

Микроконтроллеры фирмы Microchip. Первые шаги.

ВВЕДЕНИЕ.

Выбор микроконтроллера.

В настоящее время микроконтроллеры (МК) используются практически во всех областях человеческой деятельности. Фирма **Microchip** по праву в течение нескольких лет занимает второе место по производству 8-разрядных МК. Такое положение фирмы обусловлено высокой надежностью и функциональностью выпускаемых микроконтроллеров, конкурентной ценой, доступностью средств отладки, развитой системой технической поддержки разработчика.

Microchip производит широкую номенклатуру микроконтроллеров с числом выводов от 8 до 88, памятью программ от 512 до 64К слов и различной периферией: последовательный интерфейс (SPI), универсальный синхронный/асинхронный приёмопередатчик (USART), поддержка протокола I²C, аналого-цифровые (ADC) и цифро-аналоговые преобразователи (DAC), модули широтно-импульсной модуляции (PWM) и модули сравнения-захвата (CCP) и так далее.

На данный момент среди продукции фирмы **Microchip** можно выделить такие семейства микроконтроллеров, как PIC12Cxxx, PIC16C5x, PIC16Cxxx, PIC17Cxxx, PIC18Cxxx.


Семейство PIC12Cxxx характеризуется производительностью до 2,5 миллиона операций в секунду, системой команд из 33 (35) инструкций, восьмивыводным корпусом, внутренним RC-генератором на 4 МГц, нагрузочной способностью в 25 миллиампер на каждый вывод, наличием одного 8-ми разрядного таймера-счётчика и одного сторожевого таймера.

Для семейства PIC16C5x характерна производительность в 5 миллионов операций в секунду, система команд из 33 инструкций, нагрузочная способность в 20-25 миллиампер на вывод, наличие одного 8-ми разрядного таймера-счётчика и одного сторожевого таймера.


В семействе PIC16Cxxx реализовано от 4 до 12 видов прерываний, производительность в 5 миллионов операций в секунду, система команд из 35 инструкций, совместимость с семействами PIC12Cxxx и PIC16C5x., различные комбинации таймеров счётчиков (8-ми и 16-ти разрядных в разных количествах, кроме того – сторожевые таймеры), в некоторых моделях имеется последовательный интерфейс, универсальный синхронный/асинхронный приёмопередатчик, поддержка протокола I²C, АЦП (8-ми, 10-ти и 12-ти разрядные), модули широтно-импульсной модуляции, компараторы и прочее.

Для семейства PIC17Cxxx характерна производительность в 8,3 миллиона операций в секунду, 58 инструкций в системе команд, операция умножения 8*8 за один машинный цикл, совместимость с предыдущими семействами. Имеется два 16-ти разрядных, два 8-ми разрядных таймера-счётчика, один сторожевой таймер, универсальный синхронный/асинхронный приёмопередатчик, в некоторых моделях – последовательный интерфейс и поддержка протокола MI²C, 10-ти разрядный АЦП. Данное семейство позволяет подключать внешнюю память программ до 64К*16, осуществлять чтение и запись памяти программ.

В семействе PIC18Cxxx производительность возрастает до 10 миллионов операций в секунду, система команд расширяется до 77 инструкций и становится эффективной для

620086, г. Екатеринбург, ул. Чкалова, 3, «**ABERON**» 

(3432) 237-079, факс: (3432) 237-038 

<http://www.averon.ru>; e-mail: ic@averon.ru, support@averon.ru 

использования совместно с компиляторами языка Си, появилась возможность программной организации стека и возможность обработки табличных данных. В PIC18Cxxx появился умножитель частоты, имеется возможность переключения источника тактирования. Семейство совместимо с предыдущими, имеется три 16-ти разрядных, один 8-ми разрядный таймер-счётчик, один сторожевой таймер, кроме того – адресуемый универсальный синхронный/асинхронный приёмопередатчик (AUSART), поддержка протокола MI²C и последовательного интерфейса, в некоторых моделях – параллельный ведомый порт.

Выпускаемые контроллеры можно разделить на два основных типа: однократно (OTP с памятью EPROM) и многократно программируемые (с памятью EEPROM или FLASH).

EPROM (OTP) можно запрограммировать только один раз. Для отладки таких МК выпускаются JW версии кристаллов с окном в корпусе для ультрафиолетового стирания (УФ стирания). JW-контроллеры отличаются от OTP только типом корпуса.

Замечание: в большинстве JW-версий кристаллов нельзя стереть биты защиты, поэтому никогда их не устанавливайте, иначе из многократно перепрограммируемого микроконтроллера Вы получите однократно программируемый, так как установленные биты защиты не позволят ничего записать в память программ.

Контроллеры с FLASH-памятью программ (EEPROM) можно программировать и стирать при помощи программатора, количество перезаписей зависит от типа МК (обычно 1000 и более раз).


Для начального освоения лучше использовать МК с FLASH -памятью, среди которых можно рекомендовать PIC16F628 или МК серии PIC16F87х. Ниже представлена сводная таблица по этим микроконтроллерам, которая даст Вам ориентировочное представление об их характеристиках.

Что касается микроконтроллера PIC16F628, то это расширенный аналог популярного и недорогого МК PIC16F84.

Контроллеры серии PIC16F87х – более дорогие, что обусловлено наличием модуля 10-иразрядного АЦП, увеличенного объёма памяти, возможностью отладки с помощью внутрисхемного отладчика **MPLAB ICD**. Ещё одной отличительной чертой PIC16F87х является возможность этих МК читать и изменять (программировать) свою память программ, что можно использовать для организации обновления программы без использования программатора (например, через UART, RS-232, SPI, INTERNET), для хранения больших массивов данных и тому подобного.

Таблица 1. Сводная таблица по микроконтроллерам.

Контроллер	Память			АЦП, бит/ каналов	Корпус	Дополнительная периферия
	ОЗУ, байт	ПЗУ, слов	EEPROM данных, байт			
PIC16F627	224	1024*14 flash	128	Нет	18DIP, 18SOIC, 20SSOP	Таймеры: 1-16р, 2-8р, WDT, PWRT и OST. 10р PWM, 2 компаратора, USART/SCI, BOR.
PIC16F628		2048*14 flash				
PIC16F73	192	4096*14 flash	Нет	8/5	28DIP, 28SOIC, 28SSOP	Таймеры: 1-16р, 2-8р, WDT, PWRT и OST. 10р PWM, USART/SPI/I ² C, BOR.
PIC16F76	368	8192*14 flash				
PIC16F74	192	4096*14 flash	Нет	8/8	40DIP, 44PLCC, 44TQFP	Таймеры: 1-16р, 2-8р, WDT, PWRT и OST. 10р PWM, USART/SPI/I ² C, BOR.
PIC16F77	368	8192*14 flash				

620086, г. Екатеринбург, ул. Чкалова, 3, «АВЕРОН» 

(3432) 237-079, факс: (3432) 237-038 

<http://www.averon.ru>; e-mail: ic@averon.ru, support@averon.ru 

Таблица 1. Сводная таблица по микроконтроллерам (продолжение).

Контроллер	Память			АЦП, бит/ каналов	Корпус	Дополнительная периферия
	ОЗУ, байт	ПЗУ, слов	EEPROM данных, байт			
PIC16F870	128	2048*14 flash	64	10/5	28DIP, 28SOIC, 28SSOP	Таймеры: 1-16разр, 2-8разр и WDT. 10p PWM, AUSART, CCP, self programming ICD (внутрисхемный отладчик, возможность самопрограммирования).
PIC16F871				10/8	40DIP, 44PLCC, 44TQFP	
PIC16F872	128	2048*14 flash	64	10/5	28DIP, 28SOIC, 28SSOP	
PIC16F873	192	4096*14 flash	128	10/5	28DIP, 28SOIC	Таймеры: 1-16p, 2-8p и WDT. 2 10p PWM, MI^2C , AUSART, 2 CCP, self programming ICD (внутрисхемный отладчик, возможность самопрограммирования).
PIC16F874				10/8	40DIP, 44PLCC, 44TQFP	
PIC16F876	368	8192*14 flash	256	10/5	28DIP, 28SOIC	
PIC16F877				10/8	40DIP, 44PLCC, 44TQFP	

Система команд.

PIC-контроллеры имеют RISC архитектуру с ограниченным набором команд (от 33 до 77 в зависимости от семейства), однако это не является препятствием для написания компактных и эффективных программ. В PIC-ах все команды занимают одно слово (в зависимости от семейства МК – 12, 14, 16 бит) и выполняются за один машинный цикл (4 такта генератора), за исключением команд перехода, которые выполняются за два машинных цикла.

Ниже Вашему вниманию предлагается система команд для микроконтроллеров среднего, так называемого «Mid-Range» семейства (PIC16Cxxx,..., PIC16Fxxx). При описании системы команд принято использовать следующие обозначения:

- для команд, ориентированных на работу с байтами, буква “f” означает файловый регистр, который будет использоваться в команде, а буква “d” означает местоположение, куда будет помещён результат операции. Если вместо “d” стоит нуль, то результат будет помещён в рабочий регистр “w” (так называемый аккумулятор), если же вместо “d” стоит единица, то результат будет помещён в регистр “f”;
- для команд, ориентированных на работу с отдельными битами буква “b” означает номер бита, задействованного в операции, а буква “f” символизирует регистр, где расположен этот бит;

- для команд, связанных с литералами, и для команд управления буква “k” символизирует восьми- или одиннадцатиразрядную константу (адрес метки или перехода), либо значение литерала-константы.

