SE 423 Mechatronics
Laboratory Assignment #1

Introduction to Programming the Robot Car's TMS320F28379D Processor.

One-Week Lab

Goals for this Lab Assignment

- Learn about the rules and responsibilities of access to the lab room, 3080ECEB, and its PCs.
- Start to become familiar with the TMS320F28379D LaunchPad and Code Composer Studio v12.
- Build and run your first microcontroller (DSP) program.
- Study the given starter code.

Laboratory Exercises

Exercise 1

Recommended Drive Usage:

Drive	Note
`C:`	You will be cloning your repository to the `C:\` drive below in Exercise 2. You will want to get in the habit of "committing" and "pushing" your newly developed code to your repository each time you leave the lab. We will be showing you how to use `git` for this purpose.
`N:`	This drive is read-only. There is a "scratch" directory on the `N:\` drive that does give you write permissions. Use `N:\scratch` only to transfer files from one account or PC to another, not to save work, because it is flushed periodically. Because you will be using your `git` repository heavily, you should not need to use this too often.
`U:`	When you log into the PCs in the Mechatronics Lab, the computer will automatically map to your own personal drive labeled "`U:`". This directory is accessible only with your login name and is your personal space to store your work (other than your repository), if you wish. I do not recommend cloning your repository to this drive due to issues with Code Composer Studio.

Exercise 2

First, follow the other Lab 1 document, "Using the SE423 Repository," and its first section, "Create your Repository," to check out the SE423 repository on your lab PC and/or your personal laptop. If you are going to use both the lab PC and your laptop for development, you will want to create the same path on both. Create a folder using your and your partner's NetIDs (do not use any spaces) at the root `C:\` drive and keep all your files there. It is also a good idea for you to make backups of your lab files outside of your Git repository as you progress through the semester. Making this extra backup will save you when we cannot figure out what went wrong with Git. Git is fantastic and typically works great, but every once in a while, I have seen issues that cause students to lose versions of their files. This is mainly because we are beginner Git users.

Exercise 3

1. Open Code Composer Studio 12 (CCS 12) and select the "workspace" folder in your repository. For example, if your netID were "`superstdnt`", you would have checked out your repository in `C:\superstdnt\SE423_repo`. The workspace folder to select would be `C:\superstdnt\SE423_repo\workspace`.

2. Once CCS 12 is done loading your workspace, you need to load the "LABstarter" project. When you perform this load, the project is copied into your workspace. Therefore, if you rename this project, you can reload the "LABstarter" project to start another minimal project. I purposely placed this "LABstarter" project in the same folder as many of the example projects you will be studying this semester. These examples are part of TI's software development stack, called C2000ware. I have copied the necessary parts of C2000ware into our repository so that if you accidentally modify something, you can easily restore it. Again, if your NetID was "`superstdnt`", perform the following to load your starter project. In CCS, select the menu Project→Import CCS Projects. Click "Browse" and explore to "`C:\superstdnt\SE423_repo\C2000Ware_4_01_00_00\device_support\f2837xd\examples\cpu1\LABstarter`" and finally press "Finish". Your project should then be loaded into the CCS environment. Let us then rename the project "`LAB1<yourinitials>`" by right-clicking on the project name "LABstarter". In the dialog box, change the name to "lab1". Also explore into the project and find the file "`LABstarter_main.c`". Right-Click on "`LABstarter_main.c`" and select "Rename". Change the name to "`lab1_main.c`". To save you some headaches in the future, I recommend you take one more step after creating a new "LABstarter" project. Right-click on your project name in the Project Explorer and select "Properties." Select "General" on the left-hand side. Then, in the "Project" tab, find the "Connection" drop-down and select "Texas Instruments XDS2xx USB Debug Probe." Apply and Close.

3. Now that you have the project loaded, you can build the code and download it to the LaunchPad board. Have your TA show you how to connect the robot to the USB of your PC, and then in CCS, hit the green "Debug" button, and in the dialog box that pops up, select CPU1 only. This will compile the code and load it to your LaunchPad board.

