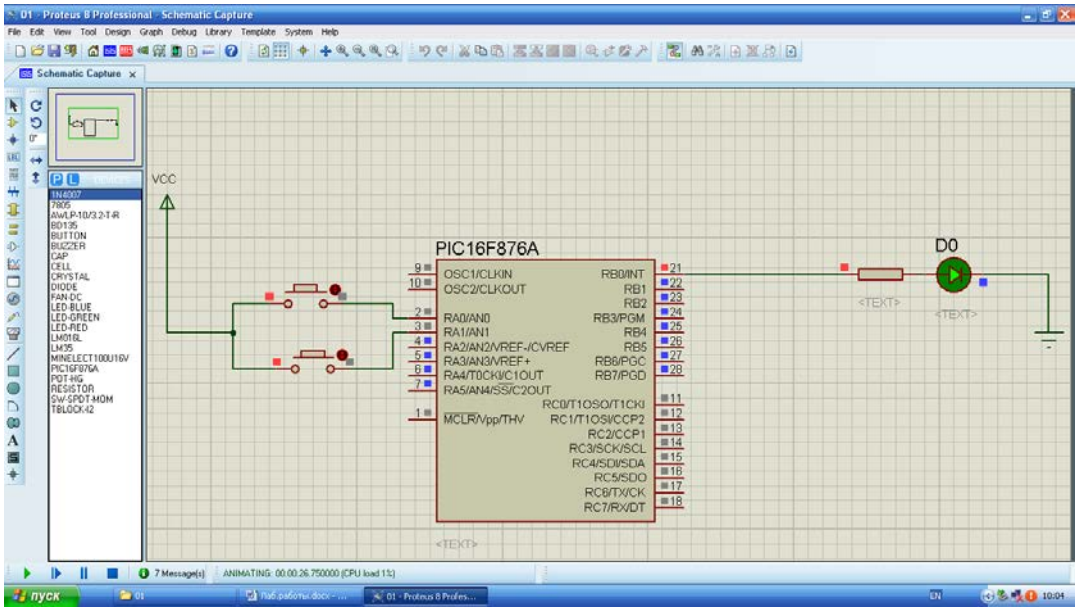

Рис. 2.2.4. Лабораторная работа №4

Код программы:

```
//====секция include====
//                k7=состояние кнопок
//                k9=состояние светодиода
//                k7=1 запуск кнопки включения
//                k7=-1 запуск кнопки выключения
//                k9=1 включение светодиода
//                k9=-1 выключение светодиода
#include<pic.h>
//====секция конфигурации====
__CONFIG (0x3F32);
//====определение портов ввода/вывода=====
#define         CHANNEL1 0b00000000  // RB0
#define         CHANNEL2 0b00000000  // RB0
#define         ledsA  PORTA
#define         ledsB  PORTB
//====прототип функции====
//void delay(unsigned short i);
void main(void)
{short int k7=0;
short int k9=-1;
  ADCON1=0b00000110;
  TRISA=0b00000011;                 //конфигурирование PORTA как вывод
      TRISB=0b00000000;             //конфигурирование PORTB как вывод
ledsB=0;
while(1)
{
```

```
if(PORTA==1)
k7=1;
if(PORTA>1)
k7=-1;
if(k7==1 &&k9==-1)
{k9=1;
ledsB=1;
}
if(k7==-1 &&k9==1)
{k9=-1;
ledsB=0;
}
}
}
```

2.2.5. Лабораторная работа №5
Термометр

Цель работы: изучить принцип работы датчика температуры LM35 и на его основе создать простейший измеритель температуры.

