C program library

For Keil Software, Inc. C251 USB software

Author: Walter Banks, Jr.

Tuesday, December 16, 1997

Contents

Introduction

This document summarizes the C library programs which a software developer can use when building embedded applications for Intel's 8x930Hx and Ax Universal Serial Bus (USB) peripheral controllers. This document summarizes the C library programs, describes vendor-specific implementations and lists the C programs. Many of these implementations are compiler specific; hence, this document focuses on the Keil Software, Inc. "251USB" software tools. This document describes each compiler-specific implementation.

The major sections in this document are:

Requirements List - Defines the requirements to which the C programs were developed. The programs conform to the requirements defined in this section.

C Programs Summary - Provides a high-level description of each program.

Programming Details - Defines the syntax, commands and keywords used in this document.

C Programs - Contains a complete listing for each program. The program names are:

- C_ASSBLY.C
- INTERUPT.C
- PCA.C
- PORTS.C
- SER_COM.C
- SERIAL.C
- SFR_MEM.C
- TIMER0.C
- TOOLS.C
- VAR_MEM.C
- WDT.C

Related Information

Document Name	8x931AA, 8x931HA Universal Serial Bus Peripheral Controller User's Manual
Order Information	Call Intel's literature center 1-800-548-4725, order number 273102
Intel Web location	http://developer.intel.com/design/usb
Keil Web location	http://www.keil.com/

Requirements List

C items for embedded controller programming

1. Writing code to exercise the Special Function Registers:
    1.1. Reading and storing to the Special Function Registers
    1.2. Writing and changing Special Function Registers contents
2. Writing code to exercise the peripherals:
    2.1. Setting up the timer/counters in different modes
    2.2. Setting up the watch dog timer
    2.3. Setting up the PCA
    2.4. Using the serial port in different modes
3. Reading from and writing to memory (RAM):
    3.1. Placing variables into specified memory locations
    3.2. Locating tables into specified memory locations
    3.3. Copying data from internal memory to external memory
    3.4. Copying data from external memory to internal memory
4. Dealing with Interrupts:
    4.1. Code for internal and external interrupt service routines
    4.2. Setting up interrupt priorities
    4.3. Disabling and enabling interrupts
5. Using the ports on the peripheral controller:
    5.1. Sending information to the ports
    5.2. Reading information from the ports
    5.3. Reading from and writing to individual port pins
6. C and Assembly:
    6.1. Using assembly code for a specific task (e.g. a timing loop)
    6.2. Using assembly code inside C programs
    6.3. Using conditional assembly

C compiler examinations

1. Testing operators:
    1.1. Arithmetic operators
    1.2. Increment and decrement operators
    1.3. Bit wise AND, OR and exclusive OR operators
    1.4. ones complement
    1.5. Left-shift and Right-shift operators
    1.6. Relational operators
    1.7. Logical operators
2. Program looping:
    2.1. Using the FOR statement
    2.2. The infinite loop
    2.3. Using the WHILE statement and DO-WHILE construction
3. Program Branching:
    3.1. Using the IF statement
    3.2. Constructions with the IF-ELSE statements
    3.3. The SWITCH statement
    3.4. The GOTO statement
4. Arrays:
    4.1. Creating and initializing arrays
    4.2. Multidimensional arrays
5. Strings:
    5.1. Initializing strings
    5.2. Printing strings (using printf, puts, etc.)
6. Functions:
    6.1. Programs with multiple Functions
    6.2. Using Local, Static and Global variables
    6.3. Parameter passing between functions
    6.4. Recursive functions
7. Structures:
    7.1. Initializing structures
    7.2. Structures within structures
    7.3. Structures containing arrays
8. Pointers
    8.1. Basic pointer usage
    8.2. Passing arguments by value or by argument
    8.3. Pointers to Character Arrays

C Programs Summary

C_ASSBLY.C

This program demonstrates how Assembly language is implemented in C programs (in-line Assembly). The program first examines port P1 (LEDs are connected to this port) and determines if any bits have a logic 1 level. When a bit with logic 1 level is detected, the program executes code which drives P1 to a logic 0, then to logic 1 (LEDs blink off, then on). When bits of P1 are at the logic 0 level, the program executes code which drives P1 to logic 1, then to logic 0 (the LEDs blink on, then off). Once P1 is examined then toggled, the code executes an endless loop. The details of using Assembly in C are described in the *Programming Details* section and in the program C_ASSBLY.C.

INTERUPT.C

This program demonstrates how to enable and disable interrupts in C and shows how to implement interrupt service routines for external and internal interrupts. The program code also shows how to set interrupt priorities. In the program, an interrupt is generated by setting the timer and allowing it to time out. The program jumps to the interrupt service routine where LEDs are illuminated to verify that the interrupt service routine has executed successfully. While in the internal interrupt service routine, the program enters an infinite loop at which time the user can generate an external interrupt by use of an external interrupt source (as described in the Intel *8x931AA, 8x931HA Universal Serial Bus Peripheral Controller User's Manual* external interrupt section). Once the external interrupt is generated, the program goes into the external interrupt service routine and the LEDs are illuminated to verify the interrupt service routine has executed successfully. The interrupt vector numbers starting address is configured in the PLC target configuration window. The details of implementing interrupts in C are shown in the *Programming details* section and in the program INTERUPT.C.

PCA.C

This program demonstrates how to implement the Programmable Counter Array (PCA) in a C program. In the program, the PCA is set as a firmware timer. Executing the code causes the timer to time out which generates the PCA interrupt. The service routine for this interrupt causes the LEDs to illuminate which indicates the routine has executed correctly. The details of how to set up the PCA in C are shown in the program PCA.C.

