Embedded Programming in C

55:036
Embedded Systems and Systems Software

Plusses and Minuses of High-level languages for Embedded Programming

- Plusses
  - Easier syntax (usually)
  - Lots of libraries and drivers
- Minuses
  - Loss of efficiency
  - Loss of direct control at the hardware level
    - e.g. typically can't count instruction cycles
    - Some C compilers (e.g.CCS) have problems coping with the PIC's Harvard Architecture

PIC C Compilers

- Microchip C18 (MCC)
- Hi-Tech
- CCS
- Etc.

PIC C Compilers

- Microchip C18 (MCC)
- Hi-Tech
- CCS
- Etc.

This is the compiler we will use.

A student version (valid for 60 days) can be downloaded from: www.microchip.com

PIC C Compilers

- Unfortunately, the various PIC C Compilers are not compatible.
- Some differences are quite significant
  - Different naming conventions
  - Different constructs for bit-level access to ports and other registers
  - Different directives and built-in functions
  - Different constructs for interrupts and ISRs
  - etc.
- Porting C code from one compiler to another can be a non-trivial task

C18 Compiler Documentation

- C18 C Compiler: Getting Started
  - Pay Special Attention to Chapters 3 and 4.
- C18 C Compiler User's Guide
- C18 C Compiler Libraries
- Documentation is linked on the Class Web Site.

C18 Integer Data Types

TYPE-SPECIFIER	SIZE
char	8-bit (signed by default)
signed char	8-bit (signed)
unsigned char	8-bit (unsigned)
int	16 bit (signed)
unsigned int	16-bit (unsigned)
short	Same as int
unsigned short	Same as unsigned int
short long	24-bit (signed)
unsigned short long	24-bit (unsigned)
long	32-bit (signed)
unsigned long	32-bit (unsigned)

Note: Multiple-byte data is stored in "little endian" form

C18 Floating Point Data Types

TYPE-SPECIFIER	SIZE
float `	32-bit (IEEE format)
double	Same as float

C18 Data Types—Constant Strings

- Constant strings can be stored in program memory:

  const rom char str[] = {'H', 'i', '\0'};
- All strings declared without the keyword *rom* are stored in data memory:

  const char str[] = {'H', 'i', '\0'};
- Pointers to program memory (rom) strings and pointers to data memory strings are not compatible since they point to different address spaces.

Note: Different versions of the strcpy function

```
/*
* Copy string s2 in data memory to string s1 in data memory
*/
char *strcpy (auto char *s1, auto const char *s2);
/*
* Copy string s2 in program memory to string s1 in data
* memory
*/
char *strcpypgm2ram (auto char *s1, auto const rom char *s2);
/*
```

C18 Example: LCD_Routines.h

```
#ifndef __LCDROUTINES_H
#define __LCDROUTINES_H

#define CMD  0
#define CH   1

/*****************************************************************
*  Function Prototypes for LCD_Routines
*****************************************************************/
void lcd_write_nibble(char);
void lcd_write_byte( char, char);
void lcd_init(void);
void lcd_gotoxy( char, char);
void lcd_putc( char c);
void lcd_ram_puts(const char *str, char len);
void lcd_rom_puts(const char rom *str, char len);

#endif
```

C18 Example: LCD_Routines.c

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                    LCD_Routines.c
////
////  lcd_init()   Must be called before any other function.

//// lcd_write_nibble(c)  write high-order 4 bits of c to LCD
////
//// lcd_write_byte(mode, c) write byte to LCD. Mode = CMD/CH
////
////  lcd_putc(c)  Will display c at the next position of the LCD.
////              The following have special meaning:
////                  \f  Clear display
////                  \n  Go to start of second line
////                  \b  Move back one position
////
////  lcd_gotoxy(x,y) Set write position on LCD (upper left is 0,0)
////
////  lcd_ram_puts(str,len) write data memory string str to LCD
////
//// lcd_rom_  puts(str, len) write program memory string str to LCD
////
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

LCD_Routines.c (cont.)

```
#include <p18cxxx.h>
#include <delays.h>
#include "LCD_Routines.h"

const char LCD_INIT_STRING[] = {0x0C, 0x06, 0x80, 0};
                    // These bytes need to be sent to the LCD
                    // to properly initialize and configure it.


void lcd_write_nibble( char n ) {
    PORTEbits.RE1 = 1;      //set LCD Enable high
    PORTD = n;              //Write high-order nibble of n to PortD
    PORTEbits.RE1 = 0;      //set LCD Enable low
    PORTEbits.RE1 = 1;      //set LCD Enable high
}

void lcd_write_byte( char mode, char n ) {
    PORTEbits.RE0 = mode;   //RS = 0 for command
    lcd_write_nibble(n);
    n = n<<4;
    lcd_write_nibble(n);
    Delay10TCYx(16);     //delay for 160 instruction cycles (40 microsec.)
}
```