Для начала работы необходимо запустить программу PROTEUS ISIS, открыть файл *temperatura.dsn.* На рабочем поле программы появится схема лабораторной работы, состоящая из микроконтроллера PIC 16F876A, цепи питания, 2-х датчиков LM35, подключенных к портам RA0 и RA1 и ЖК-дисплея, подключенного к портам RC0-RC7. Для запуска эмуляции работы схемы нажать на кнопку PLAY. Датчик температуры LM35 работает следующим образом, на его вход подается питание, а выходное напряжение будет иметь следующую зависимость 10mV/1°, т.е. с изменением температуры на 1°, напряжение на выходе датчика будет изменяться на 10mV. В результате, на дисплее отображаются температуры, снимаемые с 2-х датчиков, подключенных к портам RA0 и RA1 соответственно.

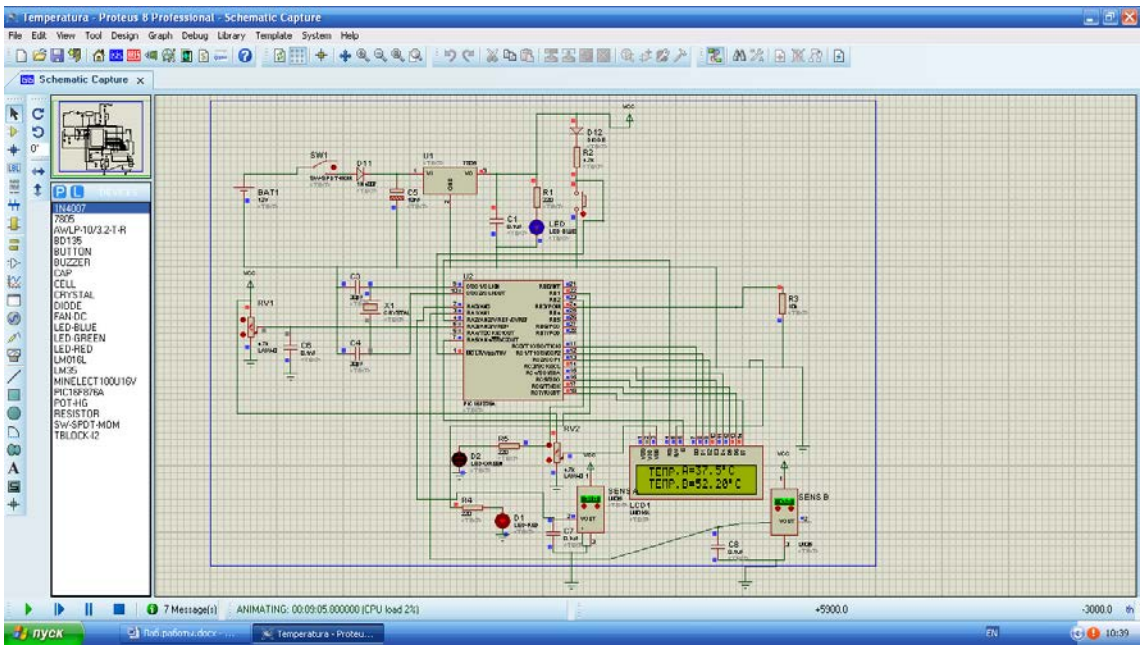

Рис. 2.2.5. Лабораторная работа №5

Код программы:

```
#include<pic.h>
//======секция конфигурации=======
__CONFIG (0x3F32);
//=======определение портов ввода/вывода=========
#define         lcd                     PORTC
#define         RS                      RA2
#define         E                       RA5
#define         CHANNEL0 0b10000001  // AN0
#define         CHANNEL1 0b10001001  // AN1
//==========прототип функции==========
void e_pulse(void);
void delay(unsigned short i);
void send_char(unsigned char data);
void send_config(unsigned char data);
void lcd_goto(unsigned char data);
void lcd_clr(void);
void dis_num(unsigned long data);
void increment(unsigned long data);
void read_adc(void);
unsigned short read_temp(void);
//====================основной блок ========================
unsigned short result;
unsigned short temp,tempA,tempB;
void main(void)
{
  ADRESH=0;                             //очистка A/D результата
  ADRESL=0;                            //очистка A/D результата

          //установка регистра ADCON1
```

```
	ADCON1=0b11000101;		// A/D правосторонняя ориентация результата
					// конфигурирование RA2 и RA5 как цифровой ввод/вывод

	TRISA=0b11011011;		//конфигурация PORTA направления ввода/вывода (I/O)
	TRISB=0b00000000;		//конфигурирование PORTB как вывод
	TRISC=0b00000000;		//конфигурирование PORTC как вывод
	PORTA=0;
	PORTB=0;
		while(1)
	{
		send_config(0b00000001);		//очистка дисплея LCD
		send_config(0b00000010);		//возврат LCD в начало
		send_config(0b00000110);		//режим ввода курсора увеличивается на 1
		send_config(0b00001100);		//вкл-е дисплея, выкл-е курсора выкл-е мигания курсора
		send_config(0b00111000);		//установка функции
		lcd_goto(0);				// запуск курсора с начала
	//===========отображение символов на LCD===========
		send_char('T');
		send_char('e');
		send_char('m');
		send_char('p');
		send_char('');
		send_char('');
		send_char('A');
		send_char('=');
		lcd_goto(20);				/курсор идет на 2-ую строку LCD
	//=================отображение символов на LCD===========
		send_char('');
		send_char('T');
		send_char('E');
		send_char('M');
		send_char('P');
		send_char('.');
		send_char('B');
		send_char('=');

		while(1)				//бесконечный цикл
		{
		//==============датчик A=========
		ADCON0=CHANNEL0;			//CHANNEL1=0b10001001
		lcd_goto(8);
		read_adc();
		temp=read_temp();
		dis_num(temp/10);
		send_char('.');
		dis_num(temp%10);
		send_char(0b11011111);
```

```
        send_char('C');
        send_char(' ');
        send_char(' ');
                tempA=temp;


  //===========датчик B===============
        ADCON0=CHANNEL1;                        //CHANNEL0=0b10000001

        lcd_goto(28);
        read_adc();
        temp=read_temp();
        dis_num(temp/10);
        send_char('.');
        dis_num(temp%100);
        send_char(0b11011111);
        send_char('C');
        send_char(' ');
        send_char(' ');