Таблица 2. Система команд микроконтроллеров среднего семейства.

Мнемоника и операнды	Описание команды	Влияние на биты регистра состояния (STATUS)
Команды, ориентированные на работу с байтами		
ADDWF f, d	Сложение содержимого аккумулятора с содержимым регистра “f”	C, DC, Z
ANDWF f, d	Логическое «И» между содержимым аккумулятора и содержимым регистра “f” (содержимое аккумулятора логически умножается на содержимое регистра “f”)	Z
CLRF f	Очистка содержимого регистра “f”	Z
CLRW	Очистка содержимого аккумулятора	Z
COMF f, d	Дополнение регистра “f”	Z
DECF f, d	Декремент содержимого регистра “f”	Z
DECFSZ f, d	Декремент содержимого регистра “f”, пропуск команды, если 0 (если результат не равен нулю, то выполняется следующая команда; если же результат равен нулю, то следующая считанная в текущем командном цикле команда игнорируется, а вместо неё выполняется «пустой» оператор NOP, в результате команда выполняется за два цикла)	
INCF f, d	Инкремент содержимого регистра “f”	Z
INCFSZ f, d	Инкремент содержимого регистра “f”, пропуск команды, если 0 (если результат не равен нулю, то выполняется следующая команда; если же результат равен нулю, то следующая считанная в текущем командном цикле команда игнорируется, а вместо неё выполняется «пустой» оператор NOP, в результате команда выполняется за два цикла)	
IORWF f, d	Логическое «ИЛИ» между содержимым аккумулятора и регистра “f” (содержимое аккумулятора логически складывается с содержимым регистра “f”)	Z
MOVF f, d	Перенос содержимого регистра “f”	Z
MOVWF f	Перенос содержимого аккумулятора в регистр “f”	
NOP	Пустой оператор	
RLF f, d	Сдвиг содержимого регистра “f” на 1 бит влево через бит переноса C	C
RRF f, d	Сдвиг содержимого регистра “f” на 1 бит вправо через бит переноса C	C
SUBWF f, d	Вычитание содержимого аккумулятора из содержимого регистра “f” (в случае отрицательного результата C=0, а в случае нулевого или положительного результата C=1; биты C и DC устанавливаются в 1 в случае отсутствия заёма из старшего разряда)	C, DC, Z
SWAPF f, d	В регистре “f” младшие четыре бита меняются местами со старшими четырьмя битами	
XORWF f, d	Исключающее «ИЛИ» между содержимым аккумулятора и содержимым регистра “f” (содержимое аккумулятора поразрядно складывается с содержимым регистра “f” по модулю 2)	Z
Команды, ориентированные на работу с отдельными битами		
BCF f, b	Сброс бита в регистре “f”	
BSF f, b	Установка бита в регистре “f”	
BTFSC f, b	Проверка бита в регистре “f”, очищенные биты пропускаются	
BTFSS f, b	Проверка бита в регистре “f”, установленные биты пропускаются	

Таблица 2. Система команд микроконтроллеров среднего семейства (продолжение).

Команды управления и команды, ориентированные на работу с литералами		
ADDLW k	Сложение константы с содержимым аккумулятора	C, DC, Z
ANDLW k	Логическое «И» между константой и содержимым аккумулятора (содержимое аккумулятора логически умножается на константу)	Z
CALL k	Вызов подпрограммы	
CLRWDT	Сброс сторожевого таймера	TO, PD
GOTO k	Переход по адресу	
IORLW k	Логическое «ИЛИ» между константой и содержимым аккумулятора (содержимое аккумулятора логически складывается с константой)	Z
MOVLW k	Перенос константы в аккумулятор	
RETFIE	Возврат из прерывания	
RETLW k	Возврат из подпрограммы с загрузкой константы в аккумулятор	
RETURN	Возврат из подпрограммы	
SLEEP	Переход в режим SLEEP	TO, PD
SUBLW k	Содержимое аккумулятора вычитается из константы (в случае отрицательного результата C=0, а в случае нулевого или положительного результата C=1)	C, DC, Z
XORLW k	Исключающее «ИЛИ» между содержимым аккумулятора и константой (содержимое аккумулятора поразрядно складывается с константой)	Z


Написание программ.


При написании программного обеспечения для PIC-контроллеров следует учитывать некоторые их особенности, в частности, особенности структуры и ядра. Ниже приведён ряд советов-примеров, призванных облегчить начальное освоение PIC-контроллеров.

Пример 1. Сохранение контекста при входе в прерывание.

В процессе написании программ необходимо помнить, что во время прерывания может измениться состояние некоторых регистров, поэтому нужно предварительно сохранять информацию о текущем состоянии системы.

```
org      h' 04'                                ; макроассемблерная директива
                                                ; установки адреса начала
                                                ; размещения последующих
                                                ; за этой директивой команд
        movwf    wbuf                          ; сохранение значения рабочего регистра
                                                ; w в буферном регистре wbuf
        swapf    STATUS, 0                    ; копирование регистра состояния
                                                ; в рабочий регистр с перестановкой
                                                ; тетрад
        bcf      STATUS, RP1                  ; комбинация из сброшенных битов RP1
        bcf      STATUS, RP0                  ; и RP0 соответствует переходу в
                                                ; банк данных BANK0
        movwf    sbuf                          ; сохранение значения регистра состояния
                                                ; в буферном регистре sbuf
        movfw    FSR                          ; копирование значения регистра косвенной
```

620086, г. Екатеринбург, ул. Чкалова, 3, «АВЕРОН» 

(3432) 237-079, факс: (3432) 237-038 

<http://www.averon.ru>; e-mail: ic@averon.ru, support@averon.ru 

; адресации

```

movwf    FSR_Buff    ; сохранение значения регистра FSR
                        ; в буферном регистре FSR_Buff
/* . . . . . */      ; функция обработки прерываний
EndInt    movfw       FSR_Buff    ; копирование содержимого буфера
                        ; FSR_Buff в рабочий регистр w
movwf    FSR          ; восстановление значения регистра
                        ; косвенной адресации
swapf    sbuf, 0      ; копирование содержимого буфера
                        ; sbuf в регистр w с перестановкой тетрад
movwf    STATUS       ; восстановление значения
                        ; регистра состояния
swapf    wbuf, 1      ; восстановление значения
swapf    wbuf, 0      ; рабочего регистра w
retfie                    ; выход из прерывания

```

Пример 2. Организация памяти.


Следует обращать внимание на организацию памяти микроконтроллеров. Память программ разбивается на страницы по 2 Килослова. Поэтому при вызове подпрограмм и при переходах на метки, расположенные в другой странице памяти необходимо предварительно модифицировать регистр PCLATH.

Перед каждой командой типа «call» или «goto» трогать PCLATH не обязательно, а вот перед вызовом табличного чтения об этом нужно позаботиться. Таблицы констант можно размещать в любом участке памяти, лишь бы они не пересекали границу в 0xNNff.

```

org      h'0000'      ; страница 0
goto     START        ; PCN берется из команды
org      h'0100'
START
movlw    HIGH Tabl
movwf    PCLATH        ; загружаем в PCLATH адрес для корректного
                        ; вычисления перехода в таблице
movfw    Pointer      ; загружаем указатель на смещение по таблице
call     Tabl          ; в стеке сохраняется текущее значение адреса,
                        ; переход произойдет правильно и без модификации PCLATH
movwf    Tabl_ptr
movlw    HIGH $
movwf    PCLATH        ; восстанавливаем PCLATH (хотя в данном
                        ; примере для данных контроллеров это не обязательно - goto и call в пределах
                        ; 2K работают без предустановки PCLATH)
goto     START
org      h'0200'
Tabl                                ; без правильного значения в PCLATH, переход, при
                        ; модификации PCL, произошел бы по адресам = (h'0100' + W)
addwf    PCL, f

```

620086, г. Екатеринбург, ул. Чкалова, 3, «АВЕРОН» (3432) 237-079, факс: (3432) 237-038 <http://www.averon.ru>; e-mail: ic@averon.ru, support@averon.ru 

```

    retlw    0           ; Начало таблицы
    retlw    1
    ...
    retlw    10          ; Конец таблицы
    org      h'0800'      ; страница 1
    movlw    HIGH START
    movwf    PCLATH       ; загружаем в PCLATH адрес для перехода на
    ; страницу 0.
    goto     START
end

```

При использовании в программе прерываний и табличного чтения необходимо на время работы с таблицей запретить прерывания.

Пример 3. Организация краткосрочных временных задержек.

Память программ, несмотря на свои размеры, конечна, поэтому при программировании сложных устройств необходимо создавать компактный код.

Предлагается использовать текущий адрес, который имеет мнемоническое обозначение в виде \$. Тогда команда «GOTO \$+1» будет означать переход на следующий оператор. А так как длительность выполнения команды «GOTO» составляет два такта, то этот оператор будет эквивалентен двум пустым операторам «NOP», таким образом, задержку на два такта можно получить, используя одну команду вместо двух. Выигрыш в командах будет тем заметнее, чем большую задержку надо получить.

Пример 4. Сложение двух операндов с пересылкой результата по адресу первого операнда.

Требуется осуществить операцию: $A+B \rightarrow A$.