LCD_Routines.c (cont.)

```
#include <p18cxxx.h>
#include <delays.h>
#include "LCD_Routines.h"

const char LCD_INIT_STRING[] = {0x0C, 0x06, 0x80, 0};
                    // These bytes need to be sent to the LCD
                    // to properly initialize and configure it.


void lcd_write_nibble( char n ) {
    PORTEbits.RE1 = 1;      //set LCD Enable high
    PORTD = n;              //Write high-order nibble of n to PortD
    PORTEbits.RE1 = 0;      //set LCD Enable low
    PORTEbits.RE1 = 1;      //set LCD Enable high
}

void lcd_write_byte( char mode, char n ) {
    PORTEbits.RE0 = mode;   //RS = 0 for command
    lcd_write_nibble(n);
    n = n<<4;
    lcd_write_nibble(n);
    Delay10TCYx(16);     //delay for 160 instruction cycles (40 microsec.)
}
```

Always put this at the beginning of your program

LCD_Routines.c (cont.)

```
#include <p18cxxx.h>
#include <delays.h>
#include "LCD_Routines.h"

const char LCD_INIT_STRING[] = {0x0C, 0x06, 0x80, 0};
                    // These bytes need to be sent to the LCD
                    // to properly initialize and configure it.


void lcd_write_nibble( char n ) {
    PORTEbits.RE1 = 1;      //set LCD Enable high
    PORTD = n;              //Write high-order nibble of n to PortD
    PORTEbits.RE1 = 0;      //set LCD Enable low
    PORTEbits.RE1 = 1;      //set LCD Enable high
}

void lcd_write_byte( char mode, char n ) {
    PORTEbits.RE0 = mode;   //RS = 0 for command
    lcd_write_nibble(n);
    n = n<<4;
    lcd_write_nibble(n);
    Delay10TCYx(16);     //delay for 160 instruction cycles (40 microsec.)
}
```

This is needed because we use delay functions from the C18 libraries. (See the C18 C Compiler Libraries Manual for details.)

LCD_Routines.c (cont.)

```
#include <p18cxxx.h>
#include <delays.h>
#include "LCD_Routines.h"

const char LCD_INIT_STRING[] = {0x0C, 0x06, 0x80, 0};
                    // These bytes need to be sent to the LCD
                    // to properly initialize and configure it.


void lcd_write_nibble( char n ) {
    PORTEbits.RE1 = 1;      //set LCD Enable high
    PORTD = n;              //Write high-order nibble of n to PortD
    PORTEbits.RE1 = 0;      //set LCD Enable low
    PORTEbits.RE1 = 1;      //set LCD Enable high
}

void lcd_write_byte( char mode, char n ) {
    PORTEbits.RE0 = mode;   //RS = 0 for command, 1 for data
    lcd_write_nibble(n);
    n = n<<4;
    lcd_write_nibble(n);
    Delay10TCYx(16);     //delay for 160 instruction cycles (40 microsec.)
}
```

Note how we write to ports.

LCD_Routines.c (cont.)

```
void lcd_init(void) {
   char i;
   const char *strPtr;
   PORTEbits.RE0 = CMD;        //RS = 0 for command
   Delay10KTCYx(100);          //delay for 250 msec.
   for (i=0; i<3; i++) {       // Initialization sequence
      lcd_write_nibble(0x30);  //set 8-bit mode three times
      Delay10KTCYx(2);         //delay for 5 msec
   }
   lcd_write_nibble(0x20);     //set 4-bit mode
   Delay10TCYx(16);            //delay for 160 instruction cycles (40 microsec.)
   lcd_write_byte(CMD, 0x28); //Set 4-bit mode, two line, 5x7 chars
   lcd_write_byte(CMD, 0x01); //clear display. Careful, this takes a LONG time
   Delay10KTCYx(50);           //delay for 125 msec.
   strPtr = LCD_INIT_STRING;
   while(*strPtr) {
    lcd_write_byte(CMD,*strPtr); //No clear or home cmds in LCD_INIT_STRING
    strPtr++;
   }
}
```

LCD_Routines.c (cont.)

```
void lcd_gotoxy( char row, char col) {
   char address;

   if(row!=0)
     address=0x40;
   else
     address=0;
   address=address + col;
   lcd_write_byte(CMD,0x80|address);
}

void lcd_putc( char c) {
   switch (c) {
     case '\f'   : lcd_write_byte(CMD,1);  // Clear Display.
                   Delay10KTCYx(50);break; //Takes a LONG time. Delay for 125 msec.
     case '\n'   : lcd_gotoxy(1,0); break; // newline
     case '\b'   : lcd_gotoxy(0,0);  break; // beginning of first line
     default     : lcd_write_byte(CH,c); break; //send char to display
   }
}
```

LCD_Routines.c (cont.)

```
void lcd_ram_puts(const char str[], char len) {
        char i;
        for (i=0; i<len; i++)
          lcd_putc(str[i]);
}

void lcd_rom_puts(const char rom *str, char len) {
        char i;
        for (i=0; i<len; i++)
          lcd_putc(str[i]);
}
```