4. Click the Suspend button to pause the code and the Resume (or Play) button to resume or start your code's execution. Use both of these to prove to yourself that both the blue and red LEDs are blinking on and off. Use one more button in CCS 12, the Restart button. Run your code, then press the pause button to stop it. If you hit the Resume button, the code resumes from where you left off. If you press the Restart button, CCS will take you back to the beginning of your code, and then you can click the Resume button to restart it. This saves time if you need to restart the same code. If you need to make changes to your code, Restart does not download the new code. You have to re-Debug your code to get the new changes downloaded to the processor.

5. Read through the `main()` function of your `lab1_main.c` file and see if you can find the function that changes the period of the timer functions. Change the period and see if the LEDs blink at a different rate.

Exercise 4

1. For this exercise, instead of using the robot, use your F28379D LaunchXL board (Red Board) by itself. Find a mini USB cable that connects to your F28379D board. Most of the time, when you use your F28379D board, you start with a new project using "HWStarter"; for this exercise, use "LABstarter" again to create a new project using the same steps as above. Rename this project "`printtest<yourinitials>`" and remember to rename the `LABstarter_main.c` file to `printtest_main.c`. Also, right-click this new project, select "Properties," and make sure the Connection is set to "Texas Instruments XDS100v2 USB Debug Probe."

2. Look at the code, and you will see that the functions `UART_printfLine` and `serial_printf` are called every time the variable `UARTPrint` is equal to one. How often is `UARTPrint` getting set to one? Show your answer to

your TA. `UART_printfLine` prints to the two lines of the robot's text LCD. Since we are not using the robot for this exercise, these functions write data to a serial port, but it is ignored since there is no LCD screen connected to your homework F28379D board. We are using only the F28379D board, not the robot, because we want to send and receive data via the USB cable to a serial terminal called Tera Term. Go ahead and uncomment the `serial_printf` line of code that is just above the `UART_printfLine` function calls. Build and run this program.

3. We need to figure out which COM port your USB-to-serial adapter is using. The easiest way to find this is to run "Device Manager" in Windows and find the "Ports" item. Under ports, find the COM number for the device titled "XDS100 Class USB Serial Port". Run Tera Term and select the "Serial" item and find the XDS100 Class USB Serial Port in the list of COM ports. The final step is to change the Baud (or Bit) rate of the COM port. Still in Tera Term, select the "Setup" menu item, then "Serial Port...". Change the "Speed" to 115200 if it is not already.

4. Now you are ready to "bug/debug" your code in Code Composer Studio. Download and run your code, and you should see text printed in the serial terminal.

5. Also click (or give focus) to Tera Term and type some text into the terminal. The typed text will not be shown in the window, but those characters are being sent to the F28379D. Notice that when you type, the number of characters received increases (which value is printed to the terminal)? In CCS and your Printtest project, find and open the file `F28379dSerial.c`. Towards the bottom of the file, find the function `__interrupt void RXAINT_recv_ready(void)`. This function is called every time a character is sent across the serial port. The sent character is received in the local variable "RXAdata". Write an if statement that checks if the character in "RXAdata" is equal to the character 'a'. If it is 'a' turn on Board Red LED. Write another if statement that checks if "RXAdata" is equal to 'b'. If it is 'b', turn off the Board Red LED. Use the command `GpioDataRegs.GPBSET.bit.GPIO34 = 1;` to turn OFF the Red Board LED. Use the command `GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;` to turn ON the Red Board LED. Also, make sure to find in `CPU_Timer0`'s interrupt function the line that toggles on and off GPIO34. Comment that line out so you can control the on/off of the RED LED by pressing the 'a' and 'b' keys. **Demo this working to your TA.**

6. In addition, write code here in the `RXAINT_recv_ready` interrupt function that looks for other pressed keys and, in the same way, turns on and off LEDS 1 through 4 on the robot's green breakout board. We do not have these LEDS on the F28379D Board, but your TA will show you the Pin Mux Guide and how to connect the Oscilloscope to the pins that control these LEDs, and you can watch them transition on the scope's screen.

   - LED1 is connected to GPIO22.
   - LED2 is connected to GPIO94.
   - LED3 is connected to GPIO95.
   - LED4 is connected to GPIO97.
   - LED5 is connected to GPIO111.

