Goals for this Lab Assignment:

1. Learn about how to use TMS320F28335’s analog-to-digital converter
2. Introduce the hardware interrupt (HWI) SYS/BIOS object.
3. Gain a better understanding of Serial Peripheral Interface, SPI, and how the DSP interfaces (or communicates) with I/O hardware.
4. Learn how to design an expansion board for the robot’s processor board

SYS/BIOS Objects Used:
Hwi, Clock

Library Functions Used:
void writeDAC7564(float dac1, float dac2)

Matlab Functions Used:
BUTTER

Prelab:
This Prelab will need to be completed before you start the lab exercise. The user’s guide for the TMS320F28335’s ADC peripheral is located at [http://coecs1.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/ADC.pdf](http://coecs1.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/ADC.pdf). There are a number of different modes of operation this ADC peripheral can be setup to perform. Keep in mind the following settings as you read through the user’s guide so you can focus on the required settings and skim over the other modes:

Desired ADC Peripheral Settings for Labs 4 and 5:

1. Cascaded sampling Mode
2. Simultaneous sampling Mode
3. Sample these 10 channels in this order ADCINA0, ADCINB0, ADCINA1, ADCINB1, ADCINA2, ADCINB2, ADCINA3, ADCINB3, ADCINA4, ADCINB4.
4. Enable Sequence 1’s Interrupt
5. Set ADC to use its internal reference

Prelab Question #1: We are going to be starting the ADC sequence every 1ms from inside a Clock function. Assuming the ADC is configured for cascaded sampling, what bit of what register needs to be set to start the conversion of the ADC sequence one? Note: ADC sequence one (SEQ1) is the only sequence used when cascaded mode is set to 1.

Prelab Question #2: Complete the given code, that you will later place in your main() function, that initializes the ADC peripheral. The power on initialization of the ADC is given for you in the function InitAdc() but you need to figure out the initialization of the other ADC peripheral registers. Much of the C code is given below. You need to set the given bit fields of the ADC register structure to the appropriate values.

Find the correct values to replace the “??” of the below code that configures the ADC as per the comment of each line. The bit field structures that are being used in this code break the register into multiple variables that have the same
The number of bits as described in the ADC user’s guide. So, for example the first variable given below for register ADCTRL1 is the bit field SUSMOD. Looking at the datasheet you see that SUSMOD is 2 bits of ADCTRL1. Therefore, the possible values you could set AdcRegs.ADCTRL1.bit.SUSMOD to are 0, 1, 2 or 3. For a bit field with only 1 bit of a register, the value is either 0 or 1, etc.

```c
InitAdc(); // This function takes care of powering on the ADC correctly.
AdcRegs.ADCTRL1.bit.SUSMOD = ??; // Emulation Suspend mode ignored
AdcRegs.ADCTRL1.bit.ACQ_PS = ??; // 5 ADCCLK periods
AdcRegs.ADCTRL1.bit.CPS = ??; // divide by 1
AdcRegs.ADCTRL1.bit.CONT_RUN = ??; // Start Stop mode
AdcRegs.ADCTRL1.bit.SEQ_OVRD = ??; //Sequencer Override Disabled
AdcRegs.ADCTRL1.bit.SEQ_CASC = ??; // Cascaded mode enabled
AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = ??; //Enable SEQ1’s interrupt
AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = ??; //Set Interrupt to occur every SEQ1 completion
AdcRegs.ADCTRL3.bit.ADCCLKPS = ??; //Core clock divided by 18
// Found this by trial and error. Could need to be changed
// given different sensor inputs
AdcRegs.ADCTRL3.bit.SMODE_SEL = ??; //Set to Simultaneous sampling mode
AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = ??; // set so that A0 B0, A1 B1, A2 B2, A3 B3, A4 B4 are
// sampled
AdcRegs.ADCREFSEL.bit.REF_SEL = ??; // Internal reference used
AdcRegs.ADCCHSELSEQ1.bit.CONV00 = ??; // set these 5 fields so that the order of ADC channels
AdcRegs.ADCCHSELSEQ1.bit.CONV01 = ??; // sampled is A0,B0,A1,B1,A2,B2,A3,B3,A4,B4.
AdcRegs.ADCCHSELSEQ1.bit.CONV02 = ??;
AdcRegs.ADCCHSELSEQ1.bit.CONV03 = ??;
AdcRegs.ADCCHSELSEQ2.bit.CONV04 = ??;
AdcRegs.ADCTRL2.bit.RST_SEQ1 = ??; // Reset SEQ1
AdcRegs.ADCST.bit.INT_SEQ1_CLR = ??; // Clear INT SEQ1 Good practice to clear possible
// interrupt before enabling the interrupt.
```

