SE420 Laboratory Assignment 3
Optical Encoder Angular Feedback and PWM Output

Goals for this Lab Assignment:

1. Learn functions to read Optical Encoder input Channels and write to PWM output channels.
2. Estimate Velocity from Optical Encoder Position Feedback.
3. Store data to be uploaded to MATLAB after the data collection time has elapsed.
4. Learn to use the given MATLAB functions for saving and uploading data.

New Library Functions Used:

readEnc1, setEPWM3A, setEPWM7A

MATLAB Functions Used:

serialreadSPIRAM.m, startCollect.m

Answer this question below at the beginning of the lab so you know how to calculate angular velocity given angle feedback in your C code. You may want to do this before you come to lab but it does not have to be turned in.

1. Review differentiating a signal using the backwards difference rule, $s = \frac{z-1}{Tz}$, also written, $s = \frac{1-z^{-1}}{T}$. Given the continuous transfer function of velocity over position, $\frac{Vel(s)}{Pos(s)} = s$, use the backwards difference approximation and find the difference equation for v_k. i.e. $\frac{Vel(z)}{Pos(z)} = \frac{1-z^{-1}}{T}$. How would you implement this difference equation in C code? This is how you will find velocity given a new angle feedback reading every sample period T.

Laboratory Exercise

Ask your instructor if there are updates to the lab repository. If there are, follow the "Course File Updates" instructions in the Using the SE420 Repository , http://coecsl.ece.illinois.edu/se420/UsingTheSE420Repository.pdf document.

To find help for a C function that is new to you, find the function or type the name of the function in your C file and then highlight the function name, right-click and select "Open Declaration." This will take you to the definition of the function. Comments before each function should give help on how to call the function.

You should now be pretty familiar with setting up and programming one of the CPU timer interrupt functions in order to execute your desired code at a certain periodic rate. For this lab we will use CPU Timer 0's interrupt function. We will not discuss these steps so recall what you did in lab 2 and reference lab 2 if you have trouble. We are simply going to list the requirements for the three exercises.

Exercise 1: Basic Input and Output

1. Create a new project from the LABstarter example and rename it lab3 along with an appropriate new name for "LABstarter_main.c".
2. Set CPU timer0's interrupt function to be called every 5 ms.
3. In CPU timer2's interrupt function comment out the lines of code that set UARTPrint to 1 after certain number of interrupt calls. Instead, add code to CPU timer0's interrupt function that sets UARTPrint to 1 every 250ms. In a below step you will be told what to print.

4. The Red Board's (the F28377s board) Red LED is connected to pin GPIO12. As we did in Lab 2, we can use GPIO12 as a GPIO pin and turn off and on the Red LED. But GPIO12 can also be setup through the F28379D multiplexer as EPWM7A, pulse width modulation channel 7A. A pulse width modulation signal is a square wave that can have its duty cycle changed in source code as needed. Our main use of PWM for SE420 will be commanding a DC motor with a desired torque command. But for this section we will drive the Red LED with a PWM signal to make the brightness of the Red LED change as we change the duty cycle of EPWM7A. Open the datasheet of the F28377s chip, F28377s Datasheet , http://coecsl.ece.illinois.edu/se420/tms320f28377s.pdf Find Table 6-2 on page 40 of the datasheet which is the Pin Multiplexing Table. Notice that the top row is titled "GPIO Mux Selection" and index can have a value 0,4,8,12,1,2,3,5,6,7,15. In the table find GPIO12's row. This row indicates the different functionalities that GPIO12's pin can be set to besides the GPIO12 functionality: EPWM7A, CANTXB, MDXB, EQEP1S, SCITXDC and UPP-ENA. Notice that one of these functionalities is EPWM7A. So to make GPIO12 function as EPWM7A we need to set its MUX index to what number?
5. Now that you know the GPIO Mux Index, we can change it in our code so that the pin is EPWM7A. In main() find the initialization of GPIO12:

```
// Red LED, GPIO12
GPIO_SetupPinMux(12, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(12, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO12 = 1;
```