PORTS.C

This program demonstrates how the 8x930xx ports can be accessed in a C program. The program uses structures to allow access to the ports and the individual port pins. The program reads the ports' status and prints it to the terminal. The value of port 0 is read and a new value is written to the port by external means. The new value of port 0 is then printed to the terminal. The details of accessing the ports are shown in the program PORTS.C.

SER_COM.C

This program shows how to use the serial communications port to transmit and receive data. The program transmits a simple message to the internal serial port. The message can be viewed with the Microsoft Windows* terminal application or any application that can allow connections through a PC's serial communication port. The program then enables the serial port interrupt, which allows the user to enter characters via the keyboard. When a character is entered it is incremented to the next character (e.g., "a" is incremented to "b") and then printed to the terminal. The details of using the serial communication port are shown in the ***Programming details*** section and in the program SER_COM.C.

SERIAL.C

Use this program to set the serial port up for communications with the terminal. The program has one function (no main function) which can be called by other programs to set up the internal serial port for communication, using timer 2 as the baud rate generator. Due to the speed required for the other serial port modes and the terminal limitations, other serial modes are not used. The details of setting up the serial port for communications are shown in the ***Programming details*** section and in the program SERIAL.C.

SFR_MEM.C

This program demonstrates one of two methods for storing to and reading from the special function registers (SFRs) to specific memory locations. This method uses pointers while the other method uses keywords included with the software for this purpose. In the program, ten pointers are assigned specific memory addresses. The SFRs and last five pointers are initialized. The contents of the SFRs used in the program are written to the first five pointers. The contents of the memory addresses specified by the last five pointers are placed in the SFRs that are used in the program. The program validates the values that are written and read to ensure they are correct. If the numbers are incorrect, the program enters an infinite loop which causes the program to stop executing. The details of using the pointers to write and read to specific memory locations is shown in the program SFR_MEM.C.

TIMER0.C

This program demonstrates the implementation of Timers in C programming. The user is prompted to select the desired timer mode. Once this selection is made, the timer begins to run in that mode. The timer times out, generating an interrupt. The interrupt service routine illuminates the LEDs to confirm that the routine has executed correctly. This program requires the use of the internal serial port connected to the serial port of a PC. The details of setting up Timer0 are shown in the program TIMER0.C.

TOOLS.C

This program demonstrates the implementation of strings, structures, switch and case statements as well as printing (printf) to and reading (scanf) from the terminal. The program requests information from the user — such as the user's name, company and USB HUB peripheral controller requirements — and stores the data in a structure. The program recommends to the user a USB HUB peripheral controller that suits their requirements using a switch and case statement. The program outputs to the terminal the user's name, company, device requirements and the recommended device. This program requires the use of the internal serial port connected to the serial port of a PC. The details of printing and reading strings to and from the terminal through the serial port are shown in the ***Programming details*** section and in the program TOOLS.C.

VAR_MEM.C

This program demonstrates the second of two methods for accessing specific memory locations. This method does not use pointers; it uses keywords included with the software for the use of storing data to known memory locations. The program:

- stores variables and values into specified memory locations.
- writes internal memory to external memory, and external memory to internal memory.
- stores a one-dimensional array of numbers and a two-dimensional array of numbers starting at specific addresses.

The LEDs illuminate to indicate that program execution has completed. Use the debugger to view the data memory to confirm that the correct data was written to the correct addresses. The details of accessing specific memory locations are shown in the ***Programming details*** section and in the program VAR_MEM.C.

WDT.C

This program demonstrates how the Watch Dog Timer (WDT) is configured in a C program. The program sets up the timer, which causes the timer to run. After an arbitrary number of refreshes the timer times out, which resets the device, thus resetting the evaluation board. The board does not reset until the program halts. Because the reset is not generated by the debugger, the software has no indication that the system has been reset. The only visible evidence that the system has been reset is the LED sequence associated with board reset. The details of setting up the Watch Dog Timer are shown in the program WDT.C.

Programming Details

Certain applications of C programming are specific to the Keil development tools. The correct implementation of certain functions or applications in C for the Keil software tools are described in the following subsections.

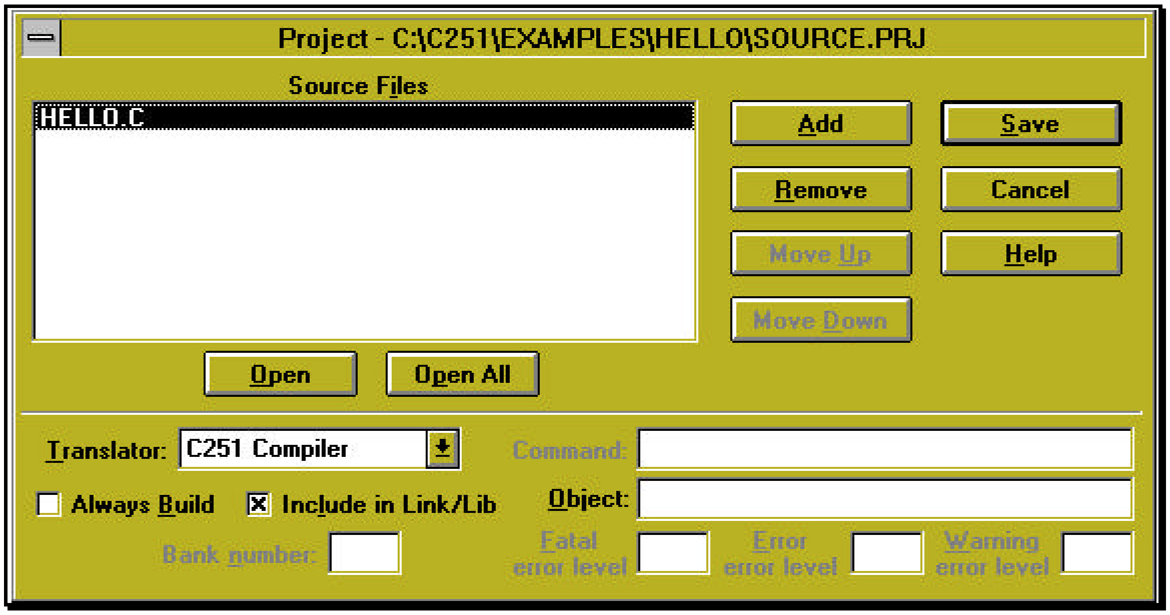
In - line Assembly

Command directives are used when writing in-line Assembly using Keil software tools. An Assembly source file (*.SRC) has to be generated from code that contains Assembly instructions. The SRC command directive must be used so that the compiler will generate an Assembly source file (*.SRC). The Assembly code is delineated by the ASM and ENDASM directives. The form is:

```
#src(filename.asm) or #src(filename.a251)

#pragma asm

        assembly code

#pragma endasm
```

