ME461 Laboratory Assignment 1
Introduction to TMS320F28379D GPIO Programming and Texas Instruments Code Composer Studio

Goals for this Lab Assignment:

1. Use CPU Timer to periodically perform desired procedures/code.
2. Work with port inputs and port outputs.
3. What to do with a compiler error.
4. Debugging your source code with Breakpoints and the Watch Window.

LED's Default GPIO Assignments:

LED1	GPIO22, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED2	GPIO52, Controlled with Registers GPBDAT, GPBSET, GPBCLEAR and GPBTOGGLE
LED3	GPIO67, Controlled with Registers GPCDAT, GPCSET, GPCCLEAR and GPCTOGGLE
LED4	GPIO94, Controlled with Registers GPCDAT, GPCSET, GPCCLEAR and GPCTOGGLE
LED5	GPIO95, Controlled with Registers GPCDAT, GPCSET, GPCCLEAR and GPCTOGGLE
LED6	GPIO97, Controlled with Registers GPDDAT, GPDSET, GPDCLEAR and GPDTOGGLE
LED7	GPIO111, Controlled with Registers GPDDAT, GPDSET, GPDCLEAR and GPDTOGGLE
LED8	GPIO130, Controlled with Registers GPEDAT, GPESET, GPECLEAR and GPETOGGLE
LED9	GPIO131, Controlled with Registers GPEDAT, GPESET, GPECLEAR and GPETOGGLE
LED10	GPIO4, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED11	GPIO5, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED12	GPIO6, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED13	GPIO7, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED14	GPIO8, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED15	GPIO9, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED16	GPIO24, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED17	GPIO25, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED18	GPIO26, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED19	GPIO27, Controlled with Registers GPADAT, GPASET, GPACLEAR and GPATOGGLE
LED20	GPIO60, Controlled with Registers GPBDAT, GPBSET, GPBCLEAR and GPBTOGGLE
LED21	GPIO61, Controlled with Registers GPBDAT, GPBSET, GPBCLEAR and GPBTOGGLE
LED22	GPIO157, Controlled with Registers GPEDAT, GPESET, GPECLEAR and GPETOGGLE
LED23	GPIO158, Controlled with Registers GPEDAT, GPESET, GPECLEAR and GPETOGGLE

Push Button's Default GPIO Assignments:

PB1	GPIO122, Read bit status with Register GPDDAT
PB2	GPIO123, Read bit status with Register GPDDAT
PB3	GPIO124, Read bit status with Register GPDDAT
PB4	GPIO125, Read bit status with Register GPDDAT

GPIO Register Use when GPIO pin set as Output: The GPIO Registers are 32 bit registers but we use unions and bitfields in the C/C++ programming language to control just one bit of the 32 bit register at a time. The ".all" part of the C/C++ union is the entire 32bit register. The ".bit.GPIO19" is just one bit in the 32 bit register. So these two lines of C code perform the same operation:

GpioDataRegs.GPASET.all = 0x00000800; //You have to think a bit with this code to know that bit 11 is being set.

GpioDataRegs.GPASET.bit.GPIO11 = 1; //This line of code is easier to understand that we are setting the 11th bit.

Register	Usage	Example

GP?DAT	GP?DAT.bit.GPIO? = 1, Sets that Pin High, 3.3V
	GP?DAT.bit.GPIO? = 0, Sets that Pin Low, 0V/GND
		GpioDataRegs.GPADAT.bit.GPIO19 = 1; Sets GPIO19 High/3.3V
		GpioDataRegs.GPADAT.bit.GPIO19 = 0; Sets GPIO19 Low/0V

GP?SET	GP?SET.bit.GPIO? = 1, Sets that Pin High, 3.3V
	GP?SET.bit.GPIO? = 0, Does Nothing
		GpioDataRegs.GPBSET.bit.GPIO37 = 1; Sets GPIO37 High/3.3V
		GpioDataRegs.GPBSET.bit.GPIO37 = 0; Does Nothing