        tempB=temp;

//============Подпрограмма настроек LCD ======================
void send_config(unsigned char data)
{
  RS=0;
  lcd=data;
  delay(10);
  e_pulse();
}
void e_pulse(void)
{
  E=1;
  delay(20);
  E=0;
  delay(20);
}
void send_char(unsigned char data)
{
  RS=1;
  lcd=data;
  delay(20);
  e_pulse();
}
void lcd_goto(unsigned char data)
{
  if(data<16)
  {
        send_config(0x80+data);
  }
  else
  {
```

```
            data=data-20;
            send_config(0xc0+data);
        }
      }
      void lcd_clr(void)
      {
        RS=0;
        send_config(0x01);
        delay(20);
      }
      void dis_num(unsigned long data)
      {
        unsigned char hundred_thousand;
        unsigned char ten_thousand;
        unsigned char thousand;
        unsigned char hundred;
        unsigned char tenth;

        hundred_thousand = data/100000;
        data = data % 100000;
        ten_thousand = data/10000;
        data = data % 10000;
        thousand = data / 1000;
        data = data % 1000;
        hundred = data / 100;
        data = data % 100;
        tenth = data / 10;
        data = data % 10;
        if(hundred_thousand>0)
        {
            send_char(hundred_thousand + 0x30); //0x30 добавляется чтобы стать
ASCII кодом
            send_char(ten_thousand + 0x30);
            send_char(thousand + 0x30);
            send_char(hundred + 0x30);
            send_char(tenth + 0x30);
            send_char(data + 0x30);
        }
        else if(ten_thousand>0)
        {
            send_char(ten_thousand + 0x30);//0x30 добавляется чтобы стать ASCII
кодом
            send_char(thousand + 0x30);
            send_char(hundred + 0x30);
            send_char(tenth + 0x30);
            send_char(data + 0x30);
        }
        else if(thousand>0)
        {
            send_char(thousand + 0x30);     //0x30 добавляется чтобы стать ASCII
кодом
            send_char(hundred + 0x30);
```

```
            send_char(tenth + 0x30);
            send_char(data + 0x30);
      }
      else if(hundred>0)
      {
            send_char(hundred + 0x30);      //0x30 added to become ASCII code
            send_char(tenth + 0x30);
            send_char(data + 0x30);
      }
      else if(tenth>0)
      {
            send_char(tenth + 0x30);  //0x30 добавляется чтобы стать ASCII кодом
            send_char(data + 0x30);
      }
    else send_char(data + 0x30);        //0x30 добавляется чтобы стать ASCII кодом
    /*else {
    send_char('0');
    send_char(data + 0x30);*/
    //}
    }
    void increment(unsigned long data)
    {
      unsigned short j;
      for(j=10;j>0;j--)
      {     lcd_goto(32);
            data=data+1;
            dis_num(data);
            delay(10000);
      }
    }
    //==================Подпрограмма АЦП=========================
    void read_adc(void)
    {
      unsigned short i;
      unsigned long result_temp=0;
      for(i=2000;i>0;i-=1)                  //цикл 2000 раз для получения среднего
значения
      {
            ADGO = 1;                           //ADGO это бит 2 регистра
ADCON0
            while(ADGO==1);               //запуск ADC, ADGO=0 после оконч-я
прогресса ADC
            result=ADRESH;
            result=result<<8;             //сдвиг влево на 8 бит
            result=result|ADRESL;         //10 битный результат с ADC
            result_temp+=result;
      }
      result = result_temp/2000;          //получение среднего значения
    }
    unsigned short read_temp(void)
    {
      unsigned short temp;
```

```
  temp=result;
  return temp;
}
//================== задержка подпрограммы (DELAY)======
void delay(unsigned short i)
{
  for(;i>0;i--);
}
```

