Начало работы в среде разработки Microchip MPLAB X. Создание USB CDC-устройства на базе PIC18F4550.

Вторая часть.

Оглавление

Перечень рисунков:

Введение.

В предыдущей статье обсуждались общие вопросы создания устройства, которое связывается по протоколу USB с компьютером. Микроконтроллер настроен как CDC устройство и компьютер воспринимает его как «виртуальный com-порт». Актуальность проблемы связана с тем, что на современных персональных компьютерах RS-232 больше не устанавливают и как-то нужно выходить из сложившейся ситуации при разработке новых устройств.

В этой статье обсуждается извлечение исходного текста из USB стека MLA для устройства PICDEM FS USB, настройка и использование прерываний от TIMER0 и совместная работа таймера и модуля USB.

Для прочтения этой статьи следует ознакомиться с предыдущей статьёй. Особых знаний о стандарте USB не требуется, необходимо знание базовых конструкций языка СИ и умение работать в Windows. Программы, которые необходимы для работы, можно скачать с сайта www.microchip.com и установить на компьютер:

Среда разработки MPLAB X IDE (версия 3.26),

Библиотека microchip library application (mla_v2015_08_10),

Компилятор языка СИ – XC8 (при написании статьи использована версия 1.36).

Возможно, с сайта www.oracle.com понадобится Java, для установки среды разработки.

Глава 1. Создание отдельного проекта.

Представляется целесообразным создать отдельный проект, чтобы исходные тексты программы остались неизменными в качестве резерва. Создаем папку, например, CDC_Standalone. Папку лучше создать там, где MPLAB X IDE создаёт по умолчанию все свои проекты. Открываем проект:

C:\microchip\mla\v2015_08_10\apps\usb\device\cdc_basic\firmware\src

копируем 8 файлов – 4 заголовочных и 4 исходных текста си:

usb_descriptors.c, main.c, app_led_usb_status.c, app_device_cdc_basic.c

usb_config.h, system_config.h, app_led_usb_status.h,

app_device_cdc_basic.h

Затем из папки

C:\microchip\mla\v2015_08_10\apps\usb\device\cdc_basic\firmware\src\system_config\picdem_fs_usb копируем 4 файла:

fixed_address_memory.h, io_mapping.h, system.h, system.c

В папке C:\microchip\mla\v2015_08_10\framework\usb\src

находится 3 файла, которые мы тоже скопируем:

usb_device.c, usb_device_cdc.c, также скопируем отсюда файл usb_device_local.h, который не отображается в исходном проекте, но потребуется файлу usb_device.c во время компиляции нашего проекта.

Из папки C:\microchip\mla\v2015_08_10\framework\usb\inc

копируем 7 файлов:

usb.h, usb_ch9.h, usb_common.h, usb_device.h, usb_device_cdc.h, usb_hal.h, usb_hal_pic18.h

Из папки C:\microchip\mla\v2015_08_10\bsp\picdem_fs_usb

копируем оставшиеся семь файлов:

leds.c, buttons.c, adc.c, power.h, leds.h, buttons.h, adc.h

Далее, когда все 29 файлов скопированы в папку CDC_Standalone, создаём новый проект (рис. 1).

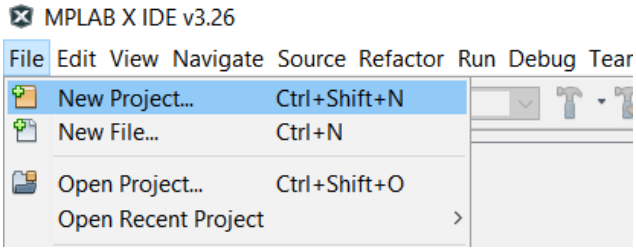

Рис. 1. Создание нового проекта.

Выбираем в поле Categories: Microchip Embedded, а в поле Projects: Standalone Project (рис. 2).

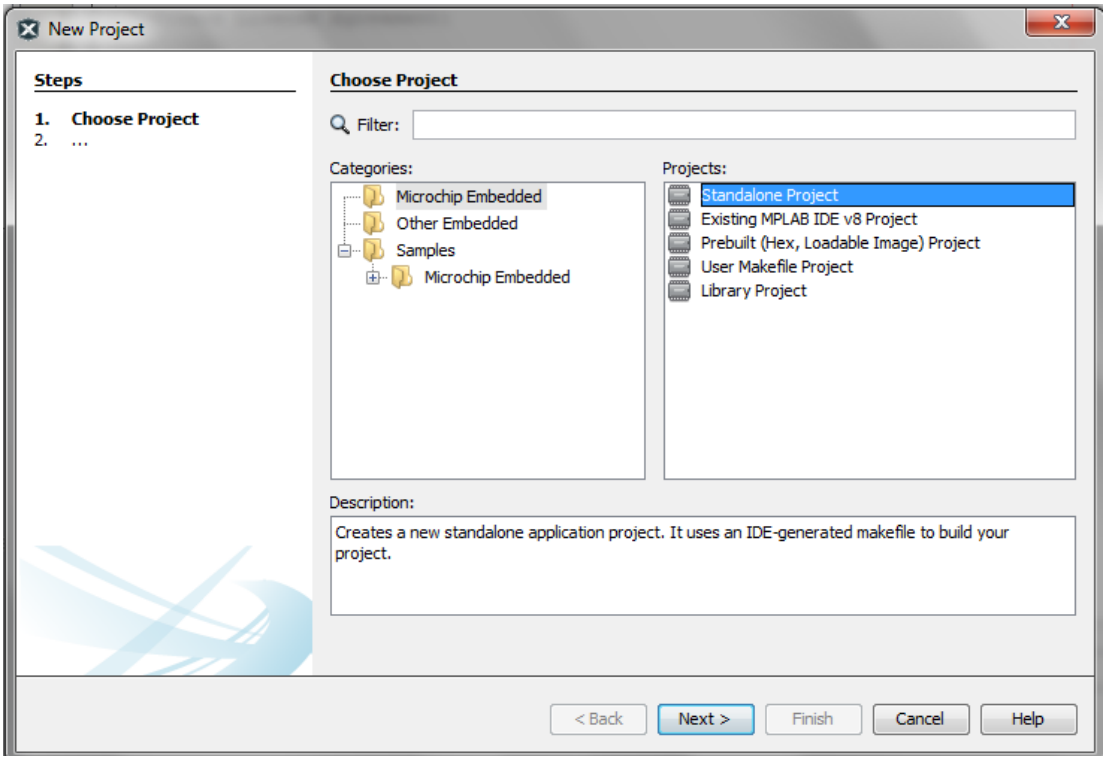

Рис. 2. Выбор типа проекта.

Затем нажимаем кнопку Next> В появившемся окне в поле Family выбираем Advanced 8-bit MCUs (PIC18), микроконтроллер выбираем в поле Device: PIC18F4550 (рис. 3). На следующем шаге в качестве отладчика можно выбрать симулятор (рис. 4). После этого выбираем имеющийся компилятор (рис. 5), на следующем шаге задаём место расположения проекта и выбираем кодировку win1251 для корректного отображения русских символов в комментариях (рис. 6). После того, как параметры проекта заданы, нажимаем кнопку Finish. Убрать другие проекты из окна Project можно, если выделить закрываемый проект, нажав правую кнопку мыши и выбрать Close (рис. 7). Присоединяем к своему проекту заранее подготовленные файлы, которые мы скопировали из MLA. Повторять структуру исходного проекта не обязательно. Файлы с расширением *.h присоединим к папке Hider Files, нажав правой кнопкой мыши и выбрав Add Existing Item (рис. 8). В появившемся окне находим скопированные файлы, которые имеют расширение *.h. Аналогичным образом присоединяем исходные файлы с расширением *.c к проекту, но к папке Source Files.