GP?CLEAR	GP?CLEAR.bit.GPIO? = 1, Sets that Pin Low, 0V/GND
	GP?CLEAR.bit.GPIO? = 0, Does Nothing
		GpioDataRegs.GPCCLEAR.bit.GPIO70 = 1; Sets GPIO70 Low/0V
		GpioDataRegs.GPCCLEAR.bit.GPIO70 = 0; Does Nothing

GP?TOGGLE	GP?TOGGLE.bit.GPIO? = 1, Sets Pin opposite of its current state.
	GP?TOGGLE.bit.GPIO? = 0, Does Nothing
		GpioDataRegs.GPDTOGGLE.bit.GPIO98 = 1; was 3.3V then 0V or was 0V then 3.3V
		GpioDataRegs.GPDTOGGLE.bit.GPIO98 = 0; Does Nothing

GPIO Register Use When GPIO Pin Set as Input: Each GPIO pin, when setup as an input, has an internal pull-up resistor that can either enabled/connected or disabled/disconnected to that GPIO pin. With the passive push button on our breakout board, we will need to enable the pull-up resistor.

Register	Usage	Example

GP?DAT	If GP?DAT.bit.GPIO? is equal to 1 then the Pin is High, 3.3V
	If GP?DAT.bit.GPIO? is equal to 0 then the Pin is Low, 0V/GND
		if (GpioDataRegs.GPADAT.bit.GPIO19 == 1) {
		    //code that needs to run when input pin GPIO19 is High/3.3V
		} else {
		    // code that needs to run when input ping GPIO19 is Low/0V
		}

Laboratory Exercises

Exercise 1:

First, make sure your repository is up to date. Under Lab 0, find the Git help file titled "Using the ME461 Repository" and read and perform the steps of the last section of the document titled "Course File Updates." These steps will pull the latest updates from the class repository you forked in Lab 0. This procedure can be a bit confusing so ask your TA for help if needed. You should perform these steps each time you come to a new lab session to make sure you have the latest starter code.

Now that you have the updates, create a new project in your workspace and call it lab1. If you forget the steps of importing the labstarter project and renaming the project and files with lab1 names, see the lab 0 document. Once you have your new lab 1 project perform the below steps.

1. For this lab, you will only be using CPU Timer 2's interrupt service routine "cpu_timer2_isr(void)". We will leave the timer0 and timer1 functions in our source code but we will not enable timer0 or timer1. So in main() find the two lines of code that set the TIE bit to enable timer 0 and timer 1. Comment these two lines so they are not included in your program. i.e.

```
//CpuTimer0Regs.TCR.all = 0x4000;
//CpuTimer1Regs.TCR.all = 0x4000;
```

2. In main() find the “ConfigCpuTimer” function call for CPU Timer 2 and set its period to 0.25 seconds. Also find CPU Timer 2’s interrupt function “cpu_timer2_isr.” Note that in this function, it is blinking on and off six LEDs on the break out board and the blue LED on the Launchpad. Build and Debug this code to make sure that the code compiles and runs. You should see the blue LED along with the second LED row blinking on and off every half second. Once that is working, terminate your debug session and go to the next step.
3. In the cpu_timer2_isr function, comment out the lines of code that toggle the row of LEDs on and off. Leave the line that toggles GPIO31. Create a global int32_t variable and name it something like “numTimer2calls.” Inside the cpu_timer2_isr function increment that variable by one each time that function is entered. In addition, every time the function is entered, set the already defined global variable “UARTPrint” to 1. By doing this you are telling the main() while loop to print text through a UART serial port to your PC. Find this serial_printf() function call in the main() while loop. Does it make sense that when you set “UARTPrint” to 1, then the while loop calls the serial_printf function? Why is UARTPrint set to zero inside the if after serial_printf is called? Change the text so that it prints your “numTimer2calls” global variable. Since “numTimer2calls” is a 32 bit integer you will need to use the %ld formatter. Also have the serial_printf() function print the value of the numRXA variable just as it does in the default serial_printf() statement. You should not print the numSWIcalls variable because the SWI is not being used.