   The GPIO_SetupPinMux function does two things. First it assigns this pin to CPU1. (If we were using a version of this processor with two CPUs, CPU2 could have assigned here and then the second CPU would have control of GPIO12) Second the "0" is setting the Pin Mux Index to 0 so the pin is multiplexed to GPIO12. Change the Pin Mux Index so that the pin is now EPWM7A. You can leave the next two lines, but since the pin is EPWM7A, they have no effect. Also notice that the default main() function calls initEPWM7A() which initializes the EPWM7 unit's registers.
6. Add the following code to your CPU timer0 interrupt function:
   a. Read the value of optical encoder channel one into a global float variable. Use readEnc1() which returns the motor's angle in radians.
   b. Send the radian value of encoder channel 1 to PWM7A output. This is a little weird in that we are taking a radians value from our feedback and directly using it as a PWM %duty cycle value. But for this exercise we are just using the optical encoder angular measurement to create a "knob" that changes the duty cycle driving the red LED. So simply take the global variable that has been assigned the angle measurement in radians and pass that value to the setEPWM7A(float u) function. In the setEPWM7A function, u is a value between -10 and 10. You can pass a value lower than -10 or higher than 10 without any issue but the value is saturated at those limits. In order to get you started thinking about driving the motor with a different PWM signal, I allowed the PWM value to range from -10 to 10. Driving the red LED with this PWM signal we will not see a difference but when driving our motor, 10 means full torque in the positive direction and -10 means full torque in the opposite direction. So a value of 0, passed to setEPWM7A, is 0% duty cycle, 5 is 50%, -5 is 50%, 10 is 100%, -10 is 100%.
   c. Every 250ms print the value of encoder channel 1 to the LCD screen. Keep your UART_printfLine function call in main()'s while loop and just set UARTPrint to 1 every 250ms (you did this already in step

3). To print a float use %.3f as the formatter to print the floating point variable with three digits of precision.

7. Build and run your application. Turn your motor/optical encoder back and forth and notice the red LED getting brighter and dimmer.
8. I would also like you to see this PWM signal on the oscilloscope. The easiest PWM signal to scope on our board is PWM3A. So just after the line of code that calls setEPWM7A, call setEPWM3A passing it the same variable you passed to setEPWM7A. Build and run this code. Your TA will help you with the oscilloscope in lab. While moving the attached optical encoder, the PWM duty cycle should vary from 0% to 100%.
9. With the optical encoder sensor, if you keep on spinning the wheel, the angle feedback keeps on getting larger and larger. So this is why we are printing the encoder's value in radians to the LCD. You can look at the encoder value on the LCD and make sure you stay between 10 and -10 so that you can see the LED dimming and brightening.
10. Demonstrate to your TA both your LED changing intensity and the signal on the oscilloscope before proceeding to step 11.
11. Delete your call to setEPWM3A, which you used for scoping the PWM, because in Exercise 2 you are going to use EPWM3A to drive the motor below.

Exercise 2: Basic Feedback Calculations

1. Use same code as exercise #1. Keep sample period at 5 ms.
2. Create a global float variable "u" to be used as the variable passed to setEPWM3A. For this exercise, initialize u to 5.0 when you create it as global variable above your main() function, like all the other global variables.
3. Change your CPU Timer0 interrupt function so that "u" is passed to setEPWM3A each sample period.
4. As a first step, compile and download this code. In a "watch expression" add your "u" global variable. Start your code running, if it is not, and slide your amp enable switch to ON to enable the DC motor. Your motor should start spinning. Now in the "watch expression" window, change your u value to different numbers between -10 and 10 and notice the motor running at different speeds and either clockwise or counter-clockwise. Slide your amp enable switch to the off position, terminate your debug session and go onto the next step.
5. Using the backwards difference rule ($s = (z-1)/Tz$) as a discrete approximation to the derivative, calculate the angular velocity of the motor in radians/second from the motor position measurement, readEnc1(). Save this in a new global float variable.
6. Print both the value of u and the calculated angular velocity to the LCD every 250 ms.
7. Demo to the TA.