Стандартный подход (требуется 3 команды):

```

movf  F, W
addwf B, W
wovwf A

```

Рациональный подход (требуется 2 команды):

```

movf  B, W
addwf A, 1

```

Пример 5. Пересылка единичного бита из регистра REG1 в регистр REG2.

Стандартный подход (требуется 4 команды):

```

btfss REG1, 2
bcf    REG2, 5
btfsc REG1, 2
bsf    REG2, 5

```

Рациональный подход (требуется 3 команды):

```

bcf    REG2, 5
btfsc REG1, 2
bsf    REG2, 5

```

Пример 6. Экономичная организация сравнения содержимого регистров.

```

movf  REG2, W
xorwf REG1, W      ; сравнение регистра REG1 с регистром REG2

```

btfsz STATUS, Z ; пропустить следующую команду, если регистры равны
goto NOTEQUAL ; переход на метку начала обработки случая «не равны»
EQUAL ; метка начала обработки случая «равны»
NOTEQUAL ; метка начала обработки случая «не равны»

Средства отладки.

Среди многочисленных средств отладки программ для микроконтроллеров в первую очередь следует отметить интегрированную среду разработки **MPLAB**, созданную фирмой **Microchip** специально для разработки проектов на базе PIC-микроконтроллеров. Эта среда позволяет писать и отлаживать программы в различных режимах. В состав этой среды входят такие модули, как менеджер проектов, текстовый редактор, симулятор, причём всё это поставляется производителем бесплатно, **MPLAB** может работать с компиляторами языка Си фирм **Microchip**, **HiTech**, **C2C**, **Byte Craft**, **CCS**. Среда поддерживает работу с различными средствами отладки: с внутрисхемным эмулятором **MPLAB ICE 2000**, внутрисхемным отладчиком **MPLAB ICD**, с программаторами **PRO MATE** и **PICSTART PLUS**. Одним словом – это очень мощный программный пакет, способный сделать Вашу работу комфортной.

Несколько слов о внутрисхемном отладчике.

Для отладки программы непосредственно в изделии можно применить внутрисхемный отладчик – **MPLAB ICD**, который поддерживается средой разработки **MPLAB**. **ICD** может программировать и работать с контроллерами серии PIC16F87х. По сравнению с программным симулятором, с помощью **ICD** можно загрузить исполняемую программу непосредственно в МК отлаживаемого устройства, запустить её и просмотреть результаты работы на компьютере. С помощью **ICD** можно просматривать и изменять содержимое регистров МК, выполнять пошаговую отладку программы внутри МК, устанавливать точки останова, а также выполнять программу в режиме реального времени. **MPLAB ICD** будет незаменим при отладке аппаратно-зависимых участков кода, которые трудно воспроизвести в симуляторе, например, при работе с АЦП, при измерении временных параметров входных сигналов, при организации обратной связи с управляемым объектом и так далее.

С помощью отладчика можно программировать микроконтроллеры непосредственно в устройстве, что бывает необходимо при использовании МК в корпусах TQFP и им подобных.

Ниже внутрисхемному отладчику посвящён отдельный раздел, где Вы можете подробно ознакомиться с особенностями его работы.

В фирме «Аверон» Вы можете приобрести аналог внутрисхемного отладчика **MPLAB ICD** всего за 40\$

Макет.

Для начального освоения или для использования в готовых устройствах фирма «Аверон» выпускает процессорные модули на основе PIC16F628 и PIC16F87х. Эти модули можно соединить с модулями измерения, индикации и управления из состава так называемого «Электронного конструктора».

РАБОТА СО СРЕДОЙ MPLAB.

Языки Высокого Уровня.

Самым популярным языком программирования высокого уровня является язык Си. А среди самых популярных компиляторов языка Си можно выделить компилятор фирмы **Hi Tech**.

Подключение к MPLAB компилятора Hi-Tech C.

После установки компилятора Hi-Tech C следует «объяснить» среде **MPLAB**, где ей искать компилятор. Для этого:

1. В **MPLAB** следует выбрать меню «Project» («Проект»), затем «Install Language Tool» («Установка языка программирования»). Должно появиться следующее диалоговое окно.

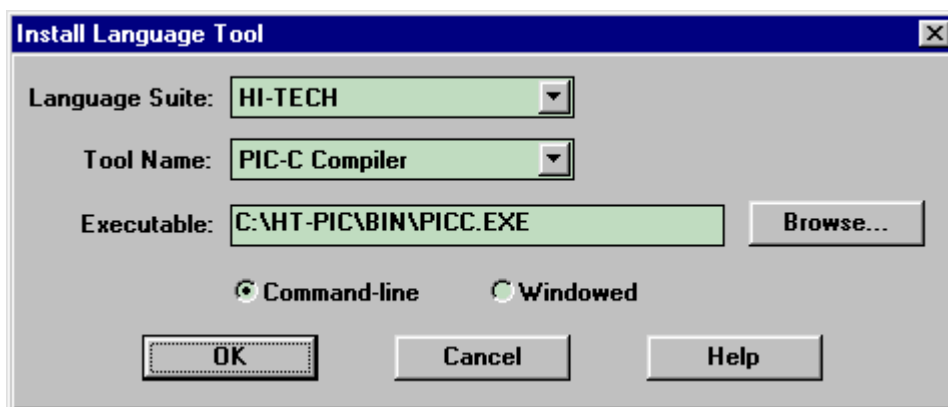


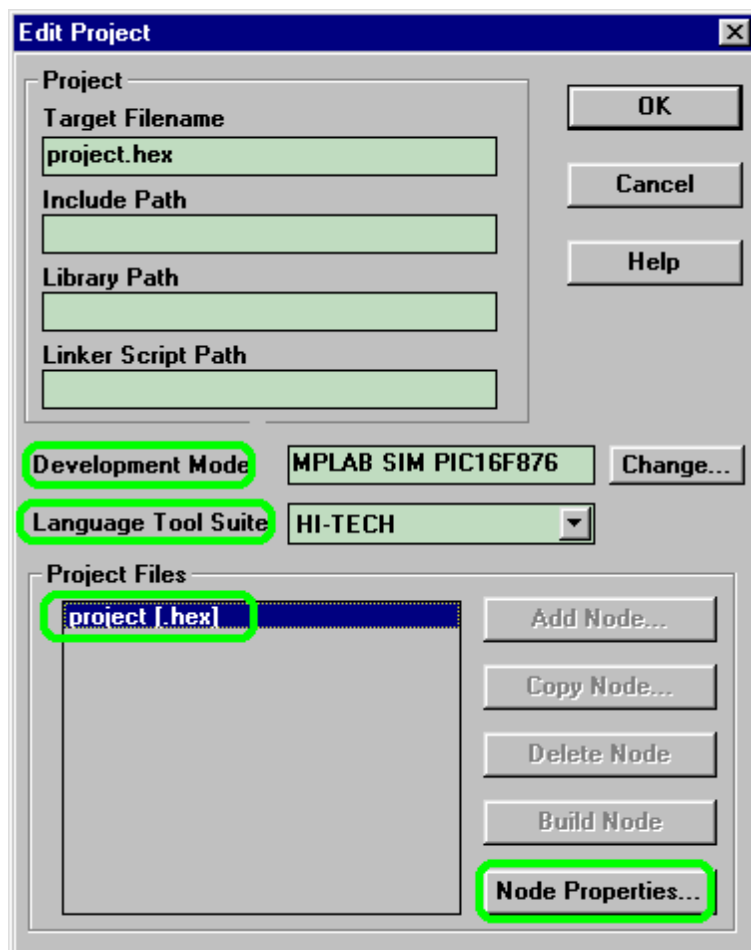
Рис. 1 Окно установки языка программирования

2. Выберите название языка «Hi-Tech», а название инструмента – «PIC-C compiler».
3. Укажите путь к директории «C:\HT-PIC\BIN\» и найдите исполняемый файл PICC.EXE.
4. Измените название инструмента на «PIC-C linker» и повторите всю процедуру вновь, то же самое следует сделать и для ассемблерного модуля.

Примечание: Linker (установщик связей), Compiler (компилятор) и Assembler (Ассемблер) должны быть подключены в обязательном порядке; этим программным модулям назначен файл PICC.EXE, в случае пробной версии Hi-Tech C следует указать файл PICL.EXE. PICC.EXE -единый запускающий файл и для компилятора, и для ассемблера, и для линкера (установщика связей) и автоматически запускают именно тот модуль, который требуется в данный момент, ориентируясь на расширение обрабатываемого файла. Таким образом, Вам не нужно непосредственно вызывать исполняемый файл нужного модуля – всё делается автоматически.

Создаём Ваш первый проект для Hi-Tech C.

Эта инструкция предназначена для контроллеров серий PIC12xxx, PIC16xxx и PIC17xxx. Она не подходит для версии PIC18xxx. Вы можете обратиться по адресу <http://www.htsoft.com/> и там узнать, как наладить взаимодействие такого компилятора со средой MPLAB.



Примечание: рекомендуется следовать описанной здесь методике; хотя на данном этапе это и не важно, но впоследствии, выбирая «PIC C Compiler» в меню «node properties» («свойства узлов») для «project[.hex]», вы можете столкнуться с невозможностью получения составных файлов, объектных файлов или библиотек, поэтому лучше следовать предлагаемой методике, поскольку она более правильная.


Рассмотрим создание и отладку программы с использованием процессорной платы «МС6» на базе PIC16F876xxx, платы индикации «EKIxxx» и MPLAB ICD («MP3») производства фирмы «Аверон».