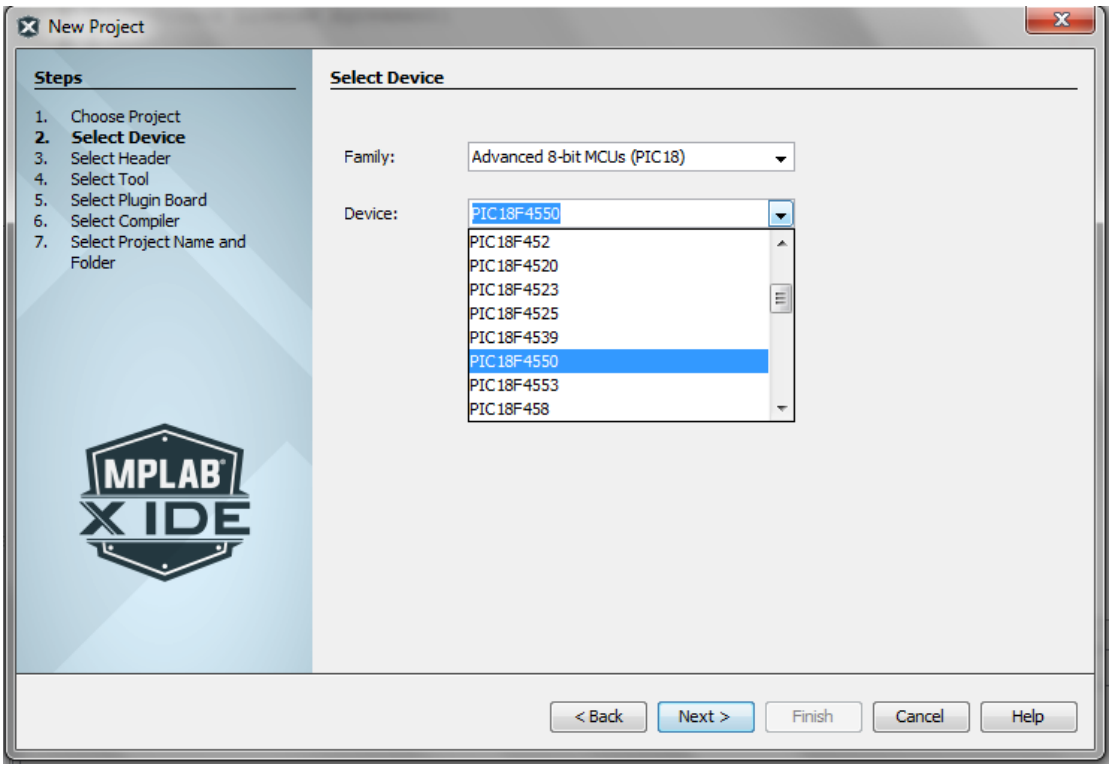

Рис. 3. Выбор типа микроконтроллера.

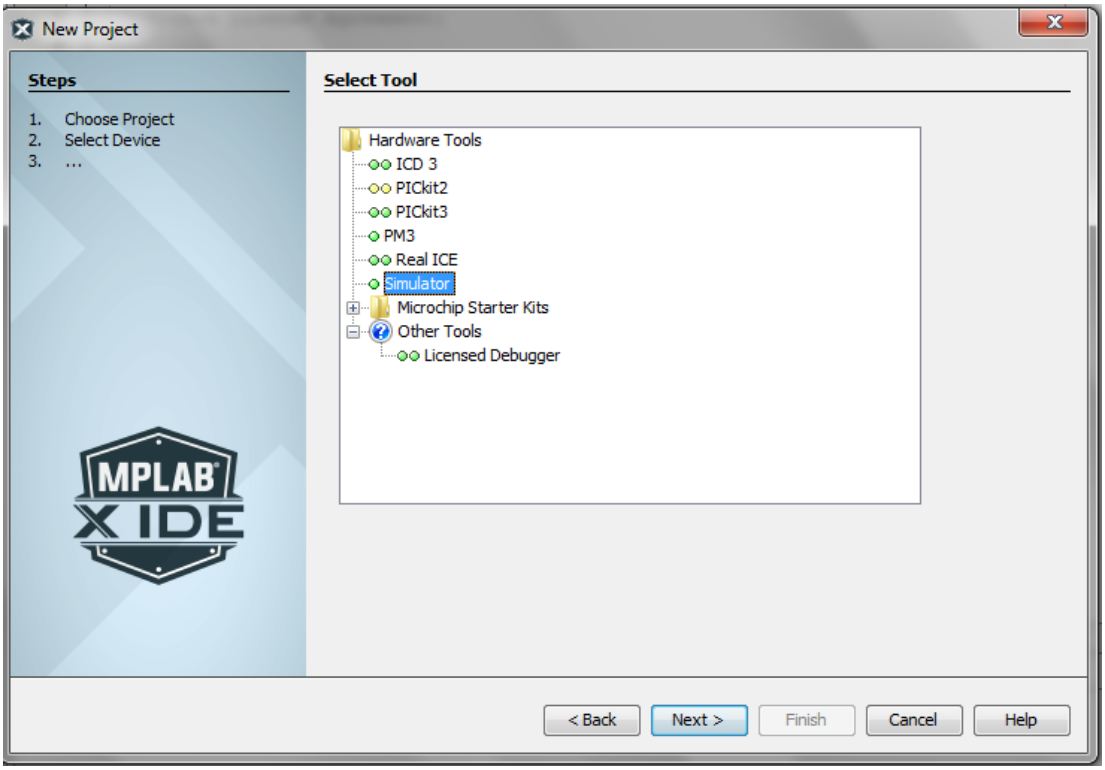

Рис. 4. Выбор отладчика.

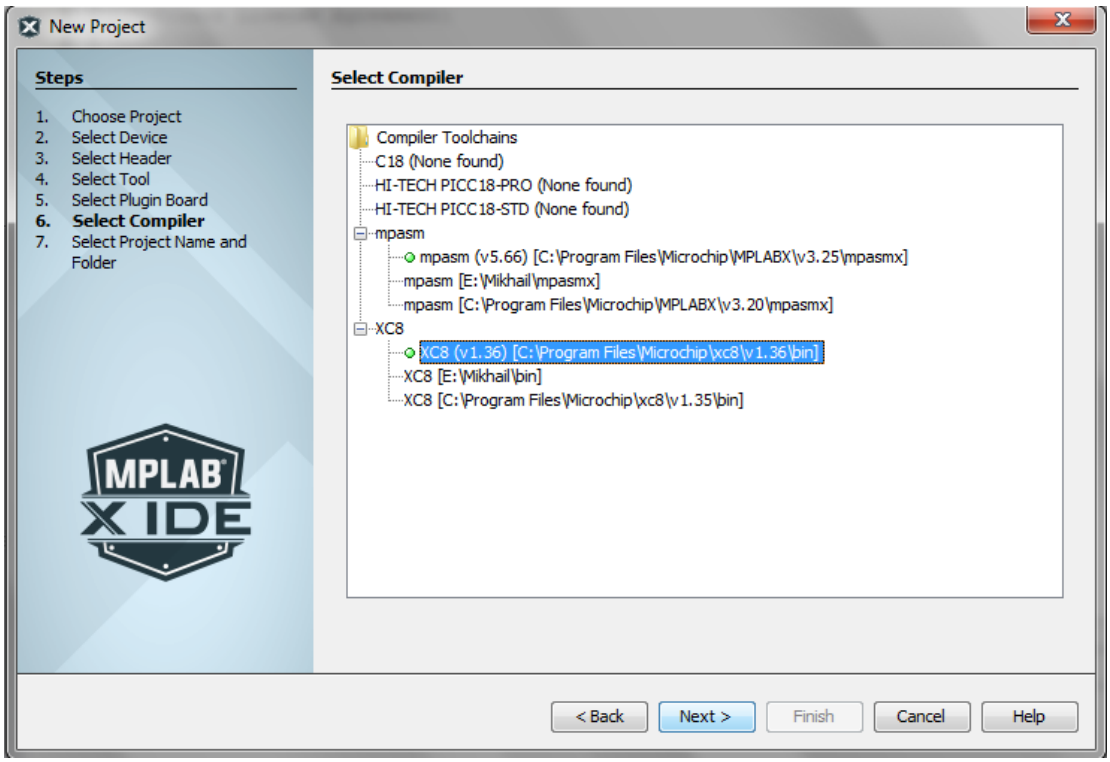

Рис. 5. Выбор компилятора.