3. ЭКОНОМИЧЕСКАЯ ЧАСТЬ

1. Технико-экономическое обоснование проекта.
2. Определить объём инвестиций.
    - Стоимость основных фондов
    - Объём инвестиций на покупку материально-производственных запасов
    - Объём инвестиций на покупку малоценного инвентаря и контрольно-измерительных приборов
    - Расчёт заработной платы производственных рабочих
3. Определить экономическую эффективность, годовой доход
4. Определить срок окупаемости инвестиций

а) Технико-экономическое обоснование проекта

- Цель, сущность и задачи проекта и его актуальность
- Экономическая эффективность проекта

б) Определить объём инвестиций

В нижеследующих таблицах даны приведённые затраты на ВКР.

Таблица 3.1. Объем инвестиций на покупку инвентаря и контрольно-измерительных приборов.

№	*Наименование*	*Кол.*	*Цена за единицы, сум*	*НДС 20%, сум*	*Стоимость с учетом НДС, сум*	*Общая стоимость с учетом НДС, сум*
1	Мультиметр	1	60000	12 000	72 000	72 000
2	Осциллограф	1	1200000	240 000	1 440 000	1 440 000
3	Программатор PicKit3	1	390000	78 000	468 000	468 000

4	Микроконтроллер PIC-16F876A	1	12 000	2 400	14 400	14 400
5	Дисплей LCD 1602	1	20 000	4 00	24 000	24 000
6	Печатная плата	1	8 000	1 600	9 600	9 600
7	Провод МГТФ	1	2 500	500	3 000	3 000
	Итого					2 031 000

Таблица 3.2. Стоимость основных фондов.