1. В среде MPLAB выберите меню «Project» («Проект»), затем – «New Project» («Новый проект»). Пользуясь программой Explorer

Рис.2 Окно редактирования проекта

(«Проводник» в среде Windows) создайте на жёстком диске папку для своего будущего проекта. Вернувшись в MPLAB, укажите путь к созданной Вами папке.

2. В графе «File Name» («Имя Файла») наберите название проекта, например «project.pjt», затем нажмите «Ok». Это вызовет появление окна редактирования проекта, такого же, как на Рис. 2.
3. Выберите следующий режим разработки (development mode): «MPLAB SIM 16F876» (см. рисунок). Не обращайте внимания на предупреждение: «No hex file has been built for this project» («Для этого проекта не создан шестнадцатеричный [hex] файл с кодом»).
4. Язык программирования следует изменить на «Hi-Tech», как показано на рисунке. Не обращайте внимания на предупреждение: «You will lose all command line options» («Все опции командной строки будут утеряны»).
5. Выделите «project [.hex]», затем нажмите на кнопку «node properties».

620086, г. Екатеринбург, ул. Чкалова, 3, «АВЕРОН» 

(3432) 237-079, факс: (3432) 237-038 

<http://www.averon.ru>; e-mail: ic@averon.ru, support@averon.ru 

6. Вы увидите окно. Выберите в качестве инструмента PIC C Linker (установщик связей) и задайте соответствующие настройки.

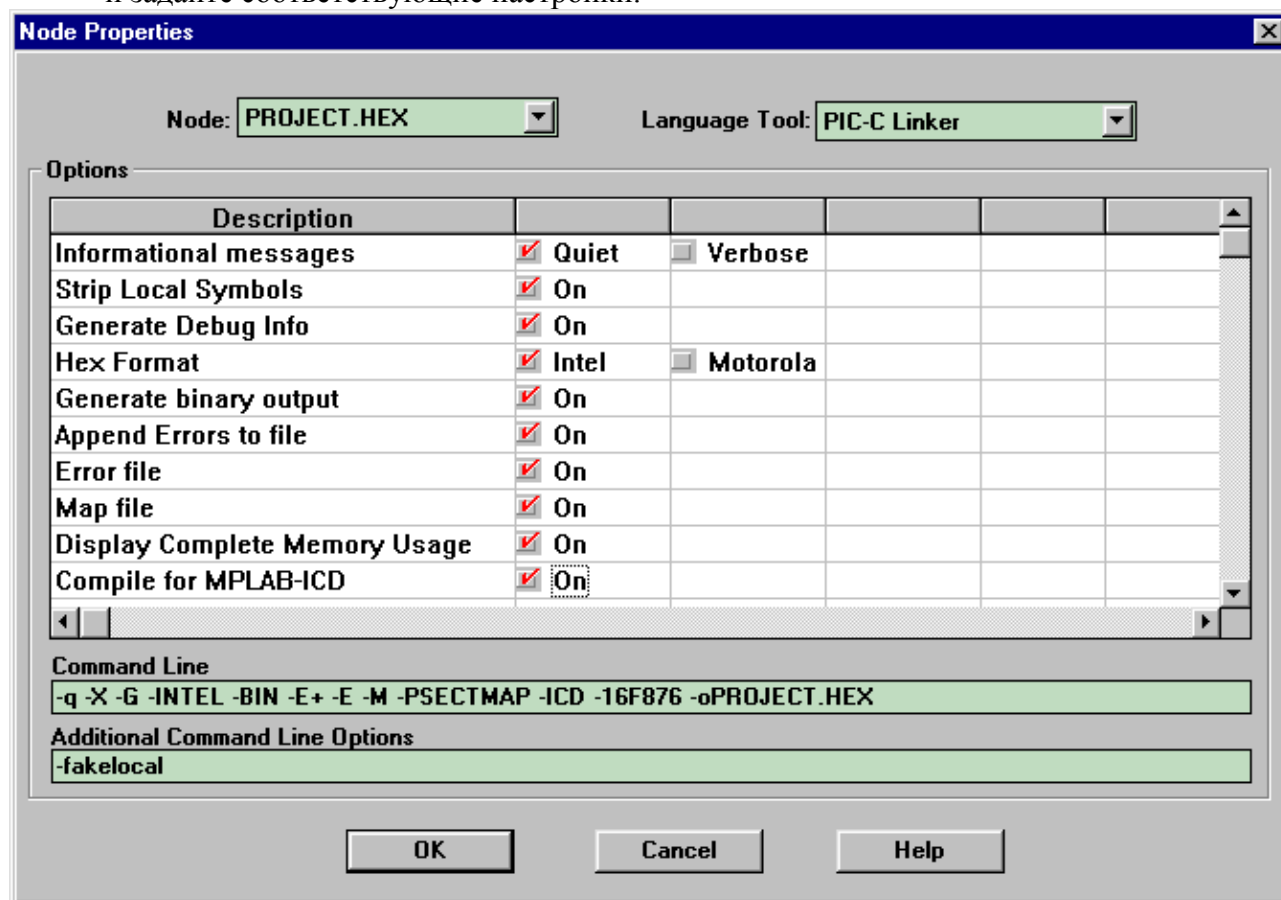


Рис. 3 Окно свойств узла.

Вашему вниманию будут предложены следующие опции:

- 1) **Informational messages** (quiet/verbose) – выдача информационных сообщений (в краткой форме/в полной);
- 2) **Strip Local Symbols** – показывать локальные переменные (символы); эта опция позволяет пользователю наблюдать значения различных локальных переменных при отладке программы;
- 3) **Generate Debug Info** – создавать отладочную информацию; эта опция полезна при работе с внутрисхемным отладчиком и симулятором;
- 4) **Hex Format** (Intel/Motorola) – создание кода в шестнадцатеричном формате (для систем Intel/Motorola);
- 5) **Generate binary output** – генерировать выходной код в двоичном формате;
- 6) **Append Errors to file** – добавлять в файл сообщения об ошибках;
- 7) **Error file** – создать отдельный файл, содержащий сообщения об ошибках;
- 8) **Map file** – создать файл, содержащий информацию об использовании памяти (карта памяти);
- 9) **Display Complete Memory Usage** – отображать итоговую информацию о полном использовании памяти;
- 10) **Compile for MPLAB-ICD** – компилировать проект для использования с внутрисхемным отладчиком MPLAB ICD.

7. Заметьте, что при включении опции «Compile for MPLAB-ICD» резервируется часть оперативной и программной памяти, что позволяет сделать файл с кодом (.hex) совместимым с внутрисхемным отладчиком **MPLAB ICD**. Для эмулятора ICEPIC 2000 эта опция не нужна.
8. Заметьте, что опция «-fakelocal» включена в «additional command line options» («дополнительные опции командной строки»). Это позволяет пользователю наблюдать глобальные переменные, используемые в программе.
Примечание: в настоящее время существует возможность модернизировать перечень предлагаемых опций с помощью дополнительных файлов, поставляемых разработчиком, в результате оптимизируется работа среды, например, ключ «-fakelocal» автоматически будет добавляться в основную командную строку при включении опции «Generate Debug Information»; но при всех преимуществах этого нововведения следует учитывать, что проекты, созданные ранее при отсутствии дополнительных опций, окажутся неработоспособны, однако, для устранения подобных проблем пользователю предлагается утилита «TOOLFIX», запуск которой автоматически исправит все конфликты.
9. Нажмите «ОК», находясь в меню добавления компонента (add node), в поле имени файла введите имя будущего исходного файла «project.c». Теперь в созданной Вами папке должно лежать два файла: project.pjt и project.c.
10. Вернитесь в диалог «edit project», выбрав меню «Project» и затем – «Edit Project».
11. Нажмите на кнопку «add node» («добавить узел [компонент]»), а затем добавьте файл «project.c».
12. Выделите имя файла project[c], щёлкнув по нему мышкой, затем нажмите кнопку «node properties» («свойства узла [компонента]»). Включите нужные опции.
В этом случае Вам, кроме вышеперечисленных, будут предложены другие опции:
 - 1) **Warning level** – выдача предупреждений;
 - 2) **Local Optimization** – оптимизация на локальном уровне;
 - 3) **Global Optimization** – оптимизация на глобальном уровне;
 - 4) **Include Search Path** – подключить путь, по которому следует искать файлы, имеющие отношение к проекту;
 - 5) **Floating point for doubles** (24 bit/32 bit) – формат представления чисел с плавающей точкой для переменных типа «double» (24 разряда/32 разряда);
 - 6) **Chars Are Signed** – переменные типа «char» – знаковые;
 - 7) **Strict ANSI Conformance** – строгое соответствие коду ANSI;
 - 8) **Define Macro** – определить макрос;
 - 9) **Undefine Macro** – не определять макрос;
 - 10) **Assembler List file** – создать файл с текстом программы на Ассемблере;
 - 11) **Generate Prototypes** – генерировать прототипы;
 - 12) **Produce Pre-processed Source Code** – производить препроцессорную обработку исходного кода; позволяет повысить эффективность;
 - 13) **Compile to Assembler Code** – компилировать применительно к коду на Ассемблере;
 - 14) **Retain Local Symbol Info** – сохранять информацию о локальных данных.
13. Дважды нажмите «Ok», чтобы вернуться в окно редактирования файла project.c.

