- **Comparison of C and Assembly**
- **How to compile a C program using CodeWarrior**
  - Using pointers to access contents of specific addresses in C
  - Including and using “derivative.h” or “hcs12.h” to use in MC9S12 port names
  - Software delays in C
  - Setting and clearing bits in C
  - Program to display a pattern on Dragon12 LEDs

A simple C program and how to compile it

Here is a simple C program

```
#define COUNT 5

unsigned int i;

main()
{
        i = COUNT;
        _ _asm(swi);
}
```

1. Start CodeWarrior and create a new project.

2. On the **Project Parameters** menu, leave the C box checked, give the project a name, and Set an appropriate directory.

3. On the **C/C++ Options** menu, select **ANSI startup code, Small memory model**, and **None** for floating point format. Then select Finish. This will open a new project for a C program.

4. Select **Edit – Standard Settings**. Select **Target – Compiler for HC12**, then click on **Options**. Click on the **Output** tab, and select the **Generate Listing File** option. Click **OK**, then **OK**.

5. C does not use an org statement to tell the compiler where to put code or data. CodeWarrior uses a linker file called *Project.prm*. You will have to edit this file to the compiler where to put your program and data. CodeWarrior has been set up to put your

program into Flash EEPROM starting at address *0xC000*. In this class, you will put your program into RAM starting at address *0x2000*, or into EEPROM starting at address *0x0400*. In the window which lists the project files, select **Project Settings – Linker Files – Project.prm**. Find the following line:

```
RAM = READ_WRITE 0x1000 TO 0x3FFF;
```

and change it to this:

```
RAM = READ_WRITE 0x1000 TO 0x1FFF;
PROG = READ_ONLY 0x2000 TO 0x3FFF;
```

Next, find the line

```
INTO ROM_C000      /*, ROM_4000*/;
```

and change it to

```
INTO PROG      /*, ROM_4000*/;
```

Save and close Project.prm.

6. In the window which lists the project files, double-click on main.c. Modify the file to look like this:

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

void main(void) {

}
```

7. Enter your C program.

8. Select **Project – Make**. This will create a *Project.abs.s19* file and a listing file *main.lst* in the bin directory. You will need to delete the first line (which starts with S0) from the Project.abs.s19 file.

9. If all went well, you should be able to download the Project.abs.s19 file into the MC9S12.

In the bin directory there will be several files with the *.lst* extension. The file *Start12.lst* contains C startup code. The file *main.lst* shows the assembly language which was produced by the C compiler.

The *Start12.lst* is fairly long, and it contains uncompiled code for a lot of things we do not use. Here are the portions of Start12.lst which we use. It just loads the stack pointer, initializes any needed global data, zeros out the rest of the global data, and calls the *main.c* code.

...

```
Function: _Startup
Source: C:\Users\Hector\Downloads ...
Options: _CPUHCS12 -D_NO_FLOAT__ -Env”GENPATH=C:\Users\...
...
399:   /* purpose:   1) initialize the stack
400:                 2) initialize the RAM, copy down init data etc (Init)
401:                 3) call main;
405:
406:    /* initialize the stack pointer */
...
```

```
0000  cf0000        [2]     LDS   #__SEG_END_SSTACK
...
460:   Init(); /* zero out, copy down, call constructors */
0003  0700          [4]     BSR   Init
...
469:   main();
0005  060000        [3]      JMP   main
470:   }
```

Here is the main.lst file.

```
*** 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:        #define COUNT 5
        4:
        5:        unsigned int i;
        6:
        7:        void main(void) {
*** EVALUATION ***

Function: main
Source : C:\Users\Hector\Downloads\...
Options : -CPUHCS12 -D__NO_FLOAT__ -Env”GENPATH=C:\Users\...

        8:
        9:                 i = COUNT;
0000 c605           [1]     LDAB #5
0002 87             [1]     CLRA
0003 7c0000         [3]     STD i
        10:                _ _asm(swi);
0006 3f
        11:
```

```
        12:     }
0007 3d             [5]     RTS
        13:
```

The file Project.map shows where various things will be put in memory. It is fairly long. Here are the relevant parts:

```
...
********************************************************************
STARTUP SECTION
--------------------------------------------------------------------
Entry point: 0x2029 (_Startup)
...

********************************************************************
SECTION-ALLOCATION SECTION
Section Name        Size   Type        From        To          Segment
-------------------------------------------------------------------------------------
.init                49    R           0x2000      0x2030      PROG
.startData           10    R           0x2031      0x203A      PROG
.text                 7    R           0x203B      0x2041      PROG
.copy                 2    R           0x2042      0x2043      PROG
.stack              256    R/W         0x1000       0x10FF     RAM
...
(long definition of memory-mapped registers)
...