Exercise 3: Collect Data and Upload to MATLAB.

1. To have your code save data and then later upload that data to MATLAB we have created a function "saveData" that saves five floating point number each time the function is called. Instead of saving these data points to the F28377S memory, the data is sent, by SPI serial port, to a SPIRAM chip. This way the F28377S's memory is not used for data collection. The idea is to call "saveData" each time your periodic timer interrupt function is called, saving five variables of your choosing. If you look in the cpu_timer0_isr interrupt function, you will see some

commented code that shows how to use "saveData". Uncomment the "saveData" line of code along with the next line that increments the variable time by 1. These lines of code save data that are sine waves with different amplitudes. In the next step, you will change what is being saved by this "saveData" function. The "saveData" function can be continuously called, but after it has stored the desired number of points, it ignores the data passed to the function and just returns. There is a #define NUM_POINTS defined in SpiRAM.h that determines how many data points can be collected. By default it is set to 6000. You will change this value, in the next step, to the length of data needed. This NUM_POINTS value must be dividable by 200 and cannot be longer than 65400. Also note that while the saveData function is storing values, it sets an LED (that your TA will show you) off. When saveData has saved NUM_POINTS of five variables, this LED will turn on.

2. For the first 5 seconds of your run, pass time (in seconds, so make sure to multiply time by 0.005 instead of the 0.001 in the given line of code.), angular velocity (in rad/sec) and u (proportional to torque) to the saveData() function. For the last two parameters of saveData just pass, 0 and 0, since we do not need to save 5 variables here. 1000 samples at 5ms equals 5 seconds so change NUM_POINTS to 1000.
3. Change "u" so that it is a time varying output:

   ```
   float ampl = 3.5;  // make a global variable
   float f = pick a frequency  // make a global variable
   u = ampl*sin(2*PI*f*time*0.005);
   ```

   This will make your plots look a little more interesting. Your equation for "u" has two variables, ampl and f, that you can modify to change the input to the motor.
4. Make sure to have your motor ready to turn on and then run your application and wait for the LED to turn on indicating all data has been stored and is ready to be uploaded to MATLAB.
5. Change MATLAB's working directory to the "matlab" folder in your Lab 3 project's directory which is in your Code Composer workspace folder. This folder stores the three functions you are going to use: list_serialports, serialreadSPIRAM and "startCollect".
6. To find which COM port your F28377S board is using, call "list_serialports" at the MATLAB command and it will list the available COM ports. Most of the time, your COM port will be the largest number but if you are not certain, you can use Windows Device Manager to find the F28377S's COM port. Then in MATLAB call the MATLAB function **data = serialreadSPIRAM('COM?')** to upload your data to MATLAB's workspace. This will return the collected data and store it in the MATLAB variable data. "data" will be a five column 2D array with NUM_POINTS rows. The first column is the logging of the first variable passed to "saveData". The second column is the second variable passed to "saveData". Plot your data in MATLAB.
7. Use "Watch Expression" tab in Code Composer Studio to modify either the "ampl" variable or the "f" variable. By doing this the sine wave applied to the motor will change. First get your motor moving with the adjusted sine wave. Then to collect another set of data, run the MATLAB function **startCollect('COM?').** You will see, after you run "startCollect", that the LED turns off while data is being stored and then turns on when all the data has been stored. Run serialreadSPIRAM again to upload the new set of data. Make a plot of the second set of data. Confirm the two plots are different.
8. Show your plots to the TA.

Lab Check Off:

1. Demonstrate your first application that reads optical encoder channel 1 and echoes the radian values to PWM7A. Scope the outputs to demonstrate your code is working also notice how the red LED is getting brighter and dimmer as the PWM duty cycle changes.
2. Demonstrate your second application that drives the motor with an open loop PWM output and prints this control effort and the motor's calculated speed to the LCD.
3. Show your MATLAB plots demonstrating that you figured out how to upload data from MATLAB.