SE423 Laboratory #1
Introduction to Programming the Robot Car's TMS320F28379D Processor

Goals:

1. Learn about the rules and responsibilities of access to the lab room, 302TB, and its PCs.
2. Start to become familiar with the TMS320F28379D LaunchPad and Code Composer Studio v12.
3. Build and run your first microcontroller (DSP) program.
4. Study the given starter code.

Exercise 1:

Recommended Drive Usage:

C: Drive — 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: Drive — This drive is a read only drive. 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 and not to save work because it is flushed periodically. Because you will be heavily using your "git" repository, you should not need to use this too often.

U: Drive — 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 only accessible by your login name and is your personal space to store your work (other than your repository) if you wish. I do not recommend that you clone your repository to this drive due to some 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 to your lab PC and/or your personal laptop. If you are going to be using both the lab PC and your laptop for development you will want to create the same path on both your laptop as on the lab PC. Create a folder using you 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 can't figure out what went wrong with Git. Git is awesome and normally works great but every once in a while I have seen issues that caused students to lose versions of their files. This is mainly due to us being beginner Git users.

Exercise 3:

- Open Code Composer Studio 12 (CCS 12) and select the "workspace" folder in your repository. For example, if your netID was "superstdnt", you would have checked out your repository in c:\superstdnt\LabRepo. The workspace folder then to select would be c:\superstdnt\LabRepo\workspace.
- 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 will be able to load the "LABstarter" project again if you would like to start another minimal project. I purposely located this "LABstarter" project in the same folder that has many of the example projects you will be studying this semester. These examples are part of the software development stack that TI calls C2000ware. I have copied the needed parts of C2000ware into our repository so that if you accidently modify something you can easily get it back. Again now 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\LabRepo\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's 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 perform one more step after you have created 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".
- 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 the pops up, select CPU1 only. This will compile the code and load it to your LaunchPad board.
- 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 and then press the pause button to stop you code. If you hit the Resume button the code starts up again from where you stopped it. 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 start your code again from the beginning. This saves time if you just 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.

- 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:

- Obviously since you just cloned the repository to the PC there will be no changes, but to show you how to check if I have made any changes performs the steps in "Using the SE423 Repository" section "Course File Updates". I may make changes to the class repository so each week when you come into lab it is a good idea to see if there are any changes by repeating these steps.
- For more practice create a new LABstarter project using the same steps as above. Rename this project something like "printtest<yourinitials>" and remember to rename the LABstarter_main.c file to say printtest_main.c or whatever you would like.
- Build and run this program. Look at the code and you will see that the function serial_printf is called every time the variable UARTPrint is equal to one. How often is UARTPrint getting set to one? **Show your answer to your TA**.
- Plug in the USB-A/B cable into your robot. We need to figure out what serial port COM number your USB serial port 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 "USB Serial Port". Run Tera Term and select the "Serial" item and find the USB Serial Port in the list of COM ports. Final thing to do is change the Baud (or Bite) rate of the COM port. Still in Tera Term select the menu item "Setup" and then "Serial Port…". Change the "Speed" to 115200 if it is not already.
- Now you are ready to "bug/debug" you code in Code Composer Studio. Download and run your code and you should see text printed in the serial terminal.
- 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. 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 Board Red LED. Use the command `GpioDataRegs.GPBSET.bit.GPIO34 = 1;` to turn OFF Board Red LED. Use the command `GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;` to turn ON Board Red 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.**

- 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.
    - 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 leaves out quite a bit of the initialization starter code to keep the listing shorter. Read though the below code and commentary 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          3.14159265358979323846264338327795
#define TWOPI       6.28318530717958647692528676655900
#define HALFPI      1.57079632679489661923132169163980


// 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, this is where I would like you to create any global variables or global functions. Actually, the only kind of functions we will create this semester will be global functions. We will never need to create a function inside another function and that includes 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 realtime 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 determines the rate at which the serial_printf function is called. serial_printf prints text to Tera Term or some other serial terminal program. Also notice that after the serial_printf function call, UARTPrint is set back 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, I would like you to put most of your written code here. In future labs you will find that code run here, in this continuous while loop "while(1)", is less important code. It will be your lower priority code that does not have as strict of timing. Also this is the only place in your code that you should call serial_printf. serial_printf is somewhat of a large function and depending on how many variables you print can take some time and is not deterministic.

```
        serial_printf(&SerialA,"Num Timer2:%ld Num SerialRX: %ld\r\n",CpuTimer2.InterruptCount,numRXA);
        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, this below 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 divide 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 ran 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 a 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' RED LED turns OFF.
4. Show your program that also turns on and off LEDS 1 through 5.
5. For your lab submission, create a subfolder in your class Box folder and name it "Lab1". Inside this folder submit:

   A "HowTo" document.

   - Explains the steps to create a new LABstarter project in Code Composer.
   - Steps to pull up TeraTerm and connect to the LaunchPads serial port.
   - Steps to clone, add, commit, push, pull your repository at github.com.