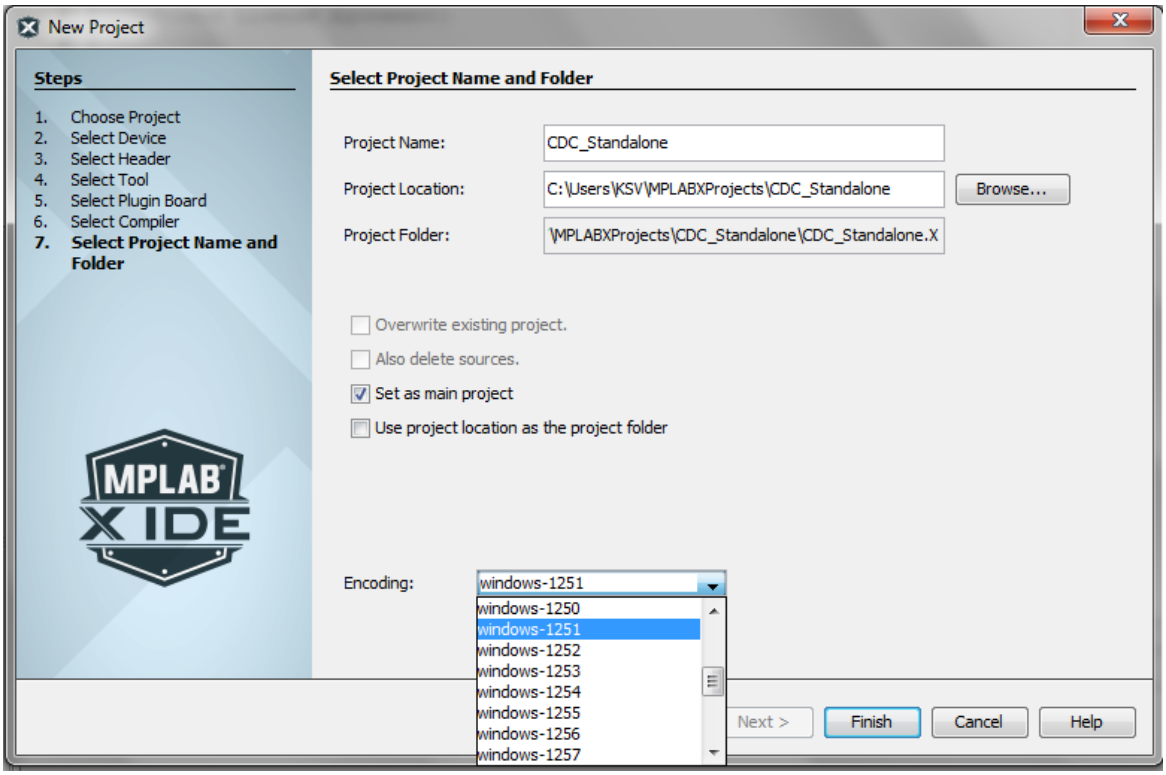

Рис. 6. Задаем параметры проекта.

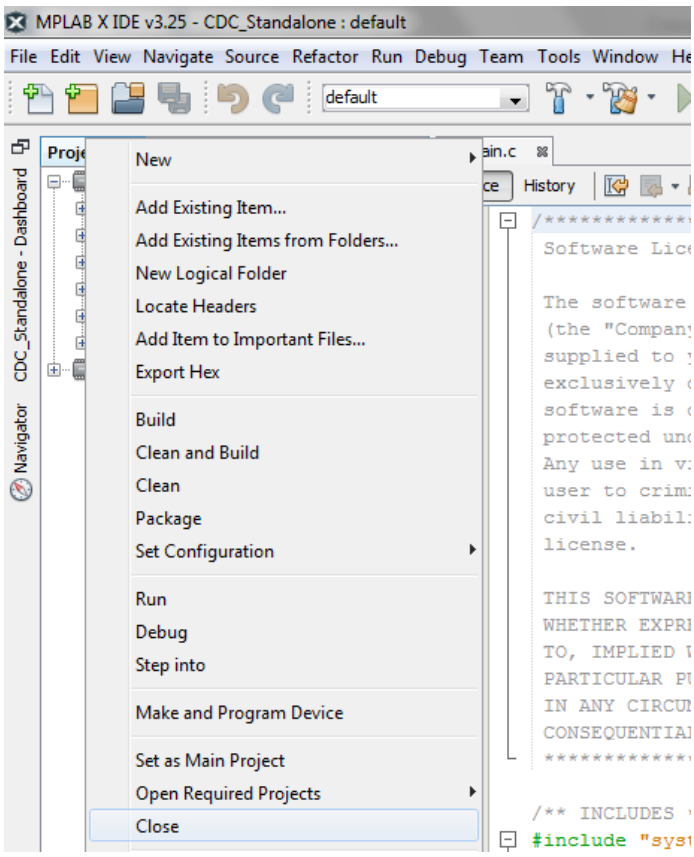

Рис. 7. Закрытие проекта.

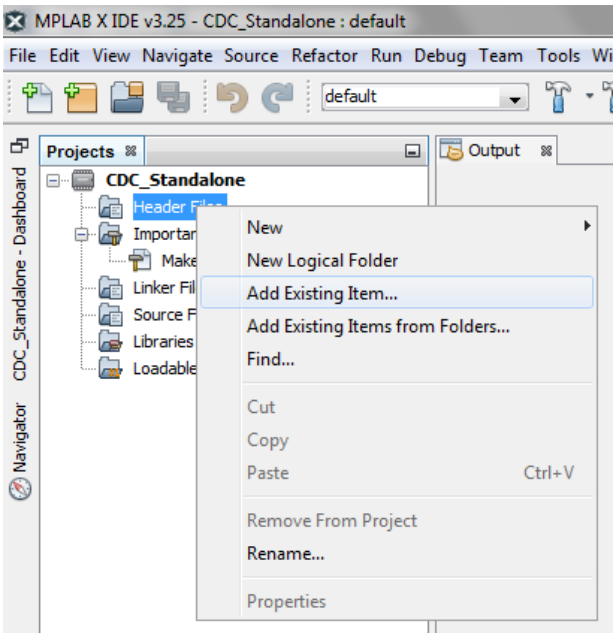

Рис. 8. Прикрепление файла к проекту.

После компиляции возникнут ошибки. Это связано, в основном, с указанием на расположение файлов. Для устранения ошибок нужно щёлкнуть левой кнопкой мыши на описании ошибки в окне Output (рис. 9). После этого появится удобное указание на ошибку (рис. 10). Исправляем строку #include <adc.h> на #include "adc.h", ошибка исчезает. Далее

подобным образом исправляем все ошибки, связанные с расположением файлов.

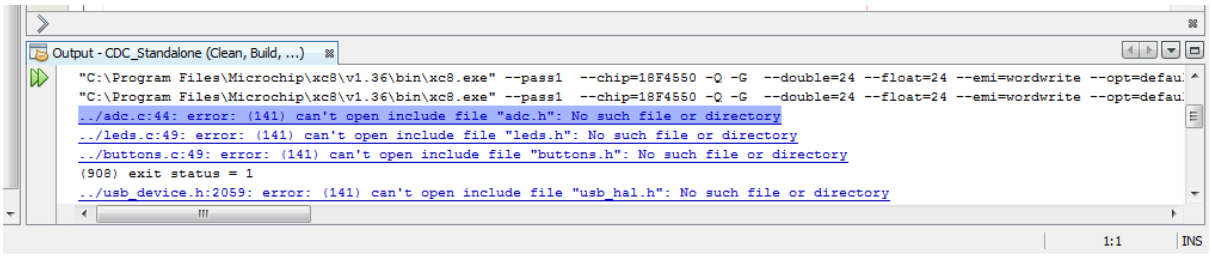

Рис. 9. Ошибка после компиляции.