   To see this printed text you need to install a serial terminal on you PC. Tera Term is installed on the Windows machines in lab. On Mac do a web search for Serial Terminal for Mac. 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 “XDS100 Class USB Serial Port”. Run Tera Term and select the “Serial” item and find the XDS100 COM port in the list of COM ports. Final thing to do is change the Baud (or Bit) speed 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. Build and Debug your code and check that the LaunchPad’s blue LED is still blinking and your text is printing to Tera Term. Show your TA this working. As in Lab 0, type text in Tera Term to increment the “numRXA” variable that you are printing. We will not use numRXA in this lab, but just wanted to show that the UART is receiving characters along with transmitting characters.
4. Write two worker functions “void SetLEDColumnsOnOff(int16_t columns)” and “int16_t ReadSwitches(void)”.
   - void SetLEDColumnsOnOff(int16_t columns) takes an 16 bit integer as a parameter. The four least significant bits of this integer determine if the four LED columns are on or off. Bit 0 determines the right most column’s state. Bit 1 determines the right middle column’s state. Bit 2 determines the left middle column’s state. Bit 3 determines the left most column’s state. So for example if 6 (which is binary 0110) is passed to your function then both the middle columns of LEDs should be lit and the left and right column of LEDs should be off. Use four if statements inside your function to check, using the bitwise AND, &, operator, if the integer passed to your function has the least significant four bits either individually set or cleared. If set, turn ON the corresponding column. If cleared, turn OFF the corresponding column. See the above tables for definitions and example code on writing to the registers that control the LEDs. Also, the default C code that toggles the LEDs on and off in the timer_isr functions should help. Here though, I want you using the GP?SET and GP?CLEAR registers to turn on or off the LEDs. To test this function you could

increment a int16_t variable by 1 in your CPU timer 2 interrupt routine and pass this value to your SetLEDColumnsOnOff function. What happens if the number increment passed the value of 15?

- int16_t ReadSwitches(void) returns a 16 bit integer that the least significant four bits indicates the state of the four switches. (Note that when each of the switches are not pressed the GPIO pin reads a 1 or high voltage. When pressed the GPIO pin read a 0 or ground. This is because the IO pin is using an internal pullup resistor.) This function should have four if statements and use the bitwise OR, "|" operator to create this return value. So start the return value at zero. Then if switch 1 is pressed OR 0x1 with value. If switch 2 is pressed OR 0x2 with value. If switch 3 is pressed OR ??? with value. If switch 4 is pressed OR ??? with value. Finally return the value with the return() function call. See the above table for the GPIO pins that are connect to the push buttons and that are setup as inputs with pull-up resistor in the default code.

5. Now that you have these worker functions, make your program a bit more interesting. Add code in your CPU timer 2 interrupt function so that you print the value returned from your ReadSwitches() function by assigning a global int16_t variable to the value of ReadSwitches() and print this additional value in your serial_printf function in main()'s while loop. Also pass this value that is returned from your ReadSwitches() function to your SetLEDColumnsOnOff(value) function. This way both the LED columns and your printed text will indicate the state of the switches
   Show this working to your TA.

Exercise 2:

1. To get some more practice with starting a new project, create **another** new project by importing the labstarter example and renaming it and its main source file. Again, disable CPU timer0 and timer1's interrupt by commenting out:

   ```
   //CpuTimer0Regs.TCR.all = 0x4000;
   //CpuTimer1Regs.TCR.all = 0x4000;
   ```

   Change the period of CPU timer 2 to 0.25 seconds. Also copy from your previous project the two worker functions you created. Do not modify these worker functions. Instead use them in the below steps.
2. Change the code in cpu_timer2_isr to increment a global 32 bit integer (you created) by 1 every time timer 2's interrupt function is called. Pass this count variable to the SetLEDColumnsOnOff() function to display the least significant 4 bits of your count variable to the four LED columns. This is similar to what you codded to test your SetLEDColumnsOnOff() function in exercise 1. Compile, download to the DSP and verify that indeed the LED columns are counting in binary. Add one more item to this code as an exercise to see the use of bitwise operators in C. Using the ReadSwitches() function, an "if" statement and one or more of the bitwise C operators &, |, ~, check if push buttons 2 and 3 are pressed. If both of these push buttons are pressed, stop incrementing the global count integer. If they are released, continue counting. Again compile and download to the DSP. When your code is working, demonstrate your application to your TA.