After the code has been **compiled** the project must be edited so that the assembly source file can be assembled and then linked. The source file is changed from FILENAME.C to FILENAME.ASM (FILENAME.A251) and the translator must be changed to A251 Assembler as shown below in Figure 1.1 and Figure 1.2 Then the project must be **made** so that the file is assembled and linked.

Figure 1.1



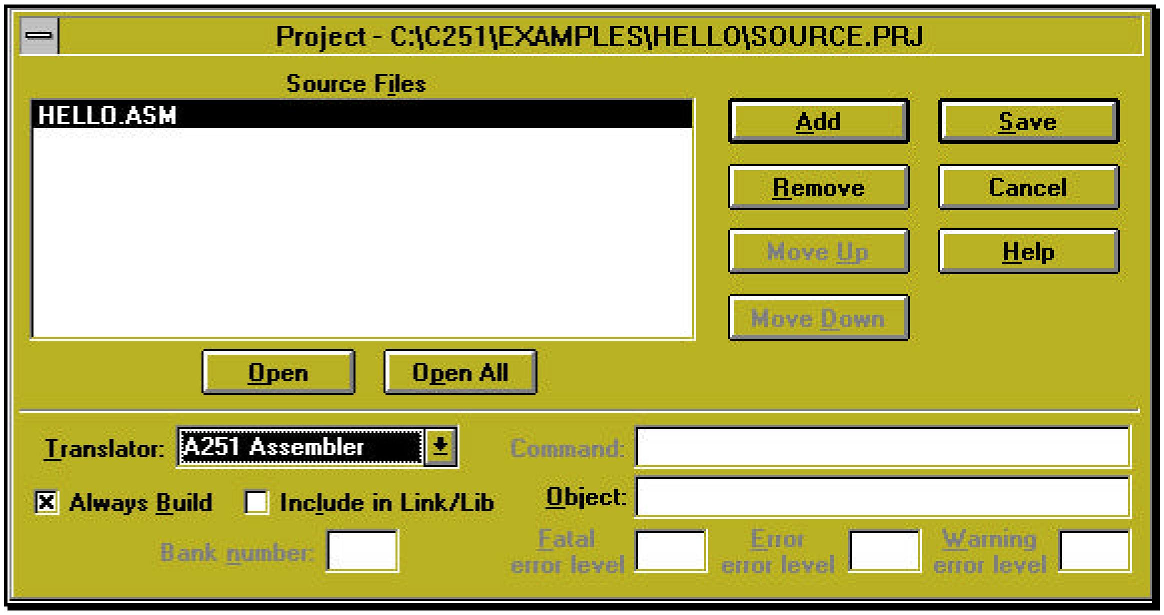
Project - C:\C251\EXAMPLES\HELLO\SOURCE.PRJ

Source Files: HELLO.C

Translator: C251 Compiler

[ ] Always Build [✓ Include in Link/Lib]

Figure 1.2



Project - C:\C251\EXAMPLES\HELLO\SOURCE.PRJ

Source Files: HELLO.ASM

Translator: A251 Assembler

[✓ Always Build] [ ] Include in Link/Lib

Interrupts

The INTERRUPT keyword tells the compiler that the function is an interrupt service routine. The USING keyword indicates that the service routine is using register bank 1. The form is:

```
void ISR(void) interrupt # using 1
{
      C code;
}
```

Serial Communications

Using the serial port for communications requires that specific registers contain specific values so the baud rate will be set correctly. The correct baud rate ensures the serial port transmits and receives correctly. The function used to set the baud rate provided in this document is called serialSetup in the SERIAL.C program. The SERIAL.C program uses timer 2 to generate the baud rate of 9600.

Serial Communications - output

Outputting data through the serial communications port to the terminal at a baud rate of 9600 requires that a delay be present in between each transmission of a character. The serial port interrupt bit must *not* be enabled for serial port output transmissions because an interrupt will be generated when data is transmitted. If no serial port interrupt service routine exists, the program may deliver unexpected results. An example of this follows:

```
char out[21];
short cnt, delayCnt;

out = “This how to output text”;

for (cnt = 0; cnt < 21; cnt++)
{
      SBUF = out[i];
for (delayCnt = 0; delayCnt < 7000; delayCnt++);
}
```

Serial Communications - input

Inputting data through the serial communications port to the terminal at a rate of 9600 baud requires that the serial port interrupt bit be enabled. Between reception and transmission, an instruction — such as writing a value to a port — must be inserted for delay purposes. An example of this follows:

```
char in;
in = SBUF;
RI = 0;
P1 = 0xF2;
in++;
SBUF = in;
TI = 0;
```

Serial Communications - printf, puts

Using the C functions **prints** and **puts** to output characters to the terminal is the same as described in the preceding section (Serial communications - output). The TI bit must be set (TI = 1) before these functions are called. An example of this follows:

```
char text[40];

text  = “This is how it is done with puts”

TI = 1;
printf(“This how it is done with printf”);
puts(text);
```