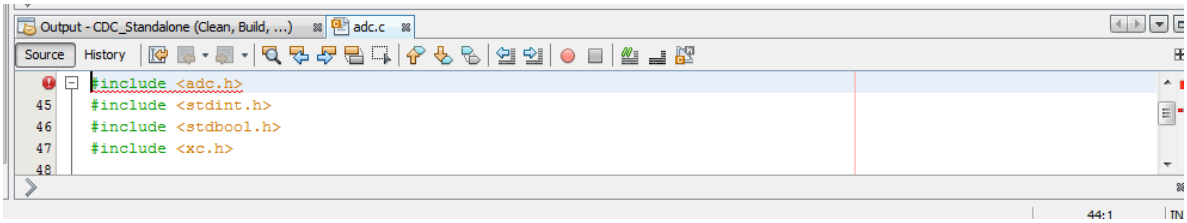

Рис. 10. Указание на ошибку.

После того, как компиляция удалась, останется большое количество предупреждений. Что легко объяснить, если сравнить уровень предупреждений в нашем проекте и в исходном проекте, который поставляется с библиотекой MLA. Для этого открываем проект, наводим курсор, щёлкаем правой кнопкой мыши выбираем Properties (рис. 12), затем заходим в активный профиль и смотрим настройки компилятора, наведя курсор на строку XC8 compiler и щёлкнув левой кнопкой мыши. В графе Warning Level имеется установка «-3», а в библиотеке MLA «0». Зададим уровень «0» в своём проекте (рис. 11). Конечно, было бы очень хорошо проработать каждое предупреждение, но оставим большое количество предупреждений на совести разработчика. Пока для нас главное, что проект компилируется.

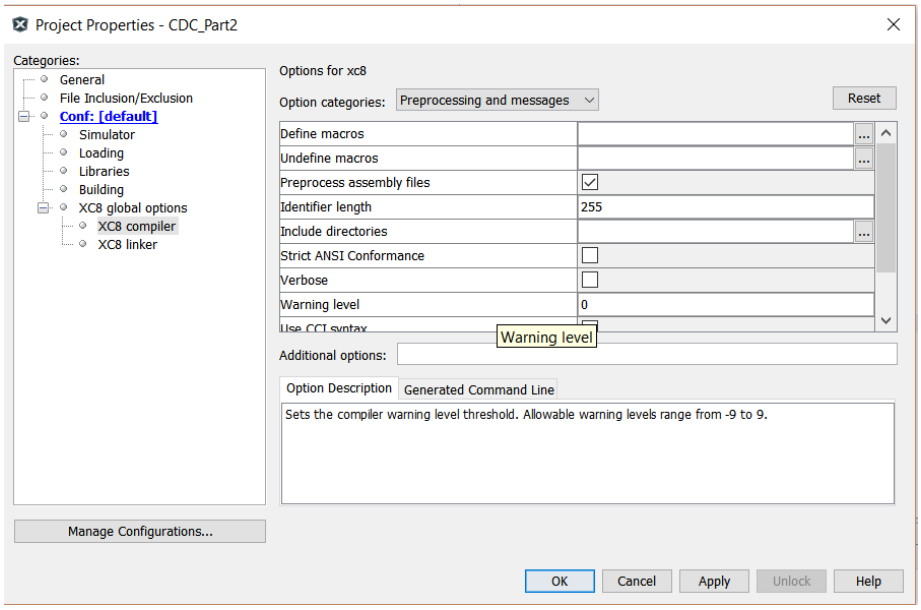

Рис. 11. Настройка “Warning level” компилятора.

Полагаем, что загрузчик уже прошит в плату PICDEM FS USB. Чтобы случайно не испортить загрузчик, хорошо будет сделать следующее: откроем system.c и сделаем предупреждения-напоминания, добавив такой текст:

#warning For compatibility with bootloader set Codeoffset 0x1000 in Additional options of XC8 Linker.

#warning For compatibility with bootloader set ROM ranges: default,-0-FFF,-1006-1007,-1016-1017 in Memory model of XC8 Linker.

Строки предупреждения можно поместить, например, сразу после описания битов конфигурации (#pragma config…). Теперь при каждой компиляции появится предупреждение.

Чтобы проект успешно загрузился и работал с загрузчиком HID Bootloader установим необходимые ограничения по расположению программы в памяти. (см. файл Read me Usage Notes for Bootloader with XC8, расположенный в папке C:\microchip\mla\v2015_08_10\apps\usb\device\bootloaders\firmware\pic18_non_j также см. [2] – главы 4.8.21 --CODEOFFSET: Offset Program Code to Address и 5.9.2 Changing the Default Interrupt Function Allocation)

Во-первых, смещение для программы. Для этого наведём мышку на папку нашего проекта (папка с изображением микросхемы) и щёлкнем правой кнопкой. В появившемся окне настроек проекта выберем самую нижнюю строку – Properties (рис. 12). Затем выделим строку XC-8 linker, в выпадающем списке Option categories выбираем Additional options, в графу Code offset вписываем адрес 0x1000, с которого будет начинаться программа и нажимаем кнопку Apply (рис. 13).

Во-вторых, в выпадающем списке Option categories выбираем Memory model, в графу ROM ranges помещаем запись default,-0-FFF,-1006-1007,-1016-1017, которая настроит компилятор таким образом, что эти адреса памяти не

будут заняты командами из нашей программы и нажимаем кнопку OK (рис. 14).

Откомпилируем программу, теперь она будет работать совместно с загрузчиком, и можно править проект без каких-то опасений, зная, что в случае необходимости проект можно легко восстановить, либо перенести, при необходимости, на другой компьютер без установки всей библиотеки MLA.

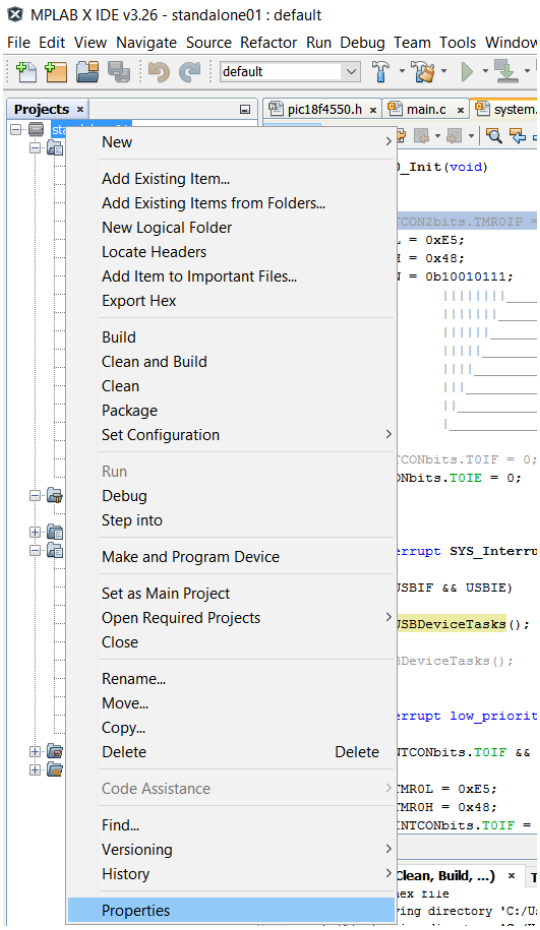

Рис. 12. Окно настройки свойств проекта.

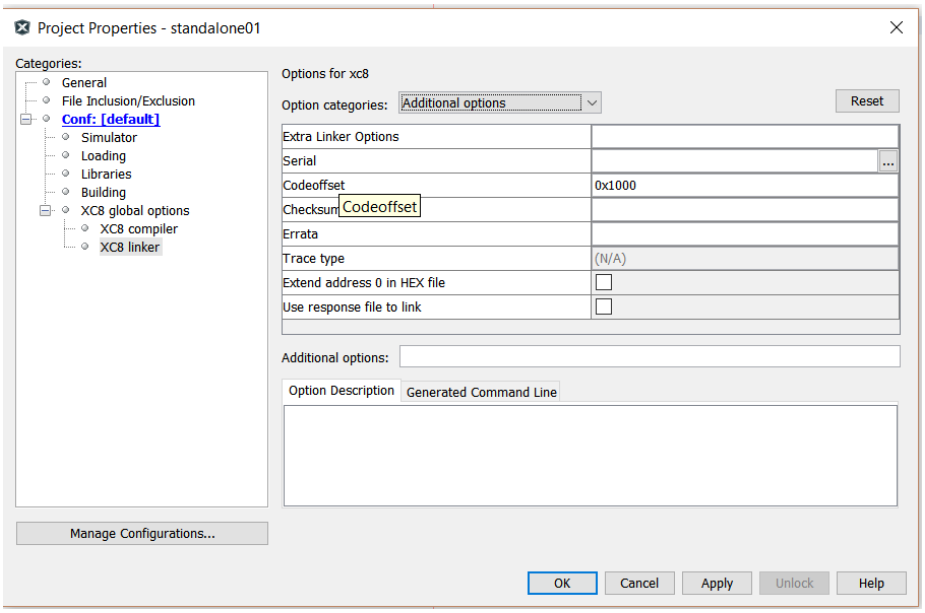

Рис. 13. Установка смещения начала программы.