Exercise 3: Breakpoints and Watch Windows

Starting with the code you just finished, we want to experiment with adding breakpoints to your code and using the "Expressions window" to edit the values of your variables.

1. In your previous code (with the DSP halted), put your cursor over the integer variable that you are incrementing. You should see that the value of the variable appears. Run your code, halt it again, and again put your cursor over the variable to confirm that it changes.
2. An easier method than using the cursor repeatedly is to add the variable to the Expressions window. When the DSP is halted, the Expressions window displays the current value of each variable in the Expressions window. To add your counting integer variable to the Expressions window, highlight the variable and then right-click, then select **Add Watch Expression…**. The variable will appear in the Expressions window with the current value of the variable. The Expressions window dialog is also found under the View menu.
3. Next play a bit with adding breakpoints and single stepping through a section of code. The code you have written to this point is very small. Add the following nonsense code to allow for easy use of breakpoints and code stepping. At the top of your C-file, but below the #includes, add the following global variables:

```
float x1= 6.0;
float x2= 2.3;
float x3= 7.3;
float x4= 7.1;
```

   Then inside your CPU timer 2 interrupt function add this nonsense code:

```
x4 = x3 + 2.0;
 x3 = x4 + 1.3;
 x1 = 9*x2;
 x2 = 34*x3;
```

   Build and load your code. Add a breakpoint to your code by double clicking on the left gray margin of your source file. A breakpoint is a location where the program will literally halt during execution. This allows you to check the values of your variables during operation. After a breakpoint, you can single step through your code (F5) and watch the variables update as different calculations are performed. You remove breakpoints by again clicking in the left gray margin.
4. If you happened not to receive any compiler errors during any of the above exercises, you should intentionally add some errors to your code so that you will see how CCS will alert you during the build process. Try double clicking on the error message. The editor will then take you to the line of code that has the error.

Exercise 4:

Still using the code from Exercise 2 and 3 make a few modifications. For many of our lab assignments we will want to have at least one of our timers running at a fast periodic rate. Most of the time that will be somewhere between a period of 1ms to 5ms. I would not be surprised though, if some of your projects will require you to run code at an even faster rates and the F28379D can definitely handle periodic rate of 0.1ms to 0.02ms. For this exercise, let's use a period of 1ms which can also be stated as a rate of 1Khz. Change CPU Timer 2's period to 1ms in order that CPU Timer 2's interrupt function is call once every millisecond.

The F28379D can do a huge amount of instructions every 1ms, but there are some things you do not want the processor performing every 1ms. For example the printing to Tera Term, if we printed every 1ms our eyes would not be able to see all the text spilling to the screen. Also calling the SetLEDColumnsOnOff() function every 1ms would cause a

blur if LED changes. So add code to your CPU Timer 2 interrupt function to only print every 100 times into the function. The % (mod) operator is perfect for this. Mod returns the remainder of an integer divided by another integer. i.e. (56 % 5) = 1. So using the int32_t integer that you are incrementing every time in the timer interrupt, write an if statement with a % (mod) condition that causes the if statement to be true every 100th time in the timer interrupt. Inside this if statement, perform all code that makes sense to run at the slower rate.
Demo this to your TA.

Lab Check Off:

1. Demonstrate your first application that continually checks the status of the four pushbuttons and displays their current state on the four LED columns.
2. Demonstrate your second application that updates a counter every quarter second and outputs the least significant 4 bits of the count to the four LED columns. The count should stop if pushbuttons 2 and 3 are pressed and resume when they are released.
3. Demonstrate that you know how to use Breakpoints and the Watch Window to debug your source code.
4. Demonstrate your 1ms timer period code working.

