# Lab 3 - Part 1
# C Language Programming, Interrupts and Timer Hardware

In this sequence of three labs, you will learn how to write simple C language programs for the MC9S12 microcontroller, and how to use interrupts and timers.

## Introduction and Objectives

The C programming language is used extensively in programming microprocessors. In this lab you will write some simple C programs which do the things you did in assembly language in Lab2. For example, Program 1 displays a counting pattern on the LEDs connected to Port B

## 1 Prelab

For the prelab write the program for Part 4 of this lab.

## 2 The Lab

1. Start a new project.
   (a) This time, in **Project Parameters**, select **C**.
   (b) On the **C/C++ Options** menu, select **ANSI startup code, Small memory model** and **None** for the floating point format.
   (c) Select **Edit - Standard Settings**, Select **Target - Compiler for HC12**, then click on **Options**. Click on the **Output** tab, and select the **Generate Listing File**. The listing file will be called `main.lst` in the bin subdirectory of the CodeWarrior project. The listing file includes the C statements as well as the assembly language which was generated.

   CodeWarrior uses a linker file called `Project.prm` that tells the compiler where to put the program and data. In the window which lists the project files, select **Project Settings - Linker Files - Project.prm**

   Find the following lines

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

and change it to

```
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`

2. Type in Program 1 and click **Project - Make**.

3. Look at the file `main.lst` and try to understand what it does. Note that there may be some things which do not make sense to you. At the very least, find the assembly language code which increments Port B.

   CodeWarrior generates a file `Project.map` in the bin subdirectory. The file Project.map shows the addresses of the C functions and of any global variables. The `Project.map` file also shows entry point to the program and the sizes of the functions (in both hex and decimal) in the `OBJECT-ALLOCATION SECTION`.

   Load the file Project.abs.s19 into your MC9S12 and run it. Verify that the LEDs increments. Where is the entry point to the program in memory.

4. Using Program 1 as a model, write a C program to implement the program from Lab 2-Part 3.

   Compile and run your program. Have an instructor verify that it works.

5. Look at the `Project.map` and determine how many bytes the program takes (the length of the .text segment). Compare this to the length of last weeks program written in assembly.

6. Put your program in the EEPROM at address `0x0400`. Remember, when you put code into EEPROM you need to do some setup which DBug12 normally does for you. You need to convert the assembly-language code (which multiplies the clock by 6) from Lab2 into C code, and add it as the first lines of you program. There is no C statement to implement the assembly-language instruction `sei`. You can use the `__asm` function to insert this (or any other assembly language instruction) into you program:

   ```
   __asm(sei);
   ```

   You will want the array which stores the turn signal patterns into the EEPROM (so the array will not disappear when you turn off power). You will want variables which will change as the program is executed to be placed in RAM. You can tell the compiler to put constant data (such as an array of patterns to be display on LEDs) immediately following the code (so the data will be loaded into EEPROM) by defining the data as type `const`. An example of setting up an array of type `const` is:

   ```
   const char table[] = {0xaa,0xbb,0xcc};
   ```

   Now you need to tell the compiler to put the program into EEPROM. You can do that by using the `Project.prm` file as follows:

   Find the following lines

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

and change it to

```
INTO EEPROM;
```

Finally, change the address pointed to by EEPROM to `0x410`. Save and close `Project.prm`. When you upload the program and try to start it from EEPROM, it will try to start from address `0x0400`. To fix that add a `BRA 0x0439`. This is the address that your program starts at.

**Program 1** A C program to increment LEDs connected to Port B.

```
1   #include <hidef.h>           /* common defines and macros */
2   #include "derivative.h"      /* derivative-specific definitions */
3
4   #define D_1MS (24000/12)    // Inner loop takes 12 cycles
5                               // Need 24,000 cycles for 1 ms
6
7   void delay(unsigned short num);
8   main()
9   {
10      DDRB = 0xff;      /* Make PORTB output */
11      PORTB = 0;        /* Start with all off */
12      while(1)
13      {
14          PORTB = PORTB + 1;
15          delay(50);
16      }
17  }
18
19  void delay(unsigned short num)
20  {
21      volatile unsigned short i;   /* volatile so compiler does not optimize */
22
23      while (num > 0)
24      {
25          i = D_1MS;
26          while (i > 0)
27          {
28              i = i - 1;
29          }
30          num = num - 1;
31      }
32  }
```