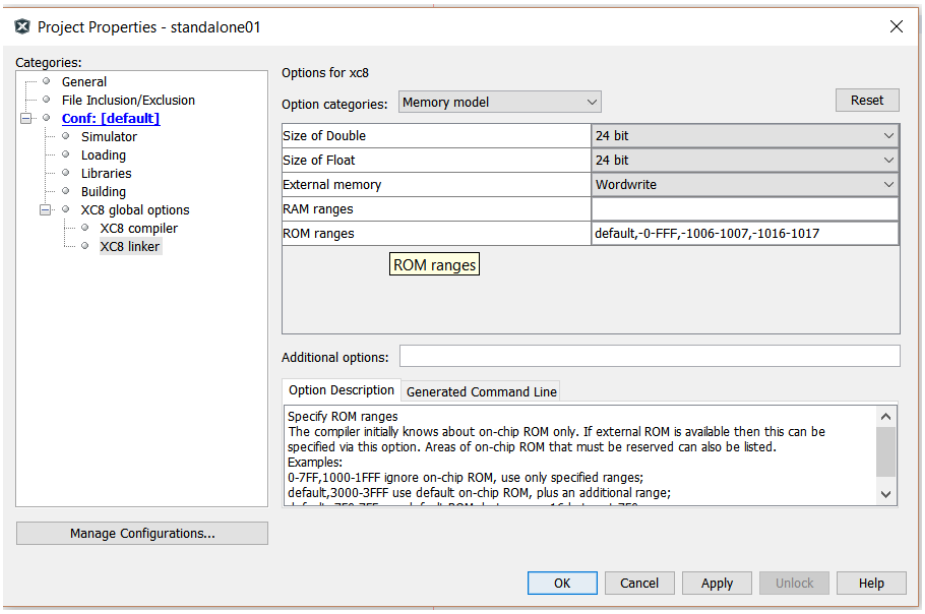

Рис. 14. Установка защиты адресов в памяти программ.

Глава 2. Управление платой PICDEM FS USB с компьютера.

Хотелось бы всё же управлять платой с компьютера. Допустим, включать и выключать светодиоды после того, как компьютер передаст определённые символы. Для работы со светодиодами в составе проекта имеется два файла: leds.c и leds.h. В файле leds.c описывается аппаратное присоединение диодов к микроконтроллеру:

```
#include <xc.h>

#define LED_D1_LAT LATDbits.LATD0
#define LED_D2_LAT LATDbits.LATD1
#define LED_D3_LAT LATDbits.LATD2
#define LED_D4_LAT LATDbits.LATD3

#define LED_D1_TRIS TRISDbits.TRISD0
#define LED_D2_TRIS TRISDbits.TRISD1
#define LED_D3_TRIS TRISDbits.TRISD2
#define LED_D4_TRIS TRISDbits.TRISD3

#define LED_ON  1
#define LED_OFF 0

#define INPUT  1
#define OUTPUT 0
```

А в файле leds.h светодиоды описаны следующим образом:

```
typedef enum
{
   LED_NONE,
   LED_D1,
   LED_D2,
   LED_D3,
   LED_D4
//     D7 = Bus powered - hard wired to power supply
//     D8 = Self powered - hard wired to power supply
} LED;
```

Далее, в файле leds.h имеются прототипы функции для работы со светодиодами, перед каждым прототипом имеется краткое описание, в котором указано название функции, какие действия выполняет, условия для работы функции, входные данные и выходные данные.

Рассмотрим, для примера, описание функции void LED_On(LED led):

Функция: void LED_On(LED led);

Описание: Включение требуемого светодиода LED

Предварительные условия: светодиод LED настраивается функцией LED_Configure()

Входные данные: LED led – светодиод, перечисленный в enum LED в файле leds.h

Выходные данные: нет.

С остальными функциями читателям будет полезно разобраться самим.

На плате PICDEM FS USB имеется четыре светодиода: D1, D2, D3, D4. Светодиод D1 моргает при работе USB. Оставшиеся три диода можно задействовать в своих целях.

Для начала просто включим какой-нибудь из диодов. Пусть это будет D4. Чтобы диод горел, нужно применить функцию

LED_On(LED_D4);

однако, из описания функции мы узнаём, что предварительно светодиод должен быть настроен функцией LED_Configure() хотя в дальнейшем тексте мы встречаем только функцию LED_Enable, конечно же она имелась ввиду.

Помещаем, для начала, эти две функции в файле main.c после строки настройки системы:

SYSTEM_Initialize(SYSTEM_STATE_USB_START);

LED_Enable (LED_D4);

LED_On(LED_D4);

Загрузим программу в микроконтроллер. Если светодиод загорелся и USB работает, значит программа и аппаратура исправны. Аналогично можно проверить диоды D3 и D2. Если проверка прошла успешно, уместно будет спрятать настройку светодиодов внутрь функции SYSTEM_Initialize(). Для этого из файла main.c удалим настройку и включение диода, затем наведём курсор на SYSTEM_Initialize(), щёлкнем правой кнопкой мыши и выберем Navigate> Goto Declaration/Definition (рис. 15).

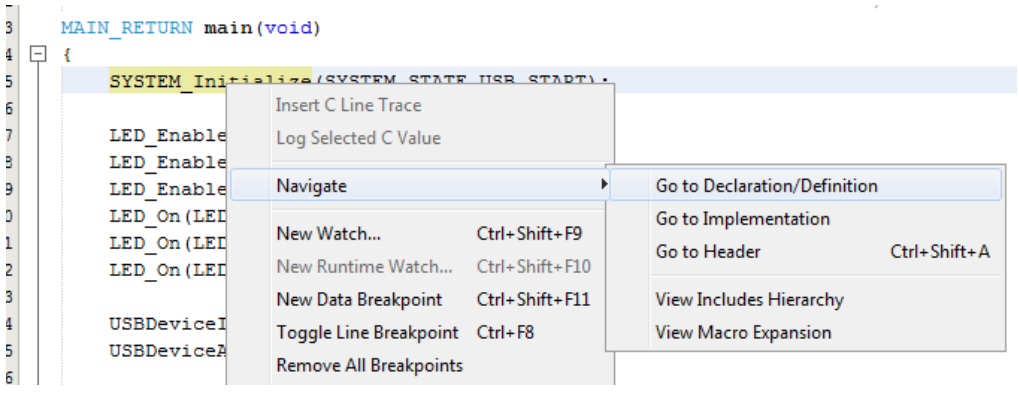

Рис. 15. Переход к файлу с текстом функции.

