538 Lecture Notes Week 7

Embedded System Programming in C

- C, developed to write UNIX (by the people who invented UNIX), is arguably the best high-level language to write code “close” to the machine. (C can be considered a “high-level assembler”).
- In the embedded world, features of C not commonly used in elementary programming are exploited.
- These include:
    - Ways to access defined memory addresses.
    - Extensive use of #define and typedefs.
    - Bit-wise operators. (&, ~, | and ^).
    - Pragmas and other non-standard language extensions.
- Furthermore, understanding the way parameters and passed in C make is possible to write some of the software in C and some in assembler.

How to access memory locations

- Usually, you are not interested in the actual memory location of variables.
- But you can determine these addresses.
- Consider the following example (created with Netbeans and produces code that will compile and work on Windows, MacOS or Linux).

```
/*
 * File:   main.c
 * Author: Ken
 *
 * Created on October 11, 2013, 9:05 AM
 */

#include <stdio.h>
#include <stdlib.h>
```

```
int globalOne;
int globalTwo;

int foo(int);

int main(int argc, char** argv) {
    globalOne = foo(5);
    return (EXIT_SUCCESS);
}

int foo(int param) {
    int localOne, localTwo;
    printf("globalOne: %08x\n", &globalOne);
    printf("globalTwo: %08x\n", &globalTwo);
    printf("localOne: %08x\n", &localOne);
    printf("localTwo: %08x\n", &localTwo);
    printf("param: %08x\n", &param);
    globalOne = 5;
    printf("sizeof ptr: %d\n", sizeof(&param));
    printf("sizeof int: %d\n", sizeof localOne);
    long lng = (long)&globalOne;
    long lng2 = 0x004061c0L;
    printf("lng diff: %d\n", lng - lng2);
    char * cp;
    cp = (char *) (lng + 1);
    *cp = 2;
    printf("globalOne: %d\n", globalOne);

    return 0;
}
```

The output is:

```
globalOne: 004061c0
globalTwo: 004061c4
localOne: 0023aa84
localTwo: 0023aa80
param: 0023aab0
sizeof ptr: 8
sizeof int: 4
lng diff: 0
globalOne: 517
```

Using Timer Overflow without Interrupts

- We now switch to the Codewarrior IDE and the hcs12 processor.
- We will write C code with the small memory model and no device initialization
- In the following examples, we will sometimes look at the generated assembler by looking at the "listing file".
- To generate a listing file:
    - Click on "Standard Settings".
    - Click on "Compiler for HC12"
    - Click on "Options"
    - In popup window, click on "Output" tab.
    - Click box "Generate Listing File"
- Programming devices requires access to the memory mapped device registers.
- The C programmer (like the assembly language programmer) needs to read the data sheets for devices to obtain information about the registers and the meanings of all bits.
- Below is a description of one of the Timer's control registers.

Module Base + 0x0006

	7	6	5	4	3	2	1	0
R	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
W					▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 15-12. Timer System Control Register 1 (TSCR1)

Read: Anytime

Write: Anytime

Table 15-7. TSCR1 Field Descriptions

Field	Description
7 TEN	**Timer Enable** 0 Disables the main timer, including the counter. Can be used for reducing power consumption. 1 Allows the timer to function normally. If for any reason the timer is not active, there is no ÷64 clock for the pulse accumulator because the ÷64 is generated by the timer prescaler.
6 TSWAI	**Timer Module Stops While in Wait** 0 Allows the timer module to continue running during wait. 1 Disables the timer module when the MCU is in the wait mode. Timer interrupts cannot be used to get the MCU out of wait. TSWAI also affects pulse accumulator.

Here's our first try at programming this device to increment a counter each time a timer overflow occurs.

```
unsigned char tovCount;
void main(void) {
   tovCount = 0;
    //Enable Timer
  *((char *) 0x46) = 0x80;
  for(;;) {
     while (((*((char *) 0x4f)) & 0x80) == 0)
            ;
     *((char *) 0x4f)  = 0x80;
     tovCount++;
  } /* loop forever */
}
```

- By using #defines, we can make the code easier to write and understand as follows:

```
  #define TSCR1 *((char *) 0x46)
  #define TFLG2 *((char *) 0x4f)
  #define TOV_MASK 0x80

  //Global
  unsigned char tovCount;

  void main(void) {
    tovCount = 0;
      //Enable Timer
    TSCR1 = TOV_MASK;
    for(;;) {
       while ((TFLG2 & TOV_MASK) == 0)
              ; //Wait till Timer Overflow
       TFLG2  = TOV_MASK;   //Clear TOV flag
       tovCount++;
    } /* loop forever */
  }
```

