GE420 Laboratory Assignment 4
Introduction to the I/O of the TMS320F28377S Processor

Goals for this Lab Assignment:

1. Learn functions to read Optical Encoder Channels and write to both PWM and DAC output channels.
2. Estimate Velocity from Optical Encoder Position Feedback.
3. Store data in an array to be uploaded to MATLAB after your application has been run.
4. Learn to use the given MATLAB functions for uploading data.

DSP/BIOS Objects Used:

- Clock

Library Functions Used:

readEnc1, setDAC1, setDAC2, setEPWM3A ,UART_printfLine

MATLAB Functions Used:

GE420_serialread, GE420_serialwrite

Prelab: YOU WILL NEED TO TURN THIS PRELAB IN AT THE BEGINNING OF LAB!

1. In Lab 3 we used the processes Clock, Swi and Task. We really didn't *need* the Swi or the Task because all the operations could have been done in the Clock because the DSP is so fast. If this wasn't the case and there was a lot more code to process, when is it appropriate to use a Swi and when is it appropriate to use a Task? I am not looking for a detailed answer here but I want you to re-read the material from the prelab sections of Lab 3 and come up with a paragraph explaining in which cases you would use a Swi and which cases a Task.
2. 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?

Laboratory Exercise

Instead of doing something new with the SYS/BIOS kernel, today we are going to focus on some of the I/O of the TMS320F28377S DSP processor. We have access to the following I/O:

- 2 Channels of Optical Encoder Input.
- 4 Channels of PWM Output.
- 2 Channels of +/- 10V DAC Output.
- 2 Channels of +/- 10V ADC Input.
- 4 Digital Inputs (Switches).
- 4 Digital Outputs (LEDS).

The functions that interface the DSP to this board are found in the files f28377sDAC.c, f28377sEPWM3A.c, f28377sEqep.c. Use these files to help you write your code with these functions. To find help for a function, type the name of the function in your C file and then highlight the function name, right-click and select "Open Declaration." Comments before each function give help on how to call the functions.

You should now be very familiar with the Clock object, so we will not discuss the steps for creating a Clock instance in the exercises for this lab. Instead, we are simply going to list the requirements for the three exercises.

Exercise 1: Basic Input and Output

1. Create a new project with the project creator program.
2. Create a Clock object, **Clock_basicio**, to call a function every 5 ms
3. Write the function, **basicio**, and associate it with the Clock object. Add the following code to your Clock's 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 PWM3A output. Use the function setEPWM3A. Also send this same radian values to DAC channels 1 & 2. Use setDAC1 and setDAC2.
    c. Every 250ms print the value of encoder channel 1 to the character LCD. To print a float use %.3f as the formatter to print the floating point variable with three digits of precision.
4. Build and run your application.
5. Verify your program by 'scoping' the DAC outputs. While moving the motor's encoder, the DAC voltage should vary between –10 and +10 volts.
6. Also verify your program by 'scoping' the PWM output. While moving the two attached encoders, the PWM duty cycle should vary from 0% to 100%.
7. Demonstrate to the TA before proceeding to next exercise.

Exercise 2: Basic Feedback Calculations

1. Partners switch places to allow other partner to program.
2. Use same code as exercise #1. Keep sample period at 5 ms.
3. Create a global float variable "u" to be used as the variable passed to setPWM3A. For this exercise the value of "u" should always be set to 5.0.
4. Change your Clock function so that "u" is passed to setPWM3A each sample period. Enable the PWM amplifier when running this code so the motor will spin (The amp is enabled by flipping the switch on the amp board).
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. Save this in a new 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: Basic Data Acquisition and PC Communication

1. Add three, 1000-point float arrays to your application. The three arrays should store elapsed time, angular velocity, and control effort "u" for the first 1000 sample periods. To add an array that Matlab can access, you need to place the arrays in specific sections of memory where Matlab will know to look for these arrays. We have created a memory section called ".my_arrs" for these Matlab accessible arrays. For example, if you want to add a 1000-point "testarray" array, declare it as follows:

```
#pragma DATA_SECTION(testarray, ".my_arrs")
float testarray[1000];
```

2. Make sure that your code only saves 1000 points to the array. DON'T write past the size of the array!
3. Change "u" so that it is a time varying output of some kind (a sine wave for instance). This will make your plots look a little more interesting. Your equation for "u" should have at least one variable (i.e. the amplitude of a sine wave) in it that you will be able to modify in upcoming steps. Since you will want to be able to modify this amplitude variable from Matlab, it also needs to be placed in a special data section. In this case, it should be declared as follows in the "my_vars" memory section:

```
#pragma DATA_SECTION(amp, ".my_vars")
float amp = 1;
```

4. To prevent the DSP from writing to your arrays while you are sending them to Matlab, we have created a variable in the "my_vars" data section called "matlabLock". To access this variable in your code, you will need to use a shadow variable called "matlabLockshadow". Paste the following line at the top of your code to access this shadow variable. This variable is set to 1 when you start reading data in Matlab, so use the variable to prevent the DSP from writing to your arrays when it is equal to 1.

```
extern float matlabLockshadow;
```

5. Run your application for at least 1000 samples.
6. Halt the DSP, and add your control effort array to the watch window. Confirm that your code is saving data correctly to the array. For instance, is the last point in the array garbage or a good value?
7. Change Matlab's working directory to be the "matlab" folder in your DSP project's directory (i.e. C:\<netid>\<RepositoryName>\trunk\<projectname>\matlab).
8. In CCS make sure to click the "resume" (Green Arrow) to start the DSP running again. Then use the MATLAB function **GE420_serialread** to upload your 3 arrays to MATLAB's workspace. Type 'help GE420_serialread' for help. (Don't forget the ' ' (single quotes) around your variable names.) Plot your data in MATLAB. Does the data in MATLAB match the data in the watch window? Note, the C type 'float' is of type 'single' in MATLAB.
9. Use the MATLAB function **GE420_serialwrite** to modify a parameter in your code that changes the time varying 'u' value (i.e. the amplitude of the sine wave you are sending to the motor). Type 'help GE420_serialwrite' for help. For the Matlab commands **GE420_serialread** and **GE420_serialwrite** to work the DSP must be running, so before you download values to your variable make sure the DSP is running.
10. After you have modified the amplitude from Matlab, use **GE420_serialwrite** again to set "matlabLock" to zero. This will allow the DSP to start saving new data to your arrays.
11. After a few moments, upload the new set of data and make a second plot (remember the DSP needs to be running to communicate with Matlab). Confirm the two plots are different.
12. 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 PWM3A and DAC channels 1 & 2. Scope the outputs to demonstrate your code is working.
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/download data to/from MATLAB.