После этого откроется файл system.c в окне редактора, курсор будет установлен на функции SYSTEM_Initialize. Внутри конструкции switch в ветке SYSTEM_STATE_USB_START помещаем настройку наших диодов и получаем такой код:

```
void SYSTEM_Initialize( SYSTEM_STATE state )
{
   switch(state)
   {
      case SYSTEM_STATE_USB_START:
         LED_Enable(LED_USB_DEVICE_STATE);
         BUTTON_Enable(BUTTON_DEVICE_CDC_BASIC_DEMO);
         LED_Off(LED_D4);
         LED_Off(LED_D3);
         LED_Off(LED_D2);
         LED_Enable (LED_D4);
         LED_Enable (LED_D3);
         LED_Enable (LED_D2);
         break;

      case SYSTEM_STATE_USB_SUSPEND:
         break;

      case SYSTEM_STATE_USB_RESUME:
         break;
   }
}
```

Теперь сделаем управление диодами, для чего будем анализировать массив данных, которые приходят из USB. Откроем main.c и наведём курсор на функцию APP_DeviceCDCBasicDemoTasks(), опять щёлкнем правой кнопкой мыши и выберем Navigate > Goto Declaration/Definition (рис. 15), и добавим после строки:

```
if(numBytesRead > 0)
{
/* After processing all of the received data, we need to send out
* the "echo" data now.
*/
```

строки, которые будут оперировать со светодиодами:

```
switch (readBuffer[0])
{
case '1':LED_On(LED_D4);LED_Off(LED_D2);LED_Off(LED_D3);break;
case '2':LED_On(LED_D3);LED_Off(LED_D2);LED_Off(LED_D4);break;
case '3':LED_On(LED_D2);LED_Off(LED_D3);LED_Off(LED_D4);break;
default:LED_Off(LED_D4);LED_Off(LED_D3);LED_Off(LED_D2);break;
}
```

В этих строках включаются/отключаются светодиоды в зависимости от первого пришедшего по USB символа (ASCII – кода). Откомпилируем программу, загрузим её в микроконтроллер и убедимся в работоспособности – запустим программу dynamic_cdc_demo.exe, передадим символ и проверим

реакцию платы. Также убедимся, что есть отклик на нажатие кнопки S2 на плате.

Однако же, при попытке заменить включение/отключение диодов платы на какие-то другие функции могут возникнуть трудности. Поэтому предпочтительнее будет создать переменную состояния системы, которую менять в зависимости от приходящих символов, а уже в зависимости от переменной запускать те или иные функции.

Глава 3. Управление платой. Переменная состояния.

Из трёх диодов, которые могут находится в двух состояниях (горит/не горит) получается восемь комбинаций, чтобы задействовать все восемь нужно составить таблицу истинности. Упростим себе задачу, допустим так: получаем число 1 (ASCII код) – загорается диод D4, остальные не горят, получаем число 2 – загорается диод D3, остальные не горят, получаем число - 3 – горит диод D2, остальные не горят. Получаем любое другое число (числа) – все диоды гаснут.

Для решения поставленной задачи зададим переменную состояний. В главном цикле будем постоянно её опрашивать и по значению включать или отключать диоды. Всего, по условию задачи, будет четыре состояния:

нулевое – сразу после начала работы, позже, при получении любого символа, кроме 1, или 2, или 3 все диоды отключены.

первое – при получении символа 1 горит только D4,

второе – при получении символа 2 горит только D3,

третье – при получении символа 3 горит только D2.

Поскольку состояний немного, то размер переменной выбираем volatile unsigned char. Префикс volatile нужен, чтобы переменная была всегда доступна из любой части программы, чтобы компилятор в результате оптимизации не внёс беспорядка в нашу программу. Поскольку это глобальная переменная и она определяет работу всей системы, а мы заранее не знаем каким образом нам надо будет изменять состояние системы при модификации программы, в будущем возможно в прерывании. Назовём переменную v_uc_SysState ([4], [5] п. 3.2.2 стр. 28). В файле main.c после строчек #include… добавим строки:

```
/**VARIABLES*****************************************/
volatile unsigned char v_uc_SysState = 0;
```

Теперь в файле app_device_cdc_bacic.c в секции VARIABLES добавим строку:

```
extern unsigned char v_uc_SysState;
```

чтобы функция APP_DeviceCDCBasicDemoTasks() «знала» о существовании такой внешней переменной. Внутри функции будем менять состояние этой переменной. Для таких манипуляций необходимо передавать внутрь функции адрес переменной состояния системы.

Снабдим функцию APP_DeviceCDCBasicDemoTasks() такой возможностью:

добавим в секции VARIABLES в файле main.c указатель на переменную состояния системы:

volatile unsigned char *us_SysState = &v_uc_SysState;

В файле app_device_cdc_bacic.c функцию перепишем таким образом:

void APP_DeviceCDCBasicDemoTasks(volatile unsigned char *SysState)

не забудем в файле app_device_cdc_bacic.h, подправить объявление функции – в скобках вместо void нужно вставить

volatile unsigned char *SysState

Саму функцию APP_DeviceCDCBasicDemoTasks переделываем. После строки

```
if(numBytesRead > 0)
{
/* After processing all of the received data, we need to send out
* the "echo" data now.
*/
```

вставляем такой текст:

```
if (numBytesRead < 2)
   {
      switch (readBuffer[0])
      {
      case '1': *SysState = 1; break;
      case '2': *SysState = 2; break;
      case '3': *SysState = 3; break;
      default:  *SysState = 0; break;
      }
   }
   else *SysState = 0;
```

строка if(numBytesRead > 0) определяет, что символы приняты.

Строка

if (numBytesRead < 2) совместно с else *SysState = 0;

отсекает группы символов. Таким образом фрагмент:

```
      switch (readBuffer[0])
      {
      case '1': *SysState = 1; break;
      case '2': *SysState = 2; break;
      case '3': *SysState = 3; break;
      default:  *SysState = 0; break;
      }
```

работает только при поступлении одиночных символов. При этом все возможные комбинации символов разделяются на четыре случая:

получен символ (ASCII – код) 1, символ 2, символ 3 и все остальные.

При этом в адрес, по которому находится переменная состояния системы записываются обычные числа 0, 1, 2, 3. То есть, переменная состояний системы изменяется в зависимости от полученных по USB ASCII – кодов. После этого нужно менять состояние диодов в зависимости от переменной состояния. Изменяем текст файла main.c следующим образом – после строки:

```
//Application specific tasks
```

изменяем функцию APP_DeviceCDCBasicDemoTasks() добавив в скобки переменную состояния, тем самым передавая параметр us_SysState:

```
APP_DeviceCDCBasicDemoTasks(us_SysState);
```

А сразу после вызова функции в том же main.c добавляем текст:

```
switch (v_uc_SysState)
{
case 0:LED_Off(LED_D4);LED_Off(LED_D3);LED_Off(LED_D2);break;
case 1:LED_On(LED_D4);LED_Off(LED_D2);LED_Off(LED_D3);break;
case 2:LED_On(LED_D3);LED_Off(LED_D2);LED_Off(LED_D4);break;
case 3:LED_On(LED_D2);LED_Off(LED_D3);LED_Off(LED_D4);break;
default:LED_Off(LED_D4);LED_Off(LED_D3);LED_Off(LED_D2);break;
}
```

В этом фрагменте анализируем переменную состояния и реагируем на её изменения.

Компилируем программу, программируем микроконтроллер. После этого всё должно заработать согласно нашему заданию. Вместо диодов можно запускать подпрограммы, которые будут, например, опрашивать датчики и передавать данные на USB.

Глава 4. Расчёт секундного интервала для TIMER0.

Рассчитаем секундный интервал нулевого таймера для прерываний.

Откроем файл system.c. Здесь в секции CONFIGURATION Bits заданы следующие биты конфигурации:

#pragma config PLLDIV = 5 // (20 MHz crystal on PICDEM FS USB board)

#pragma config CPUDIV = OSC1_PLL2

Эти биты конфигурации определяют режим работы тактового генератора микроконтроллера. В документе компилятора языка СИ [2], который расположен в папке компилятора (обычно C:\Program Files\Microchip\xc8\v1.36\docs) в разделе 5.3.5 Configuration Bit Access имеется указание на файл pic18_chipinfo.html, который расположен в той же папке, что и MPLAB XC8 C Compiler User Guide. Откроем файл pic18_chipinfo.html, выберем наш микроконтроллер и посмотрим значение битов конфигурации:

CPUDIV – System Clock Postscaler Selection bits

OSC1_PLL2 [Primary Oscillator Src: /1][96 MHz PLL Src: /2]

В [3] в разделе 2.0 OSCILLATOR CONFIGURATIONS на рис. 2-2 изображены цепи тактового генератора. При указанных выше битах конфигурации CPU микроконтроллера тактируется следующим образом: от кварцевого резонатора поступает частота 20 МГц. Эта частота делится на 5 в предделителе PLL (ФАПЧ), получается 4 МГц. После умножения в блоке PLL до 96 МГц происходит деление на 2 в постделителе PLL. Так на вход CPU подаётся 48 МГц, а поскольку каждый машинный цикл длится четыре такта генератора, то есть возможность подать на вход нулевого таймера сигнал Fosc/4 = 12 МГц.