MODULE:         -- main.c.o --
- PROCEDURES:
        main                    203B        7       7       1       .text
- VARIABLES:
        i                       1100        2       2       1       .common
MODULE:         -- Start12.c.o --
- PROCEDURES:
        Init                    2000         29     41      1       .init
```

```
        _Startup                2029        8       8       0       .init
- VARIABLES:
        _startupData            2031        6       6       3       .startData
- LABELS:
        __SEG_END_SSTACK        1100        0       0       1
...
```

This shows that the total program occupies addresses from 0x2000 to 0x2043. The stack occupies addresses from 0x1000 to 0x10FF. Our variable i is located at address 0x1100. The entry point to the program is at 0x2029.

This means that, to run the program, you need to tell DBug-12 to run the program from 0x2029, not from 0x2000:

g 2029

Pointers in C

• To access a memory location:

*address

• You need to tell compiler whether you want to access 8-bit or 16 bit number, signed or unsigned:

*(*type* *) address

– To read an eight-bit unsigned number from memory location 0x2000:

```
x = *(unsigned char *) 0x1000;
```

– To write a 0xaa55 to a sixteen-bit signed number at memory locations 0x1010 and 0x1011:

```
*(signed int *) 0x1010 = 0xaa55;
```

• If there is an address which is used a lot:

```
#define PORTB (* (unsigned char *) 0x0001)
...
x = PORTB;          // Read from address 0x0001
...
PORTB = 0x55;       // Write 0x55 to address 0x0001
```

• To access consecutive locations in memory, use a variable as a pointer:

```
unsigned char *ptr;

ptr = (unsigned char *)0x1000;
*ptr = 0xaa;        // Put 0xaa into address 0x1000
ptr = ptr+2;        // Point two further into table
x = *ptr;           // Read from address 0x1002
```

• To set aside ten locations for a table:

```
unsigned char table[10];
```

• Can access the third element in the table as:

table[2]

or as

*(table+2)

• To set up a table of constant data:

const unsigned char table[ ] = {0x00,0x01,0x03,0x07,0x0f};

This will tell the compiler to place the table of constant data with the program (which might be placed in EEPROM) instead of with regular data (which must be placed in RAM).

• There are a lot of registers (such as **PORTA** and **DDRA**) which you will use when programming in C. CodeWarrior includes the header **mc9s12dp256.h** which has all the registers predefined

Setting and Clearing Bits in C

• You often need to set or clear bits of a hardware register.

– The easiest way to set bits in C is to use the bitwise OR (|) operator:

DDRB = DDRB | 0x0F; // Make 4 LSB of Port B outputs

– The easiest way to clear bits in C is to use the bitwise AND (&) operator:

DDRP = DDRP & ~0xF0; // Make 4 MSB of Port P inputs

A software delay

• To enter a software delay, put in a nested loop, just like in assembly.

– Write a function **delay(num)** which will delay for *num milliseconds*

```
void delay(unsigned short num)
{
    volatile unsigned short i;   /* volatile so compiler*/
                                 /* does not optimize */
    while (num > 0){

        i = xxx;
                                 /* ------------------------------ */
        while (i > 0){           /* Want inner loop to delay   */
            i = i - 1;           /*  for 1ms                   */
        }                        /*------------------------------  */
        num = num - 1;
    }
}
```

• What should xxx be to make a 1 ms delay?

• Look at assembly listing generated by compiler:

```
                    19: void delay(unsigned short num)
                    20: {
                    0000 6cac    [2]     STD 4,-SP
                    21:     volatile unsigned short i;
                    22:
                    23:     while (num > 0)
                    0002 2015    [3]     BRA *+23    ;abs = 0019
                     24:    {
                     25:            i = D_1MS;
                    0004 cc0736          [2]     LDD #XXXX
                    0007 6c82    [2]     STD 2,SP
                    26:            while (i > 0)
                    0009 2005    [3]     BRA *+7 ;abs = 0010
                    27:            {
        Inner       28:                    i = i - 1;
outer   loop        000b ee82    [3]     LDX 2,SP
loop    takes       000d 09      [1]     DEX
        12 cycles   000e 6e82    [2]     STX 2,SP
                    0010 ec82    [3]     LDD 2,SP
                    0012 26f7    [3/1]   BNE *-7 ;abs = 000b
                    29:            }
                    30:            num = num - 1;
                    0014 ee80    [3]     LDX 0,SP
                     0016 09     [1]     DEX
                     0017 6e80   [2]     STX 0,SP
                    0019 ec80    [3]     LDD 0,SP
                     001b 26e7   [3/1]   BNE *-23 ;abs = 0004
                    31:     }
                    32: }
                    001d 1b84    [2]     LEAS 4,SP
                    001f 3d      [5]     RTS
```

• Inner loop takes 12 clock cycles.

• One millisecond takes 24,000 cycles
(24,000,000 cycles/sec × 1 millisecond = 24,000 cycles)

• Need to execute inner loop 24,000/12 = 2,000 times to delay for 1 millisecond