Serial Communications - scanf

Using the C function **scanf** to input characters from the terminal is different from just inputting characters through the serial port. The TI bit must be set (TI = 1) because this function prints back to the screen what the user has entered on the keyboard. The ES (serial port interrupt) bit must **not** be enabled. An example of this follows:

```
short input;

TI = 1;
scanf(“%d”, &input);
```

Memory Access

As is true for assembly language programming, specific memory addresses can be accessed in C other than with the use of pointers. The header ABSACC.H must be included in the code so that memory can be accessed. An example of this follows:

```
#include<absacc.h>

unsigned char mem;

DBYTE [0x20] = mem;     /*variable written to data memory*/
DBYTE [0x22] = 0x18;    /*number written to data memory*/
```

Variables, such as arrays, can be stored at specific memory locations using the **_at_** keyword. An example of this follows:

```
char idata table[index] _at_ 0x40
```

Char is the data type, idata is the memory specifier, table[index] is the variable and 0x40 represents the memory address where table[0] will be stored.

C programs

C_ASSBLY.C

```
/**********************************************************
 *This program is used to show how to implement in-line    *
 *assembly in C source code.  This program checks to see if*
 *any of the bits of the Port P1 are pulled high and makes *
 *a jump based on the port value.  The LEDs are illuminated*
 *to verify that the jumps were made to the correct        *
 *location.                                                *
 **********************************************************/


#include<Reg930.h>

/*This pragma is used to tell the compiler to generate a source file
  and name it c_assbly.asm.*/
#pragma src(c_assbly.asm)


void main(void)
{
      /*The LEDs are illuminated to show the start of execution.*/
      P1 = 0xF1;

      /*The beginning of the in-line Assembly code.*/
      #pragma asm
            MOV R3, P1
            CMP R3, #00H      ;The LEDs are checked to determine if they
are illuminated.
            JE LIGHTS         ;If the LEDs are not illuminated then jump
to LIGHTS.
            JMP NOLIGHTS      ;If the LEDs are illuminated jump to
NOLIGHTS.

      LIGHTS:                 ;The LEDs are turned on and then turned
off.
            MOV WR8, #2EFFH
      LP1:
            MOV P1, #0FFH     ;A delay is inserted so that the LEDs can
be seen.
            CMP WR8, #00H
            DEC WR8, #01H
            JNE LP1
            MOV P1,#00H
            JMP FINISH
      NOLIGHTS:           ;The LEDs are turned off then turned on.
            MOV WR8, #2EFFH
      LP2:
            MOV P1, #00H  ;A delay is inserted so LEDs can be seen.
            CMP WR8, #00H
            DEC WR8, #01H
            JNE LP2
            MOV P1, #0FFH
      FINISH:
      /*End of the in-line Assembly code.*/
      #pragma endasm

      while(1);
}
```

INTERUPT.C

```
/****************************************************
 *This program is used to show how interrupts are  *
 *set up in C as well as how to create functions   *
 *which are interrupt service routines.  The       *
 *altering of interrupt priority is also shown in  *
 *this program.                                    *
 *                                                 *
 *Timer 1 is set up as a 16 bit timer              *
 *and then allowed to time out with its interrupt  *
 *being enabled.  This generates an interrupt and  *
 *the code immediately jumps to the interrupt      *
 *vector where the service routine is located.     *
 *While in the timer service routine an external   *
 *interrupt is generated via the INT1# pin.  The   *
 *LEDs are illuminated to show the service routine *
 *has been executed properly.                      *
 *                                                 *
 *Interrupt name             Interrupt vector      *
 *    PCA                          0x0033          *
 *    Timer 2                      0x002B          *
 *    Serial Port                  0x0023          *
 *    Timer 1                      0x001B          *
 *    INT1#                        0x0013          *
 *    Timer 0                      0x000B          *
 *    INT0 #                       0x0003          *
 *                                                 *
 ***************************************************/

#include <Reg930.h>
#define DELAY 20000

void main(void)
{
```

```
/*The port 3 bit 3 is set so that the pin can function as the external
interrupt 1 pin.*/
      P3_3 = 1;

/*TMOD register is used to set Timer 1 as mode 1 -> 16 bit timer.*/
      TMOD = 0x01;

      /*Timer 1 interrupt is enabled and the timer is turned on.*/
      ET1 = 1;
      TR1 = 1;
      while(1);
}


/*Internal interrupt (Timer 1) service routine.*/
void timer1ISR(void) interrupt 3 using 1
{
      int cnt;

      /*Timer 1 is turned off.*/
      TCON &= 0xBF;;

/*The LEDs are turned on with delays so there will be visible proof that
the ISR was executed.*/
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x0F;
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x70;
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x55;

      /*The external interrupt priority is made high*/
      IPH0 |= 0x04;
      IPL0 |= 0x00;

      /*External interrupt is set to level triggered.*/
      TCON &= 0xFB;

      /*External interrupt 1 is enabled*/
      EX1 = 1;

      while(1);
}


/*External interrupt 1 service routine.*/
void ex_inter_isr(void) interrupt 2 using 1
{
      int cnt;

      /*The interrupt request is cleared.*/
      IE1 = 0;
      /*The LEDs are written to so that there will be visible proof that
      the interrupt service routine was executed*/
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x01;
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x02;
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x04;
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x04;
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x08;
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x10;
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x20;
      for(cnt = 0; cnt < DELAY; cnt++)
            P1 = 0x40;
      while (1);
}
```