Для настройки нулевого таймера обратимся к [3] раздел 11.0 TIMER0 MODULE. В управляющем регистре T0CON:

Бит 7 (TMR0ON) установим – это включит нулевой таймер.

Бит 6 (T08BIT) сбросим, это настроит таймер как 16-ти разрядный.

Бит 5 (T0CS) сбросим, выбрав внутренний источник тактового сигнала.

Бит 4 (T0SE) установим, после чего увеличение счётчика будет происходить по нарастанию тактовой частоты.

Бит 3 (PSA) сбросим – это подключит предделитель нулевого таймера.

Биты 2-0 (T0PS2:T0PS0) установим в положение 111, значит тактовая частота поделится на 256.

Получаем такую строку в тексте настройки нулевого таймера:

T0CON = 0b10010111;

Рассчитаем секундный интервал для нулевого таймера:

Частота, поступающая на вход таймера, известна – это Fosc/4 = 12MHz. Поделив на 256 получаем 46,875 КГц. Значит, за секунду будет происходить 46875 колебаний на входе нулевого таймера. Таймер настроен на 16 разрядов – то есть максимальное число до переполнения 65536 (десятичное). Выходит, для секунды нужно записать в регистры Timer0 такое число:

65536 – 46875 = 18661 или 0x48E5.

Это значение запишем в таймер при инициализации. И каждый раз при вызове прерывания будем сбрасывать флаг нулевого таймера, и записывать 0x48E5 в регистры таймера:

```
INTCONbits.TMR0IF = 0;
TMR0L = 0xE5;
TMR0H = 0x48;
```

Для настройки прерываний от таймера установим в регистре INTCON бит 6 – GIEL. Если этот бит установлен (GIEL = 1), то разрешены все периферийные прерывания с низким приоритетом (если бит GIEH = 1 и бит IPEN = 1).

После этого установим низкий приоритет для прерывания от нулевого таймера – в регистре INTCON2 сбросим бит TMR0IP.

В регистре INTCON установим бит TMR0IE, что разрешит прерывания от нулевого таймера.

Чтобы каким-то образом увидеть, что таймер работает, будем оперировать с диодом D4.

Получается, что для настройки прерываний от модуля необходима следующая группа команд:

```
INTCONbits.GIEL = 1;// Enable all Low Priority Interrupt
INTCON2bits.TMR0IP = 0; // 0 – Low/1 -High Priority for TIMER0
INTCONbits.TMR0IE = 1; // Enable timer0 interrupts
```

В итоге получаем такой текст первоначальной настройки для нулевого таймера:

```
void Tmr0_Init (void)
{
INTCONbits.GIEL = 1; // Enable all Low Priority Interrupt
INTCON2bits.TMR0IP = 0; // 0 – Low/1 -High Priority for TIMER0
TMR0L = 0xE5;
TMR0H = 0x48;
T0CON = 0b10010111; // Set timer 0
INTCONbits.TMR0IF = 0;
INTCONbits.TMR0IE = 1; // Enable timer0 interrupts
}
```

Текст функции Tmr0_Init вставим в файл system.c, а в файл system.h вставим объявление этой функции:

```
void Tmr0_Init (void);
```

В файле main.c добавим вызов функции настройки таймера:

```
Tmr0_Init ();
```

перед основным циклом while (1){}

Убедимся, что в функции настройки SYSTEM_Initialize, которая вызывается сразу после запуска программы, есть вызов функции LED_Enable(LED_D4). Функция SYSTEM_Initialize описана в файле system.c.

Откомпилируем текст, убедимся в отсутствии ошибок. Теперь у нас настроен нулевой таймер для работы с интервалом в одну секунду. Перейдём к следующей главе статьи чтобы запустить прерывания от нулевого таймера.

Глава 5. Настройка прерывания от TIMER0.

В микроконтроллерах семейства PIC18 реализовано два отдельных вектора прерываний и простая схема приоритетов прерываний. В исходном тексте программы на языке XC8 функция прерывания должна быть написана с квалификатором interrupt. Функция прерывания должна иметь тип void interrupt и не может иметь параметров. Если в исходном тексте программы попадается ключевое слово interrupt, компилятор располагает такую функцию по адресу прерывания и обеспечивает сохранение и восстановление всех необходимых регистров.

По умолчанию в PIC18 функция прерывания имеет высокий приоритет. Чтобы создать функцию прерывания с низким приоритетом (low-priority interrupt function) после квалификатора interrupt добавляют low_priority ([1], [2] – раздел 5.9.1 Writing an Interrupt Service Routine).

Но как понять используются ли прерывания в стеке MLA v2015_08_10 для работы модуля USB? А если используются, то какой приоритет? Для ответа на этот вопрос откроем в среде разработки файл usb_config.h нажмём Ctrl+F, в открывшейся строке поиска наберём USB_INTERRUPT, и найдём нужную строку в файле (рис. 16):

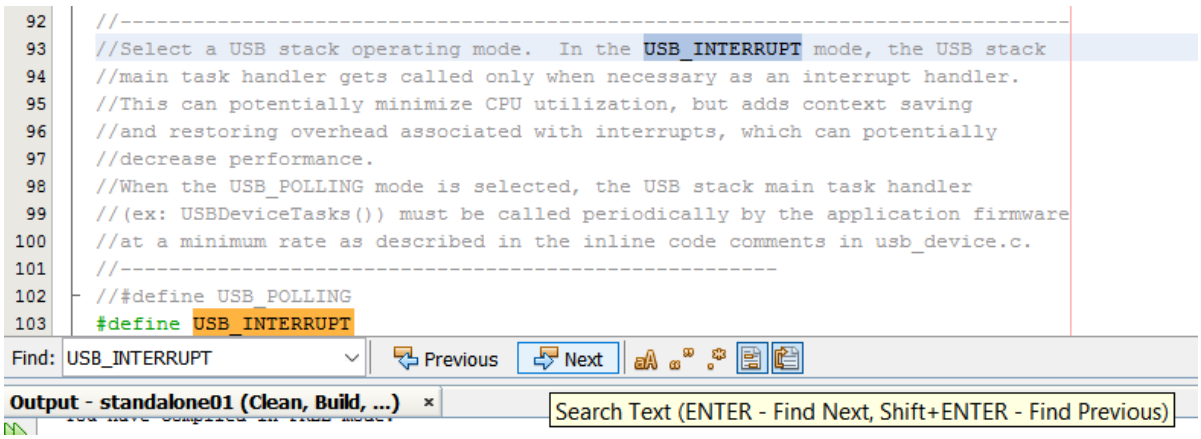

Рис. 16. Поиск строки в файле.

В комментариях к найденной строке обнаруживаем: Выбор режима работы стека USB. Строка #define USB_INTERRUPT включает прерывания для USB.

Каковы же настройки, если включен режим работы с прерываниями? Откроем файл usb_hal_pic18. Найдём фразу:

This section is for all other PIC18 USB microcontrollers

Микроконтроллер PIC18F4550 попадает в раздел «other PIC18 USB microcontrollers». Раздел посвящен обработке прерываний от модуля USB, в нем есть такие строки:

#if defined (USB_INTERRUPT)

#define USBEnableInterrupts() {RCONbits.IPEN = 1; IPR2bits.USBIP = 1; PIE2bits.USBIE = 1; INTCONbits.GIEH = 1;}

Обратимся к документу на микроконтроллер [3] в разделе прерывания (9.0 INTERRUPTS) можно прочесть следующее:

Приоритет прерываний разрешает бит IPEN в регистре RCON (бит 7). Если IPEN = 1, то приоритет прерываний разрешён. В таком случае глобальные прерывания разрешают двумя битами – GIEH и GIEL. Если бит GIEH в регистре INTCON (бит 7) установлен, то все прерывания с высоким приоритетом разрешены (то есть те прерывания для которых бит high priority установлен). Если бит GIEL в регистре INTCON (бит 6) установлен, то все прерывания с низким приоритетом (low priority) разрешены, то есть все прерывания для которых бит high priority сброшен. Если бит разрешения и соответствующий бит глобального прерывания установлены, то прерывание вызовет переход на адрес 000008h или 000018h, в зависимости от того, какой приоритет установлен битом приоритета. Индивидуальные прерывания могут быть запрещены соответствующими битами.