   The GPIO pins are controlled by registers starting with the label GPA, GPB, GPC, GPD, GPE, or GPF. GPA registers (i.e., GPASET, GPACLEAR, GPATOGGLE, GPADAT) control IO pins 0 through 31. GPB registers (i.e., GPBSET, GPBCLEAR, GPBTOGGLE, GPBDAT) control IO pins 32 through 63, etc. So which GP registers control 94, 95, 97, and 111?

   For LED5, add code in Timer 0's interrupt function so that LED5 is toggled on and off every 3rd time Timer0's interrupt function is called.

   Demo your program working to your TA.

Exercise 5

Below is an introduction to the starter code given to you for this semester's lab assignments. I used green for the code and black for my commentary. This listing omits a significant quantity of the initialization starter code to keep it shorter. Read through the code and commentary below and find the same sections of code in your project you created above. Ask your instructor to clarify any initial questions you have about the starter code. You will have many questions, of course, as this is the first time you are reading through this code. We will be going over this code many times in lecture and lab sessions.

```
//#############################################################################
// FILE:   LABstarter_main.c
//
// TITLE:  Lab Starter
//#############################################################################

// Included Files
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <limits.h>
#include "F28x_Project.h"
#include "driverlib.h"
#include "device.h"
#include "f28379dSerial.h"
#include "LEDPatterns.h"
#include "song.h"
#include "dsp.h"
#include "fpu32/fpu_rfft.h"

#define PI          3.1415926535897932384626433832795
#define TWOPI       6.283185307179586476925286766559
#define HALFPI      1.5707963267948966192313216916398


// Interrupt Service Routines predefinition
__interrupt void cpu_timer0_isr(void);
__interrupt void cpu_timer1_isr(void);
__interrupt void cpu_timer2_isr(void);
__interrupt void SWI_isr(void);

// Count variables
uint32_t numTimer0calls = 0;
uint32_t numSWIcalls = 0;
extern uint32_t numRXA;
uint16_t UARTPrint = 0;
uint16_t LEDdisplaynum = 0;
```

For these C exercises, I would like you to create any global variables or functions. Actually, the only functions we will create this semester will be global. We will never need to create a function inside another function, including not creating functions inside your `main()` function. You can, of course, put your global functions anywhere outside of other functions. It is just nice to have all your functions defined in one spot of your code so they are easier for you to find. The same goes for global variables.

```
void main(void)
{
    // PLL, WatchDog, enable Peripheral Clocks
    // This example function is found in the F2837xD_SysCtrl.c file.
    InitSysCtrl();

    InitGpio();

        // Blue LED on LaunchPad
    GPIO_SetupPinMux(31, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(31, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPASET.bit.GPIO31 = 1;

        // Red LED on LaunchPad
    GPIO_SetupPinMux(34, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(34, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPBSET.bit.GPIO34 = 1;
```

... Purposely left code out here in this listing because it is all initializations that we will discuss in future labs and not important here ...

```
    EALLOW;  // This is needed to write to EALLOW-protected registers
    PieVectTable.TIMER0_INT = &cpu_timer0_isr;
    PieVectTable.TIMER1_INT = &cpu_timer1_isr;
    PieVectTable.TIMER2_INT = &cpu_timer2_isr;
```

... Purposely left code out of listing ...

```
    // Configure CPU-Timer 0, 1, and 2 to interrupt every second:
    // 200MHz CPU Freq, 1 second Period (in uSeconds)
    ConfigCpuTimer(&CpuTimer0, 200, 10000);
    ConfigCpuTimer(&CpuTimer1, 200, 20000);
    ConfigCpuTimer(&CpuTimer2, 200, 40000);

   // Enable CpuTimer Interrupt bit TIE
    CpuTimer0Regs.TCR.all = 0x4000;
    CpuTimer1Regs.TCR.all = 0x4000;
    CpuTimer2Regs.TCR.all = 0x4000;

    init_serial(&SerialA,115200);
```

... Purposely left code out of listing ...

```
    // Enable global Interrupts and higher-priority real-time debug events
    EINT;  // Enable Global interrupt INTM
    ERTM;  // Enable Global real-time Interrupt DBGM

    // IDLE loop. Just sit and loop forever (optional):
    while(1)
    {
```