PCA.C

```
/**************************************************************
 *The programmable counter array performs a variety of timing*
 *and counting operations, including pulse width modulation  *
 *and also provides the capability of a firmware watchdog    *
 *timer.  The PCA is setup in the following code as a        *
 *firmware timer.  A value is placed in the compare register  *
 *and when the timer reaches that value an interrupt is      *
 *generated and the code jumps to the interrupt service     *
 *routine for the PCA.                                       *
 *************************************************************/
#include<Reg930.h>
#define DELAY 10000

void main(void)
{
      /*PCA compare/capture register is configured as a firmware timer
      with compare flag enabled. */
      CCAPM0 = 0x49;

/*PCA Timer/Counter mode register set the input signal as Fclk/6.*/
      CMOD = 0x80;

      /*The PCA Timer/Counter interrupt is disabled. */
      EC = 0;

/*An interrupt is generated when the timer reaches the value in the
compare registers CCAPxH,CCAPxL.*/
      CCAP0L = 0x11;
      CCAP0H = 0xCF;
/*The PCA timer/counter interrupt is enabled and the timer is turned
on.*/
      EC = 1;
      CR = 1;
      while (1);
}

/*Interrupt service routine for the 16 bit firmware timer.*/
void pcaISR(void) interrupt 6 using 2
{
      long cnt;

      /*The timer is turned off and the interrupt request is cleared.*/
      CCON = 0x00;

      /*The LEDs are written so that there will be visible proof
      that the interrupt service routine was executed.*/
      for(cnt = 0; cnt <= DELAY; cnt++)
            P1 = 0x01;
      for(cnt = 0; cnt <= DELAY; cnt++)
            P1 = 0x02;
      for(cnt = 0; cnt <= DELAY; cnt++)
            P1 = 0x04;
      for(cnt = 0; cnt <= DELAY; cnt++)
            P1 = 0x08;
      while(1);
}
```

PORTS.C

```
/************************************************************
 *This program is used to read and write to some of the     *
 *ports of the device and to output a text string through   *
 *the serial port to a terminal.  Individual port pins will *
 *be accessed as well.  The port status is read and printed.*
 ************************************************************/

#include<Reg930.h>
#include<stdio.h>
#include<string.h>

/*Global variables that are used to identify the ports.*/
char* portName0;
char* portName2;

/*Reference to the function that is used to set the serial port up.*/
extern void serialSetup(void);
```

```
/*Structure which gives the port name, current value, port number and
each of the
port bits.*/
typedef struct
{
      char* port;
      short portStatus;
      short portNumber;
      short bit0:1;
      short bit1:1;
      short bit2:1;
      short bit3:1;
      short bit4:1;
      short bit5:1;
      short bit6:1;
      short bit7:1;
}ports;

/*Function prototype*/
void readBit(ports portsBit);

void main(void)
{
      ports portP0, portP2;

/*Call to external function which is used to set up the internal serial
port at a baud rate of 9600.*/
      serialSetup();

      /*Port P0 status is defined.*/
      portP0.port = "P0";
      P0 = 0x00;
      portP0.portStatus = P0;
      portP0.portNumber = 0x00;
      portP0.bit0 = P0_0;
      portP0.bit1 = P0_1;
      portP0.bit2 = P0_2;
      portP0.bit3 = P0_3;
      portP0.bit4 = P0_4;
      portP0.bit5 = P0_5;
      portP0.bit6 = P0_6;
      portP0.bit7 = P0_7;

      printf("\n\n%s, the port status is %#x\n", portP0.port,
portP0.portStatus);
      readBit(portP0);
      portP0.portStatus = P0;
      printf("%s, the port status is now %#x\n", portP0.port,
portP0.portStatus);


      /*Port P2 status is defined.*/
      portP2.port = "P2";
      portP2.portStatus = P2;
      P2_0 = 1;
      portP2.portNumber = 0x02;
      portP2.bit0 = P2_0;
      portP2.bit1 = P2_1;
      portP2.bit2 = P2_2;
      portP2.bit3 = P2_3;
      portP2.bit4 = P2_4;
      portP2.bit5 = P2_5;
      portP2.bit6 = P2_6;
      portP2.bit7 = P2_7;

      printf("\n%s, the port status is %#x\n", portP2.port,
portP2.portStatus);
      readBit(portP2);

      P1 = 0x0F;
      while(1);
}

/*The function that is used to read the value of the port and then allow
the user the opportunity to change a port bit.*/
void readBit(ports portsBit)
{
    short oldValue0, oldValue2;

      oldValue0 = P0;
      oldValue2 = P2;
      portName0 = "P0";
      portName2 = "P2";


      if (!strncmp(portsBit.port, portName0, 2))
      {
            while(P0 == oldValue0);
      }
}
```