№	*Наименование основных фондов*	*Кол.*	*Стоимость одного основного фонда, сум*	*Всего стоимость основных фондов, сум*
1	Компьютер	1	900 000	900 000
2	Принтер	1	400 000	400 000
3	Сверлильно-фрезеровальный станок Bungard CCD/2	1	2 500 000	2 500 000
4	Цифровая паяльная станция ATTEN AT80D	1	140 000	140 000
	Итого			**3 940 000**
	Затраты на текущий ремонт и техническое обслуживание	12 % от стоимости ОФ		**472 800**
	Амортизационные отчисления составляет	20 % от стоимости ОФ		**788 000**

Таблица 3.3. Расчет заработной платы производственных рабочих.

№	*Должность*	*Кол.*	*Рабочие дни в году*	*Дневная заработная плата*	*Месячная заработная плата*
1	Руководитель проекта	1	270	54 200	650 400
2	Исполнитель проекта	1	270	30 000	360 000
	Итого			5 630 000	**1 010 400**
3	Основная заработная плата	сумма оплаты труда всех рабочих и премии в размере 40%			404 160
4	Дополнительной заработной платы производственных рабочих	10 % от основного з/п			40 416
5	Фонд оплаты труда	сумма основной и дополнительной з/п			1 454 976
6	Затраты на социальное страхование	25 % от ФОТ			363 744
7	Транспортные расходы	20 % от $З_{осн}$			80 832

Таблица 3.4. Смета затрат на проведение разработки программного обеспечения.

№	Наименование статей затрат	Сумма
1.	***Электроэнергия (W)***	
1.1.	Установленная мощность (N), кВт	0,5
1.2.	Время работы (T), час	186
1.3.	Стоимость электроэнергии за 1 кВТ (S), сум	155
	Всего расход на электроэнергию	***14415***

2.	***Действительный годовой фонд времени ЭВМ (Тпк)***	
2.1.	Количество месяцев в году (Nм), месяц	12
2.2.	Количество рабочих дней в месяце (Nд), день	22
2.3.	Средняя продолжительность рабочего дня (Nч), час	8
	Действительный годовой фонд времени ЭВМ (Тпк), часов/год	*2112*
3.	Расходы периода	**15000**
4.	***Стоимость машино-часов***	
4.1.	Затраты на амортизацию - годовые издержки на амортизацию (За), сум в год	442000
4.2.	Годовые издержки на вспомогательные материалы (Звм), сум в год	15000
4.3.	Затраты на текущий ремонт компьютера (Зт), сум в год	265200
	Цена машино-час (С), сум/год	***342***
5.	*Стоимость машинного времени (Звм)*	
5.1.	Цена машино-часов (С), рассчитывается;	*342*
5.2.	Затраты времени на программирование в часах (tn)	
5.3.	Затраты времени на отладку программы в часах (tотл)	
	Стоимость машинного времени	*0*
5	Фонд оплаты труда	**1 454 976,00**
6	Социальное страхование	**363 744,00**
7	Амортизация	**788 000,00**
	Затрат на проведение разработки программного обеспечения	2 636 135,00

Таблица 3.5. Расчет экономической эффективности.

№	*Наименование показателей*	*Единица измерения*	*Сумма*
1	Себестоимость продукта, С	сум	**2 636 135**
2	Объем производства, Q	мБт	15000
3	Расчет затраты по ВКР	сум/мБт	175,74
4	Реальная затрата на производство продукции	сум/мБт	228,47
5	Экономическая эффективность, Э	сум	52,72
6	Экономическая эффективность, Э	%	23,07692308

Таблица 3.6. Расчет инвестиций.

№	*Наименование показателей*	*Единица измерения*	*Сумма*
1	Офонд	сум	3 940 000
2	Мин	сум	2 031 000
3	Инвестиции	сум	5971000

Таблица 3.7. Расчет рентабельности выполненных работ.

№	*Наименование показателей*	*Единица измерения*	*Сумма*	*Примечание*
1	Затраты на производство	сум	2 636 135	В год
2	Инвестиции	сум	5 971 000	Всего
	Цена продукции	сум	3 163 362	
3	Прибыль от производства	сум	527 227	В год
4	Срок окупаемости	месяц	5	
5	Рентабельность	%	8,8	

4. БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

ЗАКЛЮЧЕНИЕ

При выполнении выпускной квалификационной работы были проведены разработка и изготовление схемы лабораторного стенда, на базе микроконтроллера PIC для работы с периферией, предназначенного для ознакомления студентов с типовыми схемными решениями микроконтроллерных устройств различного назначения.