Take Home Exercise:

Using the ReadSwitches() function you created above, have the CPU timer 2 interrupt function check the status of the 4 pushbuttons every 100ms. Then pass the value returned from ReadSwitches() to a new "worker" function that takes as a parameter an int16_t integer. Your worker function then needs to look at the four least significant bits of that 16 bit integer and correspondingly print the 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F character to the 23 LEDs. These are all the hexadecimal digits. For example if 2 is passed to this new function, it should turn on the LEDs in this pattern: (Black is LED on)

	■	■			1	7	13	19		1	7	13	19		1	7	13	19		1	7	13	19		1	7	13	19
■			■		2	8	14	20		2	8	14	20		2	8	14	20		2	8	14	20		2	8	14	20
		■			3	9	15	21		3	9	15	21		3	9	15	21		3	9	15	21		3	9	15	21
	■				4	10	16	22		4	10	16	22		4	10	16	22		4	10	16	22		4	10	16	22
■					5	11	17	23		5	11	17	23		5	11	17	23		5	11	17	23		5	11	17	23
■	■	■			6	12	18	NC		6	12	18	NC		6	12	18	NC		6	12	18	NC		6	12	18	NC

If you would like to choose a different pattern for 2, that would be fine also. I have grids labeling the LED numbers to help you sketch the other hex digits.

Additional Questions to answer in your report:

1. If there was such thing as a 24 bit signed integer, what would be the largest positive number it could represent and what is the smallest negative number it could represent.
2. Below are three int16_t integers represented in binary format. What are these numbers in decimal format?
   - i. 1101110000011011
   - ii. 0001111100110101
   - iii. 1000000010110011
3. In question 2i, is bit 10 high 1 or low 0?
4. In the lab starter code, you have modified the period value passed to the function "ConfigCpuTimer". For example the default code sets CPU Timer 0's period to 10000 microseconds with the following line of code "ConfigCpuTimer(&CpuTimer0, 200, 10000);" The main functionality of this function is to set the PRD (Period) register to the correct value such that the timer times out every 10000 microseconds. The PRD stores a 32 bit unsigned integer. The TIM (timer register) is also a 32 bit unsigned number. The TIM register starts at 0 and counts up by 1 every 1/200000000 seconds (200Mhz). Whenever the TIM register reaches the value stored in the PRD register an interrupt event is issued calling the CpuTimer0 interrupt service routine. At this moment the TIM register is also set back to 0 to start timing again. Knowing this, what is the largest period in seconds that the CPU Timers can be set to?
5. Using bitfields, we could check if two bits where set in the GPADAT register with the following if statement:

```
if ((GpioDataRegs.GPADAT.bit.GPIO12 == 1) && (GpioDataRegs.GPADAT.bit.GPIO13==1)) {
        // do something
}
```

   How could you perform this same check using the GPADAT.all, entire 32bit GPADAT register? Note the other bits in the GPADAT register could be 1 so your if condition will have to read GPADAT.all and clear (or mask) all the other bits besides 12 and 13.

6. Pushbuttons 1, 2, 3, and 4 are connected to GPIO 122, 123, 124, 125. GPDDAT is the register that can be read to see the state (high or low) of these pins. Your ReadSwitches() function probably used the GpioDataRegs bitfield to look at 122, 123, 124, 125 individually and that was correct. GPDDAT is for GPIO pins 96 through 127. So 122, 123, 124, 125 are bits 26, 27, 28, 29 in the 32 bit GPDDAT register. Using a right shift (>>) operation and then a bitwise and (&) operation (because bits 30 and 31 could be set) and finally a not (~) operation (because when the push button is not pressed the corresponding bit is 1) write the ReadSwitches() function in one line of code that operates on the GpioDataRegs.GPDDAT.all register. Remember ReadSwitches returns a number between 0 and 15. i.e.

```
int16_t ReadSwitches(void) {
    return( Your one line of code here );
}
```

Items you need to perform and submit for this lab, due before your next lab session.

1. Your **commented** code is your report that will be graded. Take time to add comments explaining what you understand is happening in the code you wrote and the functions in which your code is running. This commented code should also include your code for the take home exercise. Please make it obvious in your submission which code is for each exercise. I do not want short hard to understand comments. Instead, I would like short paragraphs explaining the code you wrote. In addition, a video of the take home exercise working outside of the lab room needs to be submitted to your Box folder I created for your video and report submissions in a folder you create called "Lab1". Also submit your report to that Box folder. Finally submit the current state of your HowTo.docx (or whatever type of file you want to use). This HowTo.docx file should include items that will help you remember how to perform tasks with your F28379D Launchpad board during this semester and after you are done with this course. You will submit this HowTo.docx file at each lab submission.