SER_COM.C

```
/*******************************************************
 *The serial port which is used to communicate with the*
 *outside world can be set up in different modes.      *
 *There is one synchronous mode and three asynchronous *
 *modes.                                               *
 *MODE 0 = half duplex, synchronous mode used to expand*
 *         the capability of a I/O device with shift   *
 *         registers; Baud rate = Fclk/6               *
 *MODE 1 = full duplex, asynchronous mode;             *
 *         Baud rate = variable                        *
 *MODE 2 = full duplex, asynchronous mode;             *
 *         Baud rate = Fclk/16 or Fclk/32              *
 *MODE 3 = full duplex, asynchronous mode;             *
 *         Baud rate = variable                        *
 *******************************************************/
```

```
/*****************************************************************
 *This is program is used to test the serial communications of the*
 *internal serial port.  The serial port is set up by the SERIAL.C*
 *program which has a function serialSetup which is called and    *
 *sets the necessary baud rate using timer 2.  A message is       *
 *printed to the terminal and characters are read in from the     *
 *keyboard and printed out again.                                 *
 *****************************************************************/

#include<Reg930.h>
#include<stdio.h>
#include<string.h>
#define DELAY 7000

/*Reference to the function that is used to set the serial port up.*/
extern void serialSetup(void);

/*Function prototype*/
void serialPrint(char* outputText, short textLength);

void main(void)
{
      char* output;
      char* output1;
      char* output2;
      short length, length1, length2;

    /*Character pointers are initialized.*/
      output = "\r\nThis is a test, just a test.\r\n";
      length = strlen(output);
      output1 = "\r\nThis is cool.\r\n";
      length1 = strlen(output1);
      output2 = "\r\nThis is fun.\r\n";
      length2 = strlen(output2);

/*Call to external function which is used to set up the internal serial
port at baud rate of 9600.*/
      serialSetup();

/*Call to the function which output text through the internal serial
port.*/
      serialPrint(output, length);
      serialPrint(output1, length1);
      serialPrint(output2, length2);

/*Serial port interrupt is enabled to allow the user to input data.*/
      ES = 1;
/*A small delay is required after the serial port is enabled and so the
LEDs are illuminated.*/
      P1 = 0xf1;
      while (1);

}
```

```
/*Interrupt service routine for the serial port input communications.*/
void internalSerialISR(void) interrupt 4 using 1
{
/*If there has been a reception as is indicated by the RI bit then this
part is executed.  The character from the keyboard is received and then
incremented and then printed back out.*/
      if (RI == 1)
      {
            P1 = 0x0F;
            ACC = SBUF;
            RI = 0;
/*A small delay is inserted by illuminating the LEDs to allow the serial
port enough time.*/
            P1 = ACC;
            ACC++;
            SBUF = ACC;
            TI = 0;
      }
/*If there has only been a transmission then this part is executed.*/
      else
            TI = 0;
            return;
}

/*This is the function that is used to print to the terminal through the
internal serial port.  The text and the length of the text are pass to
the function*/
void serialPrint(char* outputText, short textLength)
{
      short cnt, delayCnt;

/*The static counter is used to count how many times this function is
counted.*/
      static unsigned short counter = 0;

      for (cnt = 0; cnt < textLength; cnt++)
      {
            SBUF = outputText[cnt];
            TI = 0;

            /*A delay is inserted for the serial port transmission.*/
            for (delayCnt = 0; delayCnt < DELAY; delayCnt++);
      }
/*The number of times that the function has been called is printed.*/
      counter++;
      TI = 1;
      printf("\nThis function has been called %d times.\n", counter);
      return;
}
```

SERIAL.C

```
/*********************************************************
 *This program is used for setting up the serial port for *
 *communication.  The program uses timer 2 to generate the*
 *baud rate of 9600.  An external program calls the       *
 *serialSetup function to set up the baud rate generator  *
 * which is timer 2.                                      *
 *********************************************************/
#include<Reg930.h>
#include <stdio.h>


void serialSetup(void)
{
/*Timer 2 is disabled and set up as the baud rate generator for a 9600
baud rate with fclk = 12Mhz.*/
      TR2 = 0;

      /*Timer 2 Control register is set up for timer 2 operation.*/
      T2CON = 0x30;

      /*Timer 2 Reload/Capture registers set the baud rate for 9600.*/
      RCAP2H = 0xFF;
      RCAP2L = 0xEC;

/*Power control register enables the low clock mode, so that the clock
is 3 MHz.*/
      PCON |= 0x20;

      /*Timer 2 is enabled.*/
      TR2 = 1;

/*The serial port control register is set up for mode 1 operation*/
      SCON = 0x50;

/*Serial port interrupt priority is set to 1 which is next to the
lowest.*/
      PS = 1;

      /*The TI bit is set to allow for transmission.*/
      TI = 1;

      return;
}
```

SFR_MEM.C

```
/*********************************************************
 *This program is written to show access of the special   *
 *function registers by memory.  The values in the        *
 *registers and memory locations are checked for accuracy.*
 *If the program executes correctly then the last four    *
 *LEDs are illuminated. If the program does not execute   *
 *correctly then the program will not proceed pass the    *
 *point where the error is located.                       *
 *********************************************************/

#include<Reg930.h>
#include<stdio.h>

void main(void)
{
            char* mem1;
            char* mem2;
            char* mem3;
            char* mem4;
            char* mem5;
            char* mem6;
            char* mem7;
            char* mem8;
            char* mem9;
            char* mem10;

            /*Memory pointers are assigned addresses, the pointers point
            to these addresses.*/
            mem1 = (char*)0x60;
            mem2 = (char*)0x61;
            mem3 = (char*)0x62;
            mem4 = (char*)0x63;
            mem5 = (char*)0x64;
            mem6 = (char*)0x65;
            mem7 = (char*)0x66;
            mem8 = (char*)0x67;
            mem9 = (char*)0x68;
            mem10 = (char*)0x69;

/*Special function registers are initialized with arbitrary values.*/

            /*The B register should have the value 0x04.*/
            B = 0x34;
            B &= 6;
            if (B != 0x04)
                  while(1);

            /*The CCAP0H register should have the value 0x89.*/
            CCAP0H = 0x89;
            CCAP0H |= 8;
            if (CCAP0H != 0x89)
                  while(1);
```

```
		/*The P1 register should have the value 0x42.*/
		P1 = 0x53;
		P1 ^= 0x17;
		if (P1 != 0x44)
			while(1);

		/*The DPH register should have the value 0x10.*/
		DPH = 0x11;
		DPH <<= 4;
		if (DPH != 0x10)
			while(1);

		/*The CH register should have the value 0x10.*/
		CH = 0x42;
		CH >>= 3;
		if (CH != 0x08)
			while(1);

		/*Memory addresses are initialized with arbitrary values.*/

		/*The address at mem6 should have the value 0xC7.*/
		*mem6 = 0x23;
		*mem6 *= 0x13;
		if (*mem6 != 0x99)
			while(1);

		/*The address at mem7 should have the value 0x05.*/
		*mem7 = 0x24;
		*mem7 /= 0x07;
		if (*mem7 != 0x05)
			while(1);

		/*The address at mem8 should have the value 0x41.*/
		*mem8 = 0x39;
		*mem8 += 0x08;
		if (*mem8 != 0x41)
			while(1);

		/*The address at mem9 should have the value 0x5C.*/
		*mem9 = 0x73;
		*mem9 -= 0x17;
		if (*mem9 != 0x5C)
			while(1);

		/*The address at mem10 should have the value 0x06.*/
		*mem10 = 0x0A;
		*mem10 %= 0x3;
		if (*mem10 != 0x01)
			while(1);

	/*Special function register contents are written to memory.*/

		/*The address at mem1 should have the value 0x10.*/
```

```
            B = 0x04;
            *mem1 = B << 2;
            if (*mem1 != 0x10)
                  while(1);

            /*The address at mem2 should have the value 0x43.*/
            *mem2 = CCAP0H >> 3;
            if (*mem2 != 0x11)
                  while(1);

            /*The address at mem3 should have the value 0x02.*/
            *mem3 = P1 & 0x32;
            if (*mem3 != 0x00)
                  while(1);

            /*The address at mem4 should have the value 0x99.*/
            *mem4 = DPH | 0x99;
            if (*mem4 != 0x99)
                  while(1);

            /*The address at mem5 should have the value 0x23.*/
            *mem5 = CH ^ 0x33;
            if (*mem5 != 0x3B)
                  while(1);

/*Memory contents are written to the special function registers.*/

            /*The B register should have the value 0xE3.*/
            B = *mem6 * 5;
            if (B != 0xFD)
                  while(1);

            /*The CCAP0H register should have the value 0x01.*/
            CCAP0H = *mem7 / 5;
            if (CCAP0H != 0x01)
                  while(1);

            /*The P1 register should have the value 0x45.*/
            P1 = *mem8 + 4;
            if (P1 != 0x45)
                  while(1);

            /*The DPH register should have the value 0x59.*/
            DPH = *mem9 - 3;
            if (DPH != 0x59)
                  while(1);

            /*The CH register should have the value 0x01.*/
            CH = *mem10 % 5;
            if (CH != 0x01)
                  while(1);
```

```
/*The LEDs will stay illuminated once the program is complete.*/
            P1 = 0x0F;
            while(1);
}
```