В нашем случае:

Бит IPEN в регистре RCON установлен в единицу. Это значит, что разрешены приоритеты прерываний. Повторюсь, также в соответствии с [3] – раздел 4.1 RCON Register бит 7 IPEN (Interrupt Priority Enable bit) – бит разрешения приоритета прерываний. Если этот бит = 1, то уровни приоритета прерываний разрешены. Если IPEN = 0, то уровни приоритета прерываний запрещены (режим совместимости с PIC16CXXX), это важно.

Бит USBIP в регистре IPR2 установлен, значит прерывание от модуля USB имеет высокий приоритет.

Бит USBIE в регистре PIE2 установлен, что разрешает прерывание от USB модуля.

Бит GIEH в регистре INTCON установлен, это разрешает, в случае если IPEN = 1, все прерывания с высоким приоритетом.

То есть, настройки такие: прерывания разрешены, приоритеты прерываний разрешены, прерывания от модуля USB разрешены и имеют высокий приоритет.

Как же задействован механизм прерываний?

В файле usb_hal_pic18.h разрешается использование прерываний от модуля USB. Строка

#define USBEnableInterrupts() {RCONbits.IPEN = 1; IPR2bits.USBIP = 1; PIE2bits.USBIE = 1; INTCONbits.GIEH = 1;}

Задаёт макрос настройки прерываний. Далее в файле usb_device.c имеется описание функции USBDeviceAttach, внутри которой вызывается указанный выше макрос. Затем в функции main.c в строках инициализации микроконтроллера вызывается USBDeviceAttach(); Значит, к моменту входа в главный цикл while(1){} у микроконтроллера уже разрешены прерывания, и разрешены приоритеты прерываний.

Откроем файл system.c найдём такие строки:

```
void interrupt SYS_InterruptHigh(void)
{
```

```
    #if defined(USB_INTERRUPT)
      USBDeviceTasks();
    #endif
}
```

подправим не совсем корректный вызов прерывания из библиотеки MLA; правку производим согласно [1], [2] – раздел 5.9.1 Writing an Interrupt Service Routine и [3] – раздел 9.1 USB Interrupts и раздел 17.5 USB Interrupts. Напишем следующий код:

```
void interrupt SYS_InterruptHigh(void)
{
   if (USBIF && USBIE)
   {
   USBDeviceTasks();
   }
}
```

Откомпилируем. В случае, если загружаем через bootloader, не забываем установить смещение (рис. 12, 13) Code offset 0x1000, с которого будет начинаться программа. А затем адреса памяти, которые не будут заняты программой (рис. 14) – ROM ranges: default,-0-FFF,-1006-1007,-1016-1017. После компиляции загрузим прошивку в микроконтроллер и убедимся, что программа работает.

Откорректируем прерывание с высоким уровнем приоритета для работы с нулевым таймером. Для этого проделаем следующие операции:

В файле main.c перед главным циклом, то есть перед строкой while(1){} добавим (для демонстрационных целей) настройку высокого уровня приоритета прерываний от нулевого таймера:

```
INTCON2bits.TMR0IP = 1; // 0 – Low/1 -High Priority for TIMER0
```

Откроем файл system.c и добавим в текст имеющегося прерывания с высоким уровнем приоритета обработку прерывания от нулевого таймера:

```
void interrupt SYS_InterruptHigh(void)
{
   if (USBIF && USBIE)
   {
      USBDeviceTasks();
   }
   else if(TMR0IF && TMR0IE)
   {
      INTCONbits.TMR0IF = 0;
      TMR0L = 0xE5;
```

```
        TMR0H = 0x48;
        LED_Toggle(LED_D4);
    }
}
```

Теперь добавим манипуляции с диодом D4 в переменную состояния: состояние 1 не будет включать или выключать D4; этот диод будет включаться и отключаться нулевым таймером. Для этого в файле main.c переделаем нашу конструкцию switch, убрав в строке case 1: функцию LED_On(LED_D4);

```
switch (v_uc_SysState)
{
case 0:LED_Off(LED_D4);LED_Off(LED_D3);LED_Off(LED_D2);break;
case 1:                LED_Off(LED_D2);LED_Off(LED_D3);break;
case 2:LED_On(LED_D3);LED_Off(LED_D2);LED_Off(LED_D4);break;
case 3:LED_On(LED_D2);LED_Off(LED_D3);LED_Off(LED_D4);break;
default:LED_Off(LED_D4);LED_Off(LED_D3);LED_Off(LED_D2);break;
}
```

После компиляции загрузим прошивку в микроконтроллер и убедимся, что программа работает: данные передаются по USB, а после передачи на микроконтроллер числа 1 диод моргает раз в секунду.

Однако же не стоит вмешиваться в работу модуля USB – оставим этому модулю прерывание с высоким уровнем приоритета. А поскольку нулевой таймер, настроенный на одну секунду и имеющиеся на плате термодатчик и переменный резистор достаточно медленная аппаратура, то прерывание от нулевого таймера настроим с низким уровнем приоритета (см. [3] – раздел 11, таблица 11-1 бит TMR0IP).

В файле main.c перед главным циклом мы настроили высокий уровень приоритета для прерывания от нулевого таймера:

INTCON2bits.TMR0IP = 1; // 0 – Low/1 -High Priority for TIMER0

удалим эту настройку из исходного текста программы.

Напишем прерывание для нулевого таймера с низким уровнем приоритета.

Для этого откроем файл system.c удалим из прерывания с высоким уровнем приоритета всё что связано с нулевым таймером, оставив строки:

```
void interrupt SYS_InterruptHigh(void)
{
    if (USBIF && USBIE)
    {
        USBDeviceTasks();
    }
}
```

Напишем обработчик прерывания с низким уровнем приоритета, который будет запускаться по переполнению нулевого таймера. В функции прерывания будем отслеживать бит TMR0IF в регистре INTCON, заполнять таймер на одну секунду, затем переключать диод D4:

```
void interrupt low_priority Tmr0Isr(void)
{
   if(T0IF && T0IE)
   {
      INTCONbits.T0IF = 0;
      TMR0L = 0xE5;
      TMR0H = 0x48;
      LED_Toggle(LED_D4);
   }
}
```

Вставим этот текст в файл system.c после функции прерывания с высоким уровнем приоритета.

Откомпилируем программу. Перед тем, как прошивать *.hex файл в микроконтроллер откроем файл *.map, который расположен в той же папке, где *.hex и имеет то же самое название. Одна из строк файла *.map будет выглядеть так:

-preset_vec=01000h,intcode=01008h,intcodelo=01018h…

Что говорит нам о следующем: вектор сброса расположен по адресу 0x1000, вектор прерывания с высоким уровнем приоритета расположен по адресу 0x1008, вектор прерывания с низким уровнем приоритета расположен по адресу 0x1018. Это соответствует нашим установкам для XC8 linker и программа будет работать совместно с загрузчиком.

Прошиваем микроконтроллер и убеждаемся в работоспособности программы – должен работать USB (проверяем программой dynamic_cdc_demo), после получения символа ‘1’ раз за секунду будет переключаться светодиод D4.

Рекомендуемая литература.

[1] http://microchip.wikidot.com/faq:31
[2] MPLAB_XC8_C_Compiler_User_Guide.pdf
[3] PIC18F2455/2550/4455/4550 Data Sheet
[4] В. Тимофеев «volatile для «чайников»
[5] В. Тимофеев «Как писать программы без ошибок»
MPLAB_X_IDE_Users_Guide.pdf
PICDEM™ FS USB DEMONSTRATION BOARD USER'S GUIDE
MCHPFSUSB Firmware User`s Guide
MPLAB_XC8_Getting_Started_Guide.pdf
help_mla_usb.pdf

Автор: инженер М. В. Богураев.