- Next, we can use more sophisticated typedef's, *unions* and *bitfields* as follows:

```
  typedef unsigned char byte;
  #pragma MESSAGE DISABLE C1106 /* WARNING C1106: Non-standard 
bitfield type */
  #pragma OPTION ADD V30toV31Compatible "-
BfaGapLimitBits4294967295" /*this guarantee correct bitfield 
positions*/
  #define REG_BASE 0x0000                /* Base address for the 
I/O register block */

  /*** TSCR1 - Timer System Control Register1; 0x00000046 ***/
  typedef union {
    byte Byte;
    struct {
      byte             :1;
      byte             :1;
      byte             :1;
      byte             :1;
      byte TFFCA       :1;  /* Timer Fast Flag Clear All */
      byte TSFRZ       :1; /* Timer and Modulus Counter Stop While
in Freeze Mode */
      byte TSWAI       :1; /* Timer Module Stops While in Wait */
      byte TEN         :1;  /* Timer Enable */
    } Bits;
  } TSCR1STR;
  extern volatile TSCR1STR _TSCR1;
  #define TSCR1                              _TSCR1.Byte
  #define TSCR1_TFFCA                        _TSCR1.Bits.TFFCA
  #define TSCR1_TSFRZ                        _TSCR1.Bits.TSFRZ
  #define TSCR1_TSWAI                        _TSCR1.Bits.TSWAI
  #define TSCR1_TEN                          _TSCR1.Bits.TEN

  #define TFLG2 *((char *) 0x4f)
  #define TOV_MASK 0x80

  //Globals
  unsigned char tovCount;

  void main(void) {
    tovCount = 0;
   //Enable Timer
    //TSCR1 = 128;
    TSCR1_TEN = 1;
    for(;;) {
       while ((TFLG2 & TOV_MASK) == 0)
              ; //Wait till Timer Overflow
       TFLG2  = TOV_MASK;   //Clear TOV flag
       tovCount++;
    } /* loop forever */
  }
```

- Finally, we can include "derivative.h" to obtain:

```
  #include <hidef.h>      /* common defines and macros */
  #include "derivative.h"


  //Globals
  unsigned char tovCount;

  void main(void) {
    tovCount = 0;
```

```
   //Enable Timer
    TSCR1_TEN = 1;
    for(;;) {
       while (TFLG2_TOF == 0)
              ; //Wait till Timer Overflow
       TFLG2_TOF  = 1;   //Clear TOV flag
       tovCount++;
    } /* loop forever */
  }
```

Using Interrupts

```
  #include <hidef.h>      /* common defines and macros */
  #include "derivative.h"      /* derivative-specific definitions
*/

  byte tovCount;

  void main(void) {
      tovCount = 0;
      TSCR1_TEN = 1; //Enable Timer
      TSCR2_TOI = 1; //Enable Timer TOV interrupts
      asm("cli");  //Enable CPU interrupt recognition

      for(;;) {
      } /* loop forever */
  }

  //__interrupt void tofISR() {
  #pragma TRAP_PROC  //This "pragma" identifies the function
                     //as an Interrupt Service Routine
                     //i.e. it ends with "rti", not "rts"
     void tofISR() {
       TFLG2_TOF = 1;
       tovCount++;
      // asm("rti");
  }

  typedef void (*near tIsrFunc)(void);
  static const tIsrFunc _InterruptVectorTable[] @0xFFDEU = { /*
Interrupt vector table */
  /*lint -restore Enable MISRA rule (1.1) checking. */
    /* ISR name                                  No. Address Pri Name
Description */
    &tofISR,                          /* Timer Overflow vector */
  };
```

Other Timer Uses

```
  #include <hidef.h>      /* common defines and macros */
  #include "derivative.h"      /* derivative-specific definitions
*/
  void init(void);

  void main(void) {
      init();
      TSCR2_TOI = 1; //Enable Timer TOV interrupts
      asm("cli");  //Enable CPU interrupt recognition

      for(;;) {
      } /* loop forever */
  }

  #pragma TRAP_PROC  //This "pragma" identifies the function
                     //as an Interrupt Service Routine
                     //i.e. it ends with "rti", not "rts"
     void oc2ISR() {
       TFLG1_C2F = 1; //Acknowledge interrupt
       TC2 += 10000;  //service interrupt
  }

  void init() {
     TSCR1_TEN = 1; //Enable Timer
     //Use Channel 2 as Output Compare
      TIOS_IOS2 = 1;
     //Set OC2 count to 10,000
     TC2 = 10000;
     //Enable OC2
     TIE_C2I = 1;
  }

  typedef void (*near tIsrFunc)(void);
  /*lint -save  -e950 Disable MISRA rule (1.1) checking. */
  static const tIsrFunc _InterruptVectorTable[] @(0xFFDEU + 0x0d)
= { /* Interrupt vector table */
  /*lint -restore Enable MISRA rule (1.1) checking. */
    /* ISR name                               No. Address Pri Name
Description */
    &oc2ISR,                      /* Timer Overflow vector */
  };
```