14. Итак, мы почти у цели. Напишите эту программку (счёт секунд и отображение их числа на плате индикации):

```
#include <pic.h>


typedef unsigned char  uchar; // Определение синтаксических новообразований,
typedef signed  char  schar; // которые вводятся для удобства написания.
typedef unsigned int   uint;
typedef unsigned long  ulong;

#define F_SECONDS      0x01      // Флаг "Очередная секунда"
#define F_MSEC         0x02      // Флаг "Очередная миллисекунда"
#define INIT_TMR0      178       // для предделителя 256 и кварца Fosc = 16MHz
                                   // прерывание происходит через каждые 5 мс
#define SEC_RECOUNT  200        // константа для загрузки в пересчётный регистр
#define SET_MSECONDS Sys_flags |= F_MSEC
#define RESET_MSECONDS Sys_flags &= ~F_MSEC
#define MSECONDS (Sys_flags & F_MSEC)
#define SET_SECONDS Sys_flags |= F_SECONDS /* Установить флаг "Очередная
                                             секунда" */
#define RESET_SECONDS Sys_flags &= ~F_SECONDS /* Сбросить флаг
"Очередная                                     секунда" */
#define SECONDS (Sys_flags & F_SECONDS) /* Проверить флаг "Очередная
                                         секунда" */

uchar Sys_flags;                /* Системные флаги состояния системы */
uchar Sec_recount;              /* Пересчетный регистр секунд */
uint  Sec;
uchar Char_led[4];
uchar DigitASCII[4];

const    uchar Led8_char[]={
//const    uchar Sym[]={
    0b11000000,    //0
    0b11111001,    //1
    0b10100100,    //2
    0b10110000,    //3
    0b10011001,    //4
    0b10010010,    //5
    0b10000010,    //6
    0b11111000,    //7
    0b10000000,    //8
    0b10010000,    //9
    0b11111111,    //' '
};
/* массив кодированных значений для зажигания определённых
сегментов индикатора */

void low_level_init(void) /* Настройка портов ввода-вывода */
```

620086, г. Екатеринбург, ул. Чкалова, 3, «АВЕРОН» 

(3432) 237-079, факс: (3432) 237-038 

<http://www.averon.ru>; e-mail: ic@averon.ru, support@averon.ru 

```

{
// Инициализация направлений обмена ножек портов:
// 1 - вход;      0 - выход;

TRISA = 0b11000011;
TRISB = 0xff;
TRISC = 0;

PORTA = 0x00;           // Записать в порт A
PORTB = 0x00;           // Записать в порт B
PORTC = 0b00000001;     // Записать в порт C


/* Timer/Counter 0 Control Register (Регистр управления таймера-счётчика 0) */
OPTION = 0b11000111;    // TMR0-> 1:256 (предделитель)
TMR0 = INIT_TMR0;       // записать в таймер0 соответствующее значение
INTCON = 0;
PIE1 = 0;
PIE2 = 0;
TOIE = 1;               // Разрешить прерывание при переполнении таймера 0
}

void ParamUp(uint *data, uint max, uchar min) //функция,
                                              // инкрементирующая
                                              // параметр
{
    if(*data < max)      *data += 1;
    else                 *data = min;
}

void itoa(signed int digit) //функция разбиения числа на единицы, десятки...
{
    uint data;
    uchar i;
    uchar *s;
    i = 0;
    s = &DigitASCII[3];
    if(digit < 0) data = -digit;
    else        data = digit;
    do
    {
        *(s--) = data%10; i++;
    }
    while ((data /=10)>0);
    for(;i<4;i++) *(s--) = 0; //Заполнение незначащих нулей нулями
}

uchar led_scan;           //переменная сканирования состояния индикатора

```

620086, г. Екатеринбург, ул. Чкалова, 3, «АВЕРОН» 

(3432) 237-079, факс: (3432) 237-038 

<http://www.averon.ru>; e-mail: ic@averon.ru, support@averon.ru 

```

static void interrupt
isr(void)
{
    if(TOIF)                                // Обработчик прерываний.
    {
        TMR0 = INIT_TMR0;                  // прерывание от TMR0 с частотой 200 Гц
        TOIF = 0;                          // запись в таймер константы
        SET_MSECONDS;                       // сброс флага прерывания
        led_scan <= 1;                      // установка флага "миллисекунды"
                                           // операция сдвига на 1 разряд влево;
                                           // эквивалентна умножению на 2
        if(led_scan > 0x20) led_scan = 0x04;
        /* Вывод в LED-индикатор */
        PORTA = 0xFF;
        PORTC = 0xFF;
        PORTA = ~led_scan; // инверсное зажигание сегментов
        if(led_scan == 0x04)
        {
            PORTC = Char_led[0];
        }
        else if(led_scan == 0x08)
        {
            PORTC = Char_led[1];
        }
        else if(led_scan == 0x10)
        {
            PORTC = Char_led[2];
        }
        else if(led_scan == 0x20)
        {
            PORTC = Char_led[3];
        }
        else {led_scan = 0x04;}
    }
    else                                     // конец обработки прерывания от TIMER0
    {
        OPTION = 0b11000111;              // неизвестное прерывание – переинициализация
        INTCON = 0;
        PIE1 = 0;
        PIE2 = 0;
        TOIE = 1;                          // Разрешить прерывание при переполнении
                                           // таймера 0
    }
                                           // конец обработки несанкционированного
                                           // прерывания
}
                                           // конец обработчика прерываний

```

```


void main(void)                                /* Главная функция программы */
{
    low_level_init();                          // инициализация периферии
    GIE = 1;
    for(;;)                                    // Основной бесконечный цикл программы
    {
        if (MSECONDS)
        {
            RESET_MSECONDS;                  // если произошло прерывание от таймера, то
            // сбрасывается флаг "миллисекунды"
            if (!--Sec_recount)
            {
                // если набралось 200 прерываний, происходящих с f=200 Гц, то
                SET_SECONDS;                  // выставляется флаг "секунды"
                Sec_recount = SEC_RECOUNT;   // в пересчётный регистр вновь
                // записывается начальное число отсчёта
            }
        }
        // конец обработки флага MSECONDS

        if (SECONDS)
        {
            // если выставился флаг "секунды, то"
            RESET_SECONDS;                    /* сбрасывается флаг "Очередная секунда" */
            ParamUp(&Sec, 9999, 0);          // инкремент счётчика секунд
            itoa(Sec);                        // разложение числа секунд на разряды
            Char_led[0] = Led8_char[DigitASCII[0]]; // зажигаем тысячи
            Char_led[1] = Led8_char[DigitASCII[1]]; // зажигаем сотни
            Char_led[2] = Led8_char[DigitASCII[2]]; // зажигаем десятки
            Char_led[3] = Led8_char[DigitASCII[3]]; // зажигаем единицы

        }
        // конец обработки флага SECONDS
    }
    // конец рабочего цикла for(;;)
}
// конец функции main()

```

15. Выберите меню «Project» и нажмите «Make Project» или (F10) («Собрать [создать] проект»), чтобы всё откомпилировать. Результаты будут отображены на экране.
16. Если возникли ошибки, такие как «MPLab cannot find .hex file» («MPLab не может найти файл с кодом [*.hex]») или «Build tool not installed properly» («Неверно установлен инструмент»), удостоверьтесь, что язык программирования установлен должным образом, следуя вышеперечисленным пунктам. После устранения неполадок идём дальше.
17. Впоследствии, если у Вас будет больше исходных файлов, библиотек или объектных файлов, Вы сможете добавить их в проект, нажав кнопку «add node». Эти файлы будут скомпилированы по отдельности, затем между ними будут установлены связи.
18. Если Вы предпочитаете пошаговую (инкрементную) компиляцию, то в диалоге редактирования проекта в меню подключения путей («include path») добавьте строку «c:\ht-pic\include».

620086, г. Екатеринбург, ул. Чкалова, 3, «АВЕРОН» 

(3432) 237-079, факс: (3432) 237-038 

<http://www.averon.ru>; e-mail: ic@averon.ru, support@averon.ru 

19. Ну, вот и всё. Ваш пробный проект готов к загрузке.
20. Если Вы всё же не получили работающий проект, выполните все инструкции заново, следуя им в точности, уделяя особое внимание представленным картинкам.

Теперь Вы можете проверить работоспособность кода в симуляторе.

Использование симулятора для проверки и отладки кода.

MPLAB обладает встроенным симулятором микроконтроллеров. Если Вы компилируете программу, написанную на С или ассемблере, то Вы можете проверить работоспособность полученного кода посредством симуляции. Для этого выберите меню «options», затем – «development mode». Выберите режим «MPLAB-SIM simulator» («Симулятор MPLAB-SIM»). Используйте меню «Debug» («Отладка») и «Run» («Запуск») для выполнения команд запуска, остановки, пошагового выполнения и сброса. Установите точки останова, щёлкая правой кнопкой мыши на соответствующих строчках исходного текста программы, написанного на С, и выбирая из всплывающего (контекстного) меню пункт «breakpoint» («точка останова»). Секундомер (stopwatch), выбираемый в меню «window» («окно») и «stopwatch» очень удобен в том плане, что можно посмотреть, насколько эффективен код, то есть, сколько времени требуется для его выполнения, кроме того, можно измерять время между прерываниями таймеров и выполнять другие подобные измерительные операции; чтобы воспользоваться секундомером, сначала нужно поставить точки останова и выбрать тактовую частоту контроллера.