```
void delay(unsigned short num)
{
        volatile unsigned short i;  /* volatile so compiler */
                                    /* does not optimize */
        while (num > 0)
        {
                i = 2000;
                                    /* ------------------------------ */
                while (i > 0)       /*                                */
                {                   /* Inner loop takes 12 cycles     */
                        i = i - 1;  /* Execute 2000 times to          */
                }                   /* delay for 1 ms                 */
                                    /* ------------------------------ */
                num = num - 1;
        }
}
```

Program to increment LEDs connected to PORTB, and delay for 50 ms between changes

```
#include <hidef.h>           /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */
#define D_1MS (24000/12) // Inner loop takes 12 cycles
                         // Need 2,000 cycles for 1 ms

void delay(unsigned short num);
main()
{
      DDRB = 0xff; /  * Make PORTB output */
      PORTB = 0;      /* Start with all off */
      while(1) {
            PORTB = PORTB + 1;
            delay(50);
      }
}

void delay(unsigned short num)
{
      volatile unsigned short i;     /* volatile so compiler /*
                                     /* does not optimize */
      while (num > 0) {
            i = D_1MS;
            while (i > 0) {
                  i = i - 1;
            }
            num = num - 1;
      }
}
```

Program to display a particular pattern of lights on PORTB

```
#include <hidef.h>           /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */
#define D_1MS (24000/12)          // Inner loop takes 12 cycles
                                  // Need 2,000 cycles for 1 ms
void delay(unsigned short num);
main()
{
      const char table[] =    {0x80,0x40,0x20,0x10,
                               0x08,0x04,0x02,0x01};
      int i;
      DDRB = 0xff;            /* Make PORTB output */
      PORTB = 0;              /* Start with all off */
      i = 0;
      while(1)
      {
            PORTB = table[i];
            delay(100);
            i = i + 1;
            if (i >= sizeof(table)) i = 0;  /* Start over when */
                                            /* end is reached */
      }
}
```

Operators in C

Operator	Action	example
|	Bitwise OR	%00001010 | %01011111 = % 01011111
&	Bitwise AND	%00001010 & %01011111 = % 00001010
^	Bitwise XOR	%00001010 ^ %01011111 = % 01010101
~	Bitwise COMP	~%00000101 = %11111010
%	Modulo	10 % 8 = 2
||	Logical OR	%00000000 || %00100000 = 1
&&	Logical AND	%11000000 && %00000011 = 1
		%11000000 && %00000000 = 0

Setting and Clearing Bits in C

Assembly	C	action
bset DDRB,$0F	DDRB = DDRB | 0x0f;	Set 4 LSB of DDRB
bclr DDRB,$F0	DDRB = DDRB & ~0xf0;	Clear 4 MSB of DDRB
l1: brset PTB,$01,l1	while ((PTB & 0x01) == 0x01)	Wait until bit clear
l2: brclr PTB,$02,l2	while ((PTB & 0x02) == 0x00)	Wait until bit set

Pointers in C

To read a byte from memory location 0xE000:

```
var = *(char *) 0xE000;
```

To write a 16-bit word to memory location 0xE002:

```
*(int *) 0xE002 = var;
```

Program to count the number of negative numbers in an array in memory

```
/* Program to count the number of negative numbers in memory *
   Start at 0xE000, go through 0xEFFF
   Treat the numbers as 8-bit
*/
#include <hidef.h>          /* common defines and macros */
#include "derivative.h"     /* derivative-specific definitions */

unsigned short num_neg;     /* Make num_neg global so we can */
                            /* find it in memory use type int so */
                            /* can hold value larger than 256 */
                            /* Unsigned because number cannot */
                            /* be negative */
main()
{
        char *ptr,*start,*end;

        start = *(char *) 0xE000;     /* Address of first element */
        end = *(char *) 0xEFFF;       /* Address of last element */

        num_neg = 0;

        for (ptr = start; ptr <= end; ptr = ptr+1)
        {
                if (*ptr < 0) num_neg = num_neg + 1;
        }
        _ _asm(swi);                  /* Exit to DBug-12 */
}
```