Functions in C (parameter passing, locals)

```
  #include <hidef.h>      /* common defines and macros */
  #include "derivative.h"      /* derivative-specific definitions
*/


  int foo(int a, int b);

  void main(void) {
    int x;
    x = foo(3, 4);
    for(;;) {
    } /* loop forever */
  }

  int foo(int p, int q) {
     int x, y;
     x = p+1;
     y = p*q;
     return y+2;
  }
```

```
  *** EVALUATION ***
  ANSI-C/cC++ Compiler for HC12 V-5.0.41 Build 10203, Jul 23 2010
```

```
      1:  #include <hidef.h>      /* common defines and macros */
      2:  #include "derivative.h"      /* derivative-specific
definitions */
      3:
      4:
      5:  int foo(int a, int b);
      6:
      7:  void main(void) {
  *** EVALUATION ***

  Function: main
  Source  : C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage\Sources\main.c
  Options : -CPUHCS12 -D__ONLY_INIT_SP -D__NO_FLOAT__
-Env"GENPATH=C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage;C:\Users\Ken\Dropbox\courses\538
-
Fall2013\Projects\FunctionLinkage\bin;C:\Users\Ken\Dropbox\courses
\538-
Fall2013\Projects\FunctionLinkage\prm;C:\Users\Ken\Dropbox\courses
\538-
Fall2013\Projects\FunctionLinkage\cmd;C:\Users\Ken\Dropbox\courses
\538-Fall2013\Projects\FunctionLinkage\Sources;C:\Program Files
(x86)\Freescale\CWS12v5.1\lib\HC12c\lib;C:\Program Files
(x86)\Freescale\CWS12v5.1\lib\HC12c\src;C:\Program Files
(x86)\Freescale\CWS12v5.1\lib\HC12c\include"
-Env"LIBPATH=C:\Program Files
(x86)\Freescale\CWS12v5.1\lib\HC12c\include"
-EnvOBJPATH=C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage\bin
-EnvTEXTPATH=C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage\bin -Lasm=%n.lst -Ms
-ObjN=C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage\FunctionLinkage_Data\Standard\Ob
jectCode\main.c.o -WmsgSd1106

      8:    int x;
      9:    x = foo(3, 4);
    0000 c603         [1]     LDAB  #3
    0002 87           [1]     CLRA
    0003 3b           [2]     PSHD
    0004 52           [1]     INCB
    0005 160000       [4]     JSR   foo
    0008 1b82         [2]     LEAS  2,SP
     10:    for(;;) {
```

```
    000a 20fe          [3]     BRA   *+0 ;abs = 000a
     11:    } /* loop forever */
     12:  }
     13:
     14:  int foo(int p, int q) {
  *** EVALUATION ***

  Function: foo
  Source  : C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage\Sources\main.c
  Options : -CPUHCS12 -D__ONLY_INIT_SP -D__NO_FLOAT__
-Env"GENPATH=C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage;C:\Users\Ken\Dropbox\courses\538
-
Fall2013\Projects\FunctionLinkage\bin;C:\Users\Ken\Dropbox\courses
\538-
Fall2013\Projects\FunctionLinkage\prm;C:\Users\Ken\Dropbox\courses
\538-
Fall2013\Projects\FunctionLinkage\cmd;C:\Users\Ken\Dropbox\courses
\538-Fall2013\Projects\FunctionLinkage\Sources;C:\Program Files
(x86)\Freescale\CWS12v5.1\lib\HC12c\lib;C:\Program Files
(x86)\Freescale\CWS12v5.1\lib\HC12c\src;C:\Program Files
(x86)\Freescale\CWS12v5.1\lib\HC12c\include"
-Env"LIBPATH=C:\Program Files
(x86)\Freescale\CWS12v5.1\lib\HC12c\include"
-EnvOBJPATH=C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage\bin
-EnvTEXTPATH=C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage\bin -Lasm=%n.lst -Ms
-ObjN=C:\Users\Ken\Dropbox\courses\538-
Fall2013\Projects\FunctionLinkage\FunctionLinkage_Data\Standard\Ob
jectCode\main.c.o -WmsgSd1106

    0000 3b            [2]     PSHD
     15:    int x, y;
     16:    x = p+1;
     17:    y = p*q;
    0001 ec84          [3]     LDD   4,SP
    0003 ed80          [3]     LDY   0,SP
    0005 13            [3]     EMUL
     18:    return y+2;
    0006 c30002        [2]     ADDD  #2
     19:  }
    0009 30            [3]     PULX
    000a 3d            [5]     RTS
```

```
     20:
     21:
```

Bcd ascii