Испытаем на симуляторе написанный ранее проект. Для этого:

- 1) В меню «Options» – «Development mode» выберите режим «MPLAB-SIM simulator», на вкладке «Clock» задайте частоту симулируемого микроконтроллера (в данном случае – 16 МГц).
- 2) В тексте программы внутри главной функции найдите строку «RESET_MSECONDS», подведите к ней указатель мыши, правой кнопкой мыши вызовите контекстное меню и выберите пункт «Break Point(s)», в результате выбранная строка выделится красным цветом.
- 3) В меню «Window» выберите пункт «Stopwatch».
- 4) В меню «Window» – «Watch Windows» выберите пункт «New Watch Window...». В результате появится диалоговое окно добавления элемента, подлежащего просмотру – «Add Watch Symbol», в котором Вам будет предложен список доступных элементов; из этого списка следует выбрать переменную «Sec_recount». Затем нажмите кнопку «Properties» («Свойства») и измените формат представления переменной с шестнадцатеричного на десятичный, установив флажок «Decimal». После этого нажмите кнопку «OK», затем – «Close». Если окно просмотра окажется невидимым, то в меню «Window», в самом низу найдите название окна «Watch_1» и активизируйте его, щёлкнув по нему мышью.
- 5) Теперь, оперируя кнопкой «Run» в виде зелёного сигнала светофора на панели **MPLAB**, или пользуясь меню «Debug» – «Run» – «Run», Вы можете наблюдать изменение переменной «Sec_recount» и следить за приращением секундомера (после каждого нажатия «Run» симулятор выполняет программный код до точки останова, останавливает процесс и ждёт следующего нажатия).

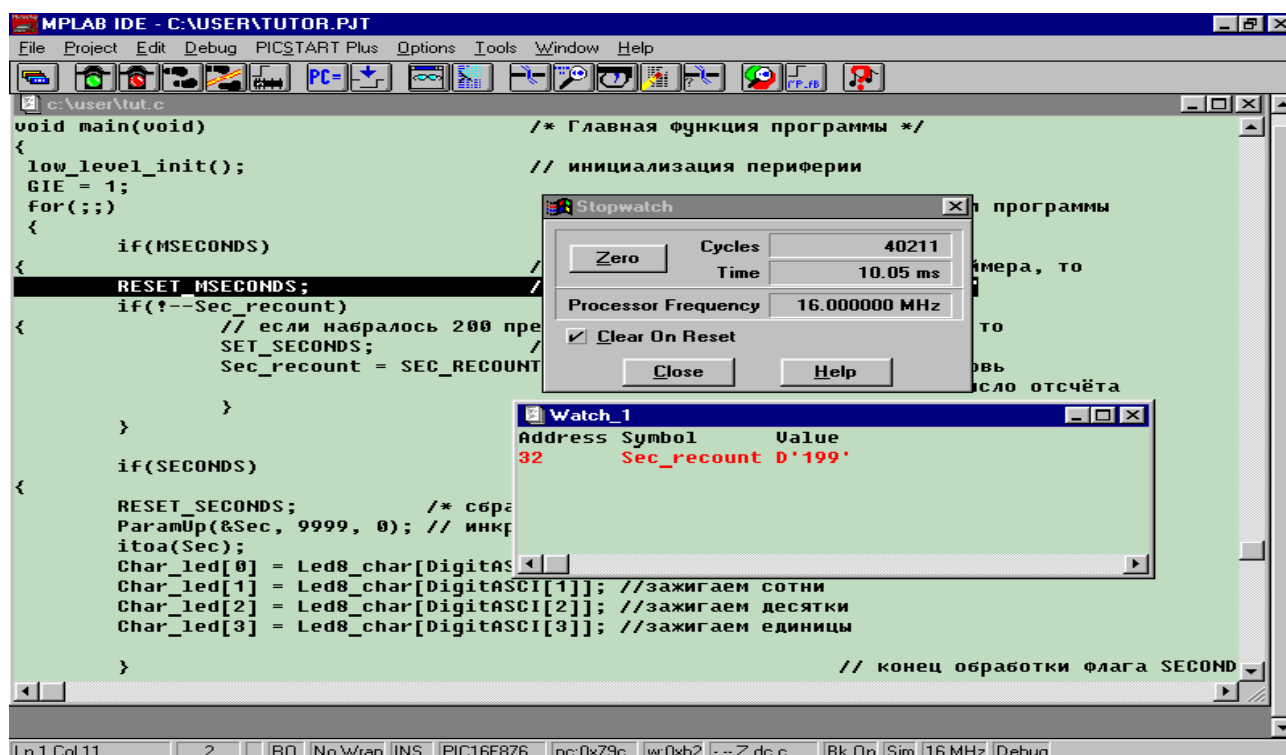


Рис. 4. Возможности MPLAB в режиме симулятора.

Использование внутрисхемного отладчика (ICD)

Внутрисхемный отладчик поддерживает не все модели микроконтроллеров, поэтому Вашему вниманию предлагается таблица, содержащая информацию о поддерживаемых микроконтроллерах и размерах памяти, необходимых для нужд внутрисхемного отладчика.

Таблица 3. Поддерживаемые микроконтроллеры.

Контроллер	Резервируемые регистры ОЗУ	Резервируемая FLASH-память
PIC16F870/871/872	0x70, 0x0BB...0x0BF	0x06E0...0x07FF
PIC16F873/874	0x70, 0x0EB...0x0F0	0x0EE0...0x0FFF
PIC16F876/877	0x70, 0x1EB...0x1EF	0x1F00...0x1FFF

Для корректного подключения Вам предлагается следующая инструкция (за основу взят внутрисхемный отладчик «MP3» фирмы «Аверон»).

ICD подключается стандартным 9-и контактным кабелем RS-232 к последовательному порту компьютера. Плата отладчика через разъем внутрисхемной отладки X1 подключается к плате с отлаживаемым микроконтроллером. Назначение выводов разъема X1 приведено в Таблице 4.

Питание платы эмулятора производится от отлаживаемой платы. В случае отладки прибора с батарейным питанием или от микрopotребляющего источника, а также в случае программирования контроллера в обесточенной плате, можно запитать плату отладчика через разъем X3 (смотри Таблицу 4).

С помощью разъема X4 можно внутрисхемно поменять прошивку микроконтроллера отладчика. Если у Вас нет программатора, поддерживающего программирование PIC16F876, Вы можете обновить прошивку прямо из среды разработки **MPLAB**.

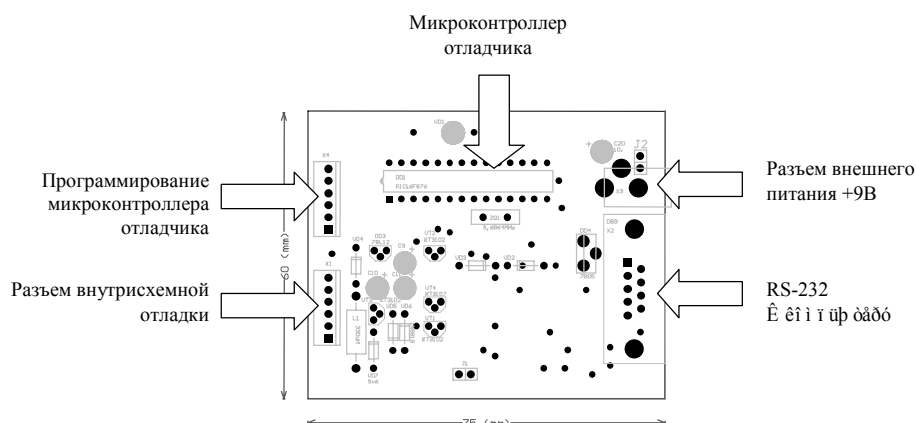


Рис. 5. Расположение основных элементов на плате.

Схема подключения **MPLAB ICD** к отлаживаемой схеме приведена на Рис.6.

Таблица 4. Назначение выводов разъемов.

Разъем	Номер вывода	Назначение вывода	Назначение разъема
X1	1	GND	Внутрисхемная отладка и программирование микроконтроллеров серии PIC16F87x.
	2	Напряжение программирования/ Сброс	
	3	RB7 – Данные программирования	
	4	RB6 – Тактирование программирования	
	5	Выбор режима низковольтного программирования	
	6	+Vdd (напряжение питания)	
X2	1, 6, 9	Не используется	Связь с компьютером, управление отладчиком с помощью MPLAB .
	2	RX	
	3	TX	
	4	DTR	
	5	GND	
	7	RTS	
X3	Центральный	+7...12В (вывод)	Разъем внешнего источника питания.
	Наружный	GND	
X4	1	GND	Разъем внутрисхемного программирования контроллера отладчика.
	2	Напряжение программирования/ Сброс	
	3	RB7 – Данные программирования	
	4	RB6 – Тактирование программирования	
	5	Выбор режима низковольтного программирования	
	6	+Vdd (напряжение питания)	

Замечание. При использовании возможностей внутрисхемной отладки **MPLAB ICD** требует резервирования следующих ресурсов отлаживаемого кристалла:

- выводы RB6 и RB7 используются для отладки и не могут использоваться как обычные порты ввода/вывода. После отладки и программирования устройства эти выводы могут использоваться по Вашему усмотрению;
- первой инструкцией, расположенной по адресу h'000' должна быть «NOP»;
- отладчик использует один уровень стека;
- отладчик резервирует часть ОЗУ и памяти программ (смотри Таблицу 3)

Преимущество внутрисхемного отладчика перед симулятором заключается в том, что Вы можете видеть, что происходит внутри кристалла. Вы можете запускать код в пошаговом

режиме, ставить точки останова, наблюдать переменные и засекают время выполнения той или иной части кода. Все действия те же, что и при использовании симулятора.