Лабораторный стенд для микроконтроллеров серии PIC типов 16F873A/876A обеспечивает работу с такими периферийными устройствами, как ЖК-дисплей, индикатор состояния всех выводов портов.

При проведении работ были решены следующие задачи:

1. Произведен анализ имеющейся литературы, выбрана конфигурация микроконтроллерной системы, пригодной для реализации в качестве учебного прибора.
2. Разработаны схемы аналоговых и цифровых узлов стенда.
3. Выбран микроконтроллер, который полностью удовлетворяет требованиям, а также является наименее дорогим.
4. Создана принципиальная схема и разработана печатная платы.
5. Осуществлена наладка схемы стенда.

Разработанная конструкция лабораторного стенда для работы с микроконтроллерами PIC может быть успешно использована для ознакомления студентов с типовыми схемными решениями микроконтроллерных устройств различного назначения в ВУЗах Республики Узбекистан.

СПИСОК ЛИТЕРАТУРЫ

1. Дитер Кохц «Измерение, управление и регулирование с помощью PIC микроконтроллеров», Издательство: МК-Пресс, ISBN 978-966-8806-15-5; 2015 г.

2. Николай Заец «Радиолюбительские конструкции на PIC-микроконтроллерах», Издательство: МК-Пресс, Корона-Век, ISBN 978-5-7931-0942-0, 978-966-8806-42-1; 2015 г.

3. Сид Катцен «PIC-микроконтроллеры. Полное руководство», Издательство: Додэка, ДМК Пресс, ISBN 978-5-97060-109-9; 2014 г.

4. Юрий Шпак «Программирование на языке С для AVR и PIC микроконтроллеров» , Издательство: Корона-Век, МК-Пресс, ISBN 978-5-7931-0842-3, 978-966-8806-67-4; 2012 г.

5. Предко М. «PIC-микроконтроллеры. Архитектура и программирование», Издательство: МИР, ISBN 978-5-94074-534-1; 2009 г.

6. Сид Катцен «PIC-микроконтроллеры. Все, что вам необходимо знать», Издательство: Додэка XXI, ISBN 978-5-94120-134-1, 0-7506-7698-1; 2008 г.

7. Тим Уилмшурст «Разработка встроенных систем с помощью микроконтроллеров PIC. Принципы и практические примеры», Издательство: МК-Пресс, Корона-Век, ISBN 978-5-903383-61-0, 978-966-8806-46-9, 978-0-7506-6755-5; 2008 г.

8. Юрий Магда «Современные микроконтроллеры. Архитектура, программирование, разработка устройств», Издательство: ДМК Пресс, ISBN 978-5-94074-882-3; 2012 г.

9. Бродин В. Б., Шагурин И. И. «Микроконтроллеры. Архитектура, программирование, интерфейс», Издательство: Эком, , ISBN: 5-7163-020-0, 1999

10. Фрунзе А. В. «Микроконтроллеры? Это же просто!», Издательство: ИД Скимен, ISBN: 5-94929-003-7, 2003

11. Электронный журнал "Радиоежегодник" - Выпуск 24. PROTEUS по-русски, 2013

12. Электронный журнал "Радиоежегодник" - Выпуск 33. Микроконтроллеры - книги от А до Я, 2014

13. Электронный журнал "Радиоежегодник" - Выпуск 9. Микроконтроллеры PIC, 2012

14. http://www.microchip.ru/files/d-sheets-rus/mplab_ide.pdf

15. http://www.pickit2.ru/doku.php/pickit2

16. http://proteus123.narod.ru/

17. https://ru.wikipedia.org/wiki/MPLAB