TIMER0.C

```
/************************************************************
 *This program sets up and test the modes of timer 0 by     *
 *prompting the user to enter which mode that they desire   *
 *timer 0 to be in, then timer 0 is set to that mode and    *
 *the timer 0 interrupt is enabled.  When an interrupt is   *
 *generated the interrupt service routine is executed.  The *
 *LEDs are illuminated to show that the program has executed*
 *correctly.                                                *
 ************************************************************/
#include<Reg930.h>
#include<stdio.h>
#define DELAY 10000

/*Reference to the function that is used to set the serial port up.*/
extern void serialSetup(void);

void main(void)
{
      float mode;

/*Call to external function which is used to set up the internal serial
port at a baud rate of 9600.*/
      serialSetup();

      /*Print out the user menu and get the user's input.*/
      puts("\n\nThe following timer modes may be selected for
TIMER0\n\n");
      puts("Mode 0 - 13 bit timer\r");
      puts("Mode 1 - 16 bit timer\r");
      puts("Mode 2 - 8 bit timer with Auto-reload\r");
      puts("Mode 3 - two 8 bit timers\n");
      printf("Enter the Mode number that you would like timer 0 to be
set as and press \nenter:");
      scanf("%f", &mode);
      printf("\nThe mode is %.2f\n", mode);

      /*The timers are turned off in the Timer control register*/
      TCON = 0x00;


      /*The mode of timer 0 is set in the timer mode register.*/
    /*Mode 0 -> 13 bit timers*/
      if (mode == 0)
      {
            TMOD = 0x00;
            goto setbits;
       }
```

```
     /*Mode 1 -> 16 bit timers*/
       else if (mode == 1)
    {
            TMOD = 0x01;
      goto setbits;
    }

      /*Mode 2 -> 8 bit timer with Auto-reload*/
    else if (mode == 2)•
    {
            TMOD = 0x02;
            goto setbits;
    }

    /*Mode 3 -> two 8 bit timers*/
      else if (mode == 3)
    {
            TMOD = 0x03;
        goto setbits;
    }

      /*Incorrect mode is selected.*/
      else
      {
            puts("\n\nYou have entered the wrong value, reset and start
again.");
            while(1);
      }

    /*The interrupt for timer 0 is enabled and timer 0 is turned on.*/
      setbits:
            ET0 = 1;
            TR0 = 1;
      while(1);

}

/*Interrupt service routine for timer 0, interrupt vector located
at 0x000B but actually 0x400B*/
void timer0Isr(void) interrupt 1 using 1
{
      short cnt;

      /*The interrupt is disabled.*/
      ET1 = 0;
    while (1)
      {
/*LEDs turned on to show that interrupt service routine was executed*/
            for (cnt = DELAY; cnt > 0; cnt--)
                  P1 = 0x40;
            for (cnt = DELAY; cnt > 0; cnt--)
                  P1 = 0x20;
            for (cnt = DELAY; cnt > 0; cnt--)
```

```
                  P1 = 0x10;
            for (cnt = DELAY; cnt > 0; cnt--)
                  P1 = 0x08;
      }
}
```