Разница между внутрисхемным отладчиком-эмулятором и симулятором в том, что симулятор – это всего лишь программа, исполняемая компьютером, а внутрисхемный отладчик-эмулятор включается в испытываемую схему и может взаимодействовать с реальными объектами “железного” мира.

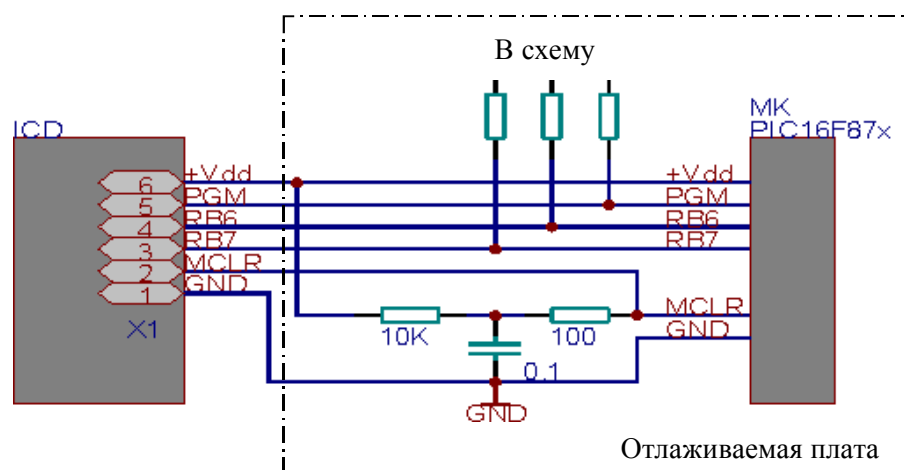


Рис. 6. Пример подключения MPLAB ICD к отлаживаемой плате.

Мы будем использовать внутрисхемный отладчик **MPLAB-ICD**. Чтобы понять основную идею о том, как работает отладчик, Вам следует почитать документацию, которую можно отыскать в справочной системе **MPLAB** (меню «Help», затем – «MPLAB-ICD help»), либо воспользоваться документацией из серии технической поддержки фирмы «Аверон».

Теперь следует настроить отладчик-эмулятор.

1. Подключите отладчик к порту COM1 Вашего компьютера через последовательный кабель.
2. Подключите отладчик согласно приведённой выше инструкции.
3. Подайте на отлаживаемую плату питание.
5. Запустите **MPLAB**. Загрузите проект (можно использовать проект, описанный выше). Выберите меню «Options», затем – «Development Mode». Выберите нужные опции.
6. Вы увидите диалоговое окно. Выберите нужные опции.
7. Установки, предлагаемые по умолчанию, подходят практически к любой ситуации, за исключением генератора тактовой частоты (осциллятора). При использовании внешнего кристалла (кварца) с частотами выше 8 МГц следует использовать установки, предлагаемые по умолчанию. В случае, если частотодающий кварц работает на частоте ниже 8 МГц, в установках осциллятора следует использовать «XT» вместо «HS». При использовании в качестве задающего генератора RC-цепочки в настройках осциллятора следует ставить «RC» вместо «HS».
8. С помощью предлагаемых выше аппаратных средств отладки можно проверить, как работает отладчик с конкретным проектом.
9. Откомпилируйте программу, выбрав меню «Project» и затем – «Make Project». Либо можно просто нажать клавишу F10.

10. Загрузите программу во FLASH-память отладчика, нажав кнопку «Program» («Программирование») на панели управления **MPLAB ICD**.
Замечание: при прошивке следует обратить внимание на настройки программирования, где, кроме выбора соответствующего режима осциллятора, рекомендуется переопределить конечный адрес памяти программ с помощью кнопки «Def. Addr» и сбросить следующие биты конфигурации: Watchdog Timer, Power Up Timer, Brown out Detect, Low Voltage Program, Code Protect Data EE, Code Protect (все они должны находиться в состоянии Off/Disable); кроме того, не забудьте выставить нужный диапазон рабочих частот, соответствующий используемому в Вашем проекте кварцу.
11. Когда прошивка завершится, запустите программу, выбрав меню «Debug» («Отладка»), затем – подменю «Run» («Запуск») и «Run» («Пуск»). Либо просто нажмите клавишу F9. На плате индикации Вы должны увидеть отображение процесса инкрементирования секунд.
12. Чтобы убедиться в возможностях отладчика, попробуйте с помощью точки останова и окна просмотра понаблюдать за какой-нибудь переменной, например за переменной «Sec». Для этого:
 - 6) Остановите работу микроконтроллера, нажав кнопку «Halt» в виде красного сигнала светофора на панели **MPLAB**, либо выбрав последовательно пункты меню «Debug» – «Run» – «Halt».
 - 7) В тексте программы внутри главной функции найдите строку «itoa(Sec);», подведите к ней указатель мыши, правой кнопкой мыши вызовите контекстное меню и выберите пункт «Break Point(s)», в результате выбранная строка выделится красным цветом.
 - 8) В меню «Window» – «Watch Windows» выберите пункт «New Watch Window...». В результате появится диалоговое окно добавления элемента, подлежащего просмотру – «Add Watch Symbol», в котором Вам будет предложен список доступных элементов; из этого списка следует выбрать переменную «Sec». Затем нажмите кнопку «Properties» («Свойства») и измените формат представления переменной с шестнадцатеричного на десятичный, установив флажок «Decimal». После этого нажмите кнопку «OK», затем – «Close». Если окно просмотра окажется невидимым, то в меню «Window», в самом низу найдите название окна «Watch_1» и активизируйте его, щёлкнув по нему мышью.
 - 9) Теперь, оперируя кнопкой «Run» в виде зелёного сигнала светофора на панели **MPLAB**, или пользуясь меню «Debug» – «Run» – «Run», Вы можете наблюдать изменение переменной «Sec» синхронно с её отображением на плате индикации (после каждого нажатия «Run» отладчик выполняет программный код до точки останова, останавливает процесс и ждёт следующего нажатия).

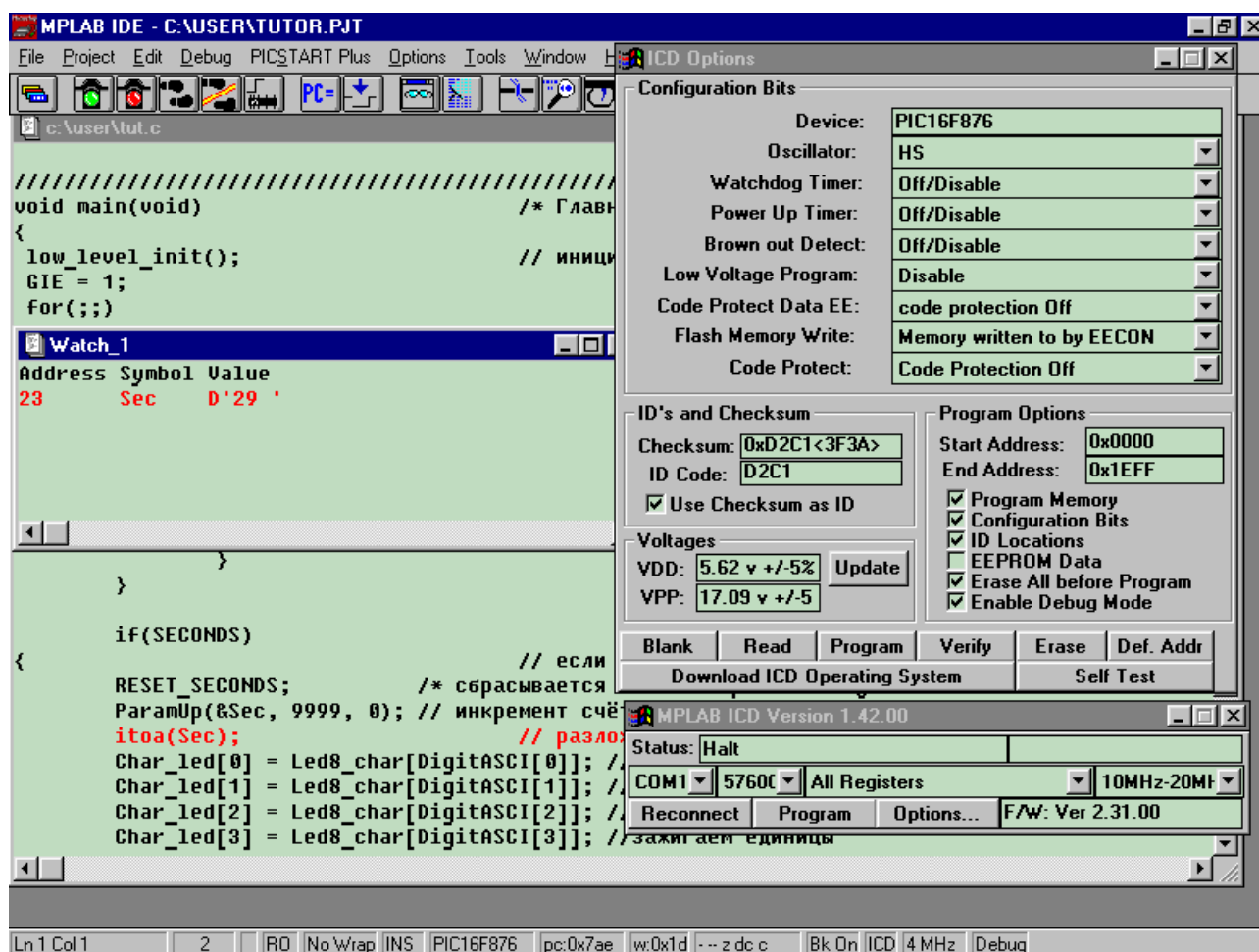


Рис. 7. Возможности MPLAB в режиме ICD.