Look below in the timer 2's interrupt function, `cpu_timer2_isr()`, and you will see that `UARTPrint` is set to 1 every 50th time the timer 2's function is entered. So both the rate at which timer 2's function is called and this modulus 50 determine the rate at which the `serial_printf` function is called. `serial_printf` prints text to Tera Term or some other serial terminal program. Also note that after the `serial_printf` function call, `UARTPrint` is reset to 0. Think about why that is important. Add a comment to the `UARTPrint = 0;` line of code explaining why it must be set back to zero here to make this code work correctly. (Correctly means that the `serial_printf` function is called at a periodic rate.)

```
      if (UARTPrint == 1 ) {
```

For this exercise, you should put most of your written code here. In future labs, you will find that the code run here, in this continuous while loop "`while(1)`", is less important. It will be your lower-priority code, which does not have as strict timing requirements. Also, this is the only place in your code that you should call `UART_printfLine` (and `serial_printf` for this lab). `UART_printfLine` is somewhat of a large function, and depending on how many variables you print, it can take some time and is not deterministic.

```
        // Normally, on the Robot Car, we only use the below UART_printfLine functions to write to the
        // on board LCD screen. The serial_printf below is only used in lab 1 to print to a serial port.
        // terminal over a USB cable, as you will for your homework.
        //serial_printf(&SerialA,"Num Timer2:%ld Num SerialRX: %ld\r\n",CpuTimer2.InterruptCount,numRXA);

        //IMPORTANT!! %ld is for an int32_t. To print an int16_t, use %d
        UART_printfLine(1,"Timer2 Calls %ld",CpuTimer2.InterruptCount);
        UART_printfLine(2,"T0 %ld,T1 %ld",CpuTimer0.InterruptCount,CpuTimer1.InterruptCount);
        UARTPrint = 0;
      }
    }
}
```

Right now, consider the calling of this function, `cpu_timer0_isr()` "magic". (We will explain this in detail soon in the course.) "`cpu_timer0_isr()` is called every 10ms, without fail, in this starter code. (It actually "interrupts" the code running in the `main()` "`while(1)`" while loop.) In main() you can change the 10000 (microseconds) in the line of code "`ConfigCpuTimer(&CpuTimer0, 200, 10000);`" to have it be called at a different rate.

```
// cpu_timer0_isr - CPU Timer0 ISR
__interrupt void cpu_timer0_isr(void)
{
    CpuTimer0.InterruptCount++;

    numTimer0calls++;

    if ((numTimer0calls%5) == 0) {
        // Blink LaunchPad Red LED
            GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;
    }

    // Acknowledge this interrupt to receive more interrupts from group 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

Right now, consider the calling of this function, `cpu_timer2_isr()` "magic". (We will explain this in detail soon in the course.) "`cpu_timer2_isr()` is called every 40ms, without fail, in this starter code. (It actually "interrupts" the code running in the `main()` "`while(1)`" while loop.) In `main()` you can change the 40000 (microseconds) in the line of code "`ConfigCpuTimer(&CpuTimer2, 200, 40000);`" to have it be called at a different rate.

```
// cpu_timer2_isr CPU Timer2 ISR
__interrupt void cpu_timer2_isr(void)
{
    // Blink LaunchPad Blue LED
    GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;

    CpuTimer2.InterruptCount++;
```

Since "`CpuTimer2.InterruptCount`" increments by 1 each time in this function, the following if statement sets `UARTPrint` to 1 every 10th time into this function "`cpu_timer2_isr()`". The % in C is modulus. Modulus returns the remainder of the division operation. So 23 % 10 equals 3, 67 % 10 equals 7, etc.

```
    if ((CpuTimer2.InterruptCount % 10) == 0) {
        UARTPrint = 1;
    }
}
```

Lab Checkoff

1. Demonstrate to your TA that you have successfully created, built, and run your first DSP project.
2. Show your program that changed the period of the blink rate of the blue and red LED.
3. Show in lab that you can print text to the robot's text LCD and the PC's serial terminal. Also show that when you type text into the serial terminal, the number of characters received changes in the print line, and when you type 'a', the RED LED turns ON, and when you type 'b', the RED LED turns OFF.
4. Show your program that also turns on and off LEDS 1 through 5 (On the Oscilloscope).