TOOLS.C

```
/*****************************************************
 *This program is used to examine how the software   *
 *interacts with stings and structures.  The program *
 *prompts the user to enter his/her name and then    *
 *enter the user's company as well as info about the *
 *USB device needed.  From this information the      *
 *program recommends to the user the USB controller  *
 *they should be using.                              *
 *****************************************************/
#include<Reg930.h>
#include<stdio.h>

/*Reference to the function that is used to set the serial port up.*/
extern void serialSetup(void);

/*Structure definition for the 8x930USB peripheral controller.*/
typedef struct
{
      short ports;
      char package[4];
      short romSize;
}device;

/*Structure definition for company, user and device.*/
typedef struct
{
      char company[20];
      char userName[20];
      device controller;
}profile;

void main(void)
{
      char* chip;
      profile userApp;

/*Call to external function which is used to set up the internal serial
port at a baud rate of 9600.*/
      serialSetup();

      /*Dialogue has the user enter company and device information.*/
      puts("\n\nThis is the USB Hub product selector menu.\n");
      printf("Enter your first name: ");
      scanf("%s", &userApp.userName);
      printf("\nEnter the company that you represent: ");
      scanf("%s", &userApp.company);
      puts("\nEnter the number of downstream ports you will be
needing");
      printf("(4 ports or 3 ports): ");
      scanf("%d", &userApp.controller.ports);
      printf("\nEnter the desired package for the controller (PLCC or
SDIP): ");
      scanf("%s", &userApp.controller.package);
      puts("\nEnter the desired ROM size.");
      printf("i.e. 0, 8 or 16 (kbytes): ");
      scanf("%d", &userApp.controller.romSize);

      /*The user's device is determined.*/
      if (userApp.controller.ports == 3)
      {
            switch (userApp.controller.romSize)
            {
                  case 0: chip = "80930HD";
                        break;
                  case 8: chip = "80930HD";
                        break;
                  case 16: chip = "8093HE";
                        break;
            }
      }
      else
      {
            switch (userApp.controller.romSize)
            {
                  case 0: chip = "80930HF";
                        break;
                  case 8: chip = "80930HF";
                        break;
                  case 16: chip = "8093HG";
                        break;
            }
      }

/*The results of the user's inquiry are printed to the terminal screen
through internal serial port.*/
      printf("\n%s, representing  %s.", userApp.userName,
userApp.company);
      printf("\nThe peripheral controller recommended for your
application is the %s\n", chip);
      printf("in the %s package with %d downstream ports and %d
kilobytes size ROM.", userApp.controller.package,
userApp.controller.ports, userApp.controller.romSize);
while(1);
}
```

VAR_MEM.C

```
/*********************************************************
 *This program is used to show the implementation of      *
 *variables being located in memory as well as tables in   *
 *memory.  Arbitrary values are written to a one          *
 *dimensional array and a two dimensional array.          *
 *                                                         *
 *The memory can be accessed using CBYTE, CWORD, DBYTE     *
 *DWORD, NBYTE, NWORD, XBYTE and XWORD.                   *
 *CBYTE and CWORD allow access to the program memory.      *
 *DBYTE and DWORD allow access to the internal data memory.*
 *NBYTE and NWORD allow access to the near memory.         *
 *XBYTE and XWORD allow access to the external data memory.*
 *Each memory type is accessed in the same manner:         *
 *        NBYTE[address] = value   (for byte access)       *
 *        DWORD[address] = value   (for word access)       *
 *        variable = CBYTE[address]                        *
 *        variable = XWORD[address]                        *
 *                                                         *
 *The absacc.h header file must be included in the program *
 *to allow memory access.                                  *
 **********************************************************/
#include<Reg930.h>
#include<absacc.h>
#define INDEX 0x09
#define INDEX2 0x03

/*Global variable declaration of two arrays and their starting
addresses.*/
char idata table[INDEX] _at_ 0x40;
char idata table2[INDEX][INDEX2] _at_ 0x4A;

void main(void)
{
      short cnt, cnt2;
      unsigned char mem;

      mem = 0x04;

/*The contents of the variable mem are located at address 0x20 in
internal data memory*/
      DBYTE [0x20] = mem;


      /*The value is written to internal data memory address 0x22.*/
      DBYTE [0x22] = 0x18;

      /*Copy internal memory to external memory*/
      XBYTE [0x20] = DBYTE [0x22];

      /*The value is written to external data memory address 0x22.*/
      XBYTE [0x22] = 0x41;
      /*Copy external memory to internal memory*/
      DBYTE [0x24] = XBYTE [0x22];


      /*One dimensional array is stored starting at data memory 0x40*/
      for (cnt = 0x00; cnt < INDEX; cnt++)
      {
            if (cnt == 0)
                  cnt++;
            table[cnt] = 0x04 * cnt;
      }


      /*Multi-dimensional array is stored starting at data memory 0x4A*/
      for (cnt = 0x00; cnt < INDEX; cnt++)
      {
            for (cnt2 = 0x00; cnt2 < INDEX2; cnt2++)
            {
                  table2[cnt][cnt2] = 0x02 + cnt2;
                  DBYTE [0x4A] = table2[cnt][cnt2];

            }
      }

/*LEDs indicate that the program is complete.*/
P1 = 0xF1;

while(1);
}
```

WDT.C

```
/*******************************************************
 *The Watchdog timer is used to reset the chip is it is*
 *allowed to time out.  It provides a means of recovery*
 *from routines that do not complete successfully.  The*
 *timer must be refreshed so that it does not time out.*
 *The following program sets up the Watchdog timer and *
 *then after ten refreshes, allows it to time          *
 *out, which resets the system once the program is     *
 *halted.                                              *
 *******************************************************/
#include<Reg930.h>
#define DELAY 100

void main(void)
{
      short delayCnt, cnt;

      cnt = 0;
      ACC = 0x42;

      /*First this value is written*/
      WDTRST = 0x1E;
/*Second this value is written, these values clear and enable the WDT
and must be written to the periodically to the register before it
overflows to start it again.*/
      WDTRST = 0xE1;
/*The Watch Dog timer is reset 6 times before being allowed to time out
and reset the system.*/
      do
      {
            WDTRST = 0x1E;
            WDTRST = 0xE1;
            for (delayCnt = 0; delayCnt < DELAY; delayCnt++);
            cnt++;
      }
      while(cnt < 6);
   while(1);
 }
```