LCDtest.c
program memory string example

```
#include <p18cxxx.h>
#include <delays.h>
#include "LCD_Routines.h"
void main() {

   static const rom char ConstString[] = {'H', 'i', '\n', 'D', 'u', 'd', 'e','\0'};
// const char VarString[] = {'H', 'e', 'l', 'l','o', '\n ', 'W', 'o','r', 'l', 'd','\0'};

    ADCON1 = 0b10001110;        // Enable PORTA & PORTE digital I/O pins
   TRISA  = 0b11100001;        // Set I/O for PORTA
   TRISB  = 0b11011100;        // Set I/O for PORTB
   TRISC  = 0b11010000;        // Set I/0 for PORTC
   TRISD  = 0b00001111;        // Set I/O for PORTD
   TRISE  = 0b00000000;        // Set I/O for PORTE
   lcd_init();
   lcd_gotoxy(0,0);
   lcd_rom_puts(ConstString, 7);
// lcd_ram_puts(VarString, 11);
   while(1);
}
```

LCDtest.c
data memory string example

```
#include <p18cxxx.h>
#include <delays.h>
#include "LCD_Routines.h"
void main() {

// static const rom char ConstString[] = {'H', 'i', '\n', 'D', 'u', 'd', 'e','\0'};
  const char VarString[] = {'H', 'e', 'l', 'l','o', '\n ', 'W', 'o','r', 'l', 'd','\0'};

   ADCON1 = 0b10001110;          // Enable PORTA & PORTE digital I/O pins
  TRISA  = 0b11100001;          // Set I/O for PORTA
  TRISB  = 0b11011100;          // Set I/O for PORTB
  TRISC  = 0b11010000;          // Set I/0 for PORTC
  TRISD  = 0b00001111;          // Set I/O for PORTD
  TRISE  = 0b00000000;          // Set I/O for PORTE
  lcd_init();
  lcd_gotoxy(0,0);
// lcd_rom_puts(ConstString, 7);
  lcd_ram_puts(VarString, 11);
  while(1);
}
```

C18--Using Interrupts

```
void HiPriISR(void);
void LoPriISR(void);

#pragma code highVector=0x08
void atHighVector(void)
{
 _asm GOTO HiPriISR _endasm
}
#pragma code

#pragma code lowVector=0x18
void atLowVector(void)
{
 _asm GOTO LoPriISR _endasm
}
#pragma code
```

C18--Using Interrupts

```
void HiPriISR(void);
void LoPriISR(void);

#pragma code highVector=0x08
void atHighVector(void)
{
 _asm GOTO HiPriISR _endasm
}
#pragma code

#pragma code lowVector=0x18
void atLowVector(void)
{
 _asm GOTO LoPriISR _endasm
}
#pragma code
```

Prototypes for ISRs

C18--Using Interrupts

```
void HiPriISR(void);
void LoPriISR(void);

#pragma code highVector=0x08
void atHighVector(void)
{
 _asm GOTO HiPriISR _endasm
}
#pragma code

#pragma code lowVector=0x18
void atLowVector(void)
{
 _asm GOTO LoPriISR _endasm
}
#pragma code
```

Insert goto statement at high priority interrupt vector address (0x08)

C18--Using Interrupts

```
void HiPriISR(void);
void LoPriISR(void);

#pragma code highVector=0x08
void atHighVector(void)
{
 _asm GOTO HiPriISR _endasm
}
#pragma code

#pragma code lowVector=0x18
void atLowVector(void)
{
 _asm GOTO LoPriISR _endasm
}
#pragma code
```

Insert goto statement at low priority interrupt vector address (0x18)

C18--Using Interrupts (cont.)

```
#pragma interruptlow LoPriISR // Low-priority interrupt service routine
void LoPriISR(void) {

  /* Low priority ISR code goes here. */
  /* Just write it like an ordinary function */
  /* Automatically saves and restores "compiler-managed resources" */
  /* See section 3.4 in Compiler User's Guide */

}


#pragma interrupt HiPriISR      // High-priority interrupt service routine void
HiPriISR(void) {

  /* High priority ISR code goes here    */
  /* Automatically supports "retfie fast" */

}
```

C18—Additional Examples

- Appendix A7 in the text—P3 and P4 templates programmed in C
  - Careful! Several errors
    - P3 example doesn't work due to some timing problems
      - The T40() function doesn't delay for 40 microseconds.
    - DisplayC() doesn't handle strings stored in program memory—i.e. it's not equivalent to the assembly language version in Chapter 12
- Chapter 5 of the C18 Compiler User's Guide
  - You can find the code for these examples and others in:

    C:\MCC18\example

One Last Note

- You cannot use RS-232 Serial I/O in your C program if you are using QwikBug
  - QwikBug takes over the serial port
  - Generates debug signals from serial input
- If you need to use RS-232, it is recommended that you use a PIC that is not programmed with QwikBug
  - Download programs to PIC via Device Programmer or Bootloader
  - Be sure to remove the "QB break" jumper on the QwikFlash board