13. Теперь Вы можете запрограммировать микроконтроллер таким образом, чтобы он мог работать без отладчика. Для этого следует в окне «ICD Options» в разделе «Program Options» снять флажок «Enable Debug Mode» и вновь прошить микроконтроллер.
14. Конец.

Некоторые советы по эффективному программированию с использованием компилятора Hitech C.

Существуют специальные приёмы, позволяющие оптимизировать программный код. Вашему вниманию предлагаются некоторые из них:

- 1) При инициализации переменных в начале программы обратите внимание на порядок инициализации: вначале следует инициализировать переменные в нулевом банке (bank0), затем в первом (bank1), втором (bank2) и так далее по порядку.
- 2) Возможно, некоторые переменные вовсе не требуется инициализировать, поэтому ни к чему тратить на них время.
- 3) Там, где возможно, старайтесь переставлять операторы местами таким образом, чтобы компилятор не загружал лишний раз аккумулятор.

- 4) Чтобы избежать переключений между банками, в арифметических операциях используйте переменные из одного банка.
- 5) Если возможно, попытайтесь использовать 8-миразрядную арифметику вместо 16-тиразрядной.
- 6) По возможности пользуйтесь указателями на элемент массива, а не индексами.
- 7) Использование последовательности операторов `if`, `else if`, `else if` ... часто даёт более эффективный код, нежели использование оператора `case`.
- 8) При использовании структуры `switch - case` старайтесь, чтобы константы принимали последовательные значения, между которыми отсутствуют разрывы.
- 9) Из соображений минимума переключений между банками вытекает, что данная организация программы:

```
var = value1;  
if (!flag)  
var = value2;
```

более оптимальна, чем такая:

```
if (flag)  
var = value1;  
else  
var = value2;
```

В этом случае необходимо убедиться, что во время выполнения этой части кода не произойдёт прерывание.

- 10) Очистка, инкремент или декремент байта выполняется за один цикл. А вот присвоение байту какого-либо значения требует двух циклов (значение \rightarrow W, и W \rightarrow byte).
- 11) По возможности используйте битовые переменные, а не переменные типа `unsigned char`. Установка, очистка, проверка или пропуск бита выполняется за один цикл. Хотя Вы и не можете объявить бит внутри функции, Вы можете выиграть на глобальном объявлении этого бита.
- 12) При вызове функций существуют определённые издержки. Попробуйте заменить свои небольшие функции макросами. Старайтесь избегать применения функций в обработчике прерываний.
- 13) Большие блоки однотипного кода следует заменить функциями, если позволяет размер стека.
- 14) Для переключения отдельных битов пользуйтесь записью типа `~0` вместо `0xFFFF`, поскольку в этом случае, при условии, что переменная типа `"int"` лежит в диапазоне от 16 до 32 бит, обеспечивается аппаратная независимость. (Кстати, следует помнить, что в 8 разрядных переменных биты нумеруются справа налево.) Пример кода на C:

```
unsigned char x=0b0001;  
bit_set(x,3); //теперь x=0b1001;  
bit_clr(x,0); //теперь x=0b1000;*/  
#define bit_set(var,bitno) ((var) |= 1 << (bitno))// макрос, реализующий установку бита;  
#define bit_clr(var,bitno) ((var) &= ~(1 << (bitno)))// макрос, реализующий сброс бита;
```

Этот же приём можно использовать для переключения нескольких битов согласно маске, в этом случае фрагмент кода выглядит следующим образом:

```
unsigned char x=0b1010;  
bits_on(x,0b0001); //теперь x=0b1011
```

```
bits_off(x,0b0011); //теперь x=0b1000
#define bits_on(var,mask) var |= mask //макрос, реализующий установку
// битов, определяемых маской;
#define bits_off(var,mask) var &= ~0 ^ mask //макрос, реализующий сброс
// битов, определяемых маской;
```

15) Для осуществления более эффективного умножения или деления на 2, 4, 8 и так далее следует пользоваться операциями логического сдвига: сдвиг влево соответствует умножению, а сдвиг вправо – делению; например, запись $x \ll 1$ соответствует умножению переменной на 2, а $x \ll 2$ – умножению на 4, $x \ll 3$ – умножению на 8, $x \gg 1$ – делению на 2, и так далее.

16) Часто встречается ситуация, когда целочисленная переменная (“int”) занимает 16 разрядов, в то время как байт состоит из 8 бит; так вот, чтобы обращаться к отдельным байтам в составе переменной, воспользуйтесь следующим примером:

```
unsigned int x;
unsigned char y;
#define hbyte(x) (unsigned char)(x>>8) // макрос определения старшего байта
#define lbyte(x) (unsigned char)(x & 0xFF) // макрос определения младшего байта
x=0x1234;
y=hbyte(x); // теперь y=0x12
y=lbyte(x); // теперь y=0x34
```

Этот пример хорош для просмотра переменной в любом банке, однако он не подходит для изменения байта в составе переменной; для этого необходимы указатели и разные директивы #define для разных банков:

```
#define lbyte_atbank0(x) (unsigned char)((*(unsigned char *)&x)+0)
#define hbyte_atbank0(x) (unsigned char)((*(unsigned char *)&x)+1)
#define lbyte_atbank1(x) (unsigned char)((*(bank1 unsigned char *)&x)+0)
#define hbyte_atbank1(x) (unsigned char)((*(bank1 unsigned char *)&x)+1)
```

17) При инициализации портов следует пользоваться следующим советом: при переопределении линий порта со входа на выход, сначала рекомендуется определять направление, а уже затем изменять значение; для иллюстрации этого правила рассмотрим два примера:

```
//это правильный способ: сначала –направление, потом –порт;
#define INPUT 1 // вход
#define OUTPUT 0 // выход
#define CLOCK RB1 // 1-й канал порта
#define CLOCK_DIRECTION TRISB0 // защёлка направления
CLOCK_DIRECTION=OUTPUT; // направление – выход
DelayUs(10); //задержка на 10 микросекунд
CLOCK=1; // запись значения
```

```
//неправильный способ: значение порта может остаться нулевым
#define INPUT 1 // вход
#define OUTPUT 0 // выход
#define CLOCK RB1 // 1-й канал порта
#define CLOCK_DIRECTION TRISB0 // защёлка направления
CLOCK=1; // запись значения
DelayUs(10); //задержка на 10 микросекунд
CLOCK_DIRECTION=OUTPUT; // направление – выход
```


Если какой-нибудь канал порта настроен на вход, то при выполнении операции типа «чтение – модификация – запись» сигнал на выводе будет считан и записан в защёлку данных поверх предыдущего значения. Пока этот канал настроен как вход, никаких проблем не возникает, однако, когда этот канал будет перенастраиваться на выход, значение в защёлке данных может отличаться от требуемого.

18) Для организации в программе бесконечного цикла можно использовать как оператор `for (; ;)` без параметров, так и оператор `while(1)`, где в качестве условия поставлено заведомо верное выражение; однако практика программирования показывает, что конструкция с оператором `for (; ;)` более эффективна, нежели с `while(1)`.

Список рекомендуемой литературы.

1. По языку программирования Си:
 - Керниган Б., Ритчи Д., Язык программирования Си. Пер. с англ., 3-е изд., испр. – СПб.: «Невский диалект», 2001. – 352 с.: ил.
 - Керниган Брайан В., Пайк Роб, Практика программирования. Пер. с англ. – СПб.: «Невский диалект», 2001. – 381 с.: ил..
2. Описание пакета MPLAB и сопутствующая информация:
 - Описание MPLAB ICD (на русском языке):
http://www.microchip.ru/files/software/mplabi/mplab_ide.pdf.
 - Описание PIC16F62х (на русском языке):
<http://www.microchip.ru/files/d-sheets-rus/pic16f62x.pdf>
 - Описание пакета MPLAB (на английском языке):
<http://www.microchip.com/1010/pline/tools/picmicro/devenv/mplabi/index.htm>
 - Описание пакета MPLAB (на русском языке):
http://www.microchip.ru/files/software/mplabi/mplab_ide.pdf
 - Руководство пользователя MPASM (на русском языке):
<http://www.microchip.ru:80/lit/articles/21/73/MPASM.pdf>
 - Описание MPLAB ICD (на английском языке):
<http://www.microchip.com/1010/pline/tools/picmicro/icds/mplabidc/index.htm>
 - Ещё одно описание MPLAB ICD (на русском языке):
<http://u1.chat.ru/pdf/51184br.pdf>
3. Для начального ознакомления с предметом:
 - Астапкович А. М., Семинары ASK LAB 1997/ Под ред. М. Б. Сергеева – СПб.: Политехника, 1998. – 134 с.: ил.