**Know how to use the BUTTER function** in Matlab to design both low- and high-pass filters. Understand how to take the coefficients returned by these functions to create difference equations that are coded in C to implement a filter. The help for “filter” in Matlab gives the order of coefficients for filters created in MATLAB. You will have to design two filters for the lab, so give some thought on how to code the difference equations in C.

**Laboratory Exercise**

**Exercise 1: Using the TMS320F28335’s ADC Peripheral**

To give you some experience with the hardware interrupt (HWI) section in SYS/BIOS, we will setup the ADC peripheral to sample a sequence of ADC inputs and on completion of this sequence have the ADC generate an interrupt.
The function that runs when this interrupt occurs is called the ISR (interrupt service routine) of that interrupt event. You will write this ISR to read all the sampled ADC values and also start the reading of the optical encoder LS7366 SPI chips.

The role of the TMS320F28335 processor on the robot’s processor board is currently 3-fold. One is not time critical and that is printing to the text LCD screen. The second role is running the motor PI speed control algorithm every 1ms. We will focus on implementing this PI control in Lab 5. The third role is reading all sensor inputs connected to the TMS320F28335’s peripherals and outputting to devices through PWM outputs, DAC outputs or general purpose I/Os. In Lab 3 you already experimented with the optical encoder inputs, PWM outputs and DAC outputs. For this first exercise you will take the code given in the TMS320F28335’s project creator that already samples the optical encoder sensors every 1ms and add the sampling of the desired ADC channels.

Much more detail is given below in the procedures to accomplish this implementation but in short you will be adding the starting of the ADC sequence, the sequence completion interrupt and the reading of the ADC samples into the overall flow of sampling the 4 SPI LS7366 optical encoder chips. In the start_dataCollection() Clock function instead of calling start_SPI() you will set the appropriate ADC register bit to start the ADC conversion sequence. On completion of the sampling sequence the ADC’s interrupt ISR will be called. Inside this function you will read the 10 ADC readings and then call the function start_SPI() which will continue to read all the optical encoder readings as in the original code. In this way you have inserted the reading of the ADC channels at the same sample rate as the reading of the optical encoders.

Earlier TMS320F2000 series of TI processors had only 12 interrupt sources. As this series of processor matured more interrupt sources were added. Such a large number that TI came up with a separate peripheral call the PIE (peripheral interrupt expansion) to handle the extra interrupts. The PIE now allows for 96 different interrupts, 8 interrupts per each 12 legacy interrupts. Looking at the System Control and Interrupt User’s Guide http://coecs.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/syscontrolandinterrupts.pdf, Table 111, page 124, you can see all the sources of interrupts for the TMS320F28335. The interrupt we are interested in for this assignment is ADCINT. Finding it in Table 111, it is the PIE interrupt INT1.6. This indicates that legacy interrupt #1 will need to be enabled along with PIE interrupt 1.6. Finally locate in Table 112, INT1.6 (ADCINT), and note that its interrupt Vector ID is ________.

**Echo ADC values to DAC Procedure:**

1. As a first step, add your prelab’s initialization code for the ADC to your main function. Place this code right before the post_init() function call in main(). After this initialization code also add the following additional lines that enables legacy interrupt #1 and PIE interrupt 1.6.

```
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;  // Clear possible interrupts before enabling int.
PieCtrlRegs.PIEIER1.bit.INTx6 = 1;       // enable PIE interrupt 1.6
IFR &= ~M_INT1;                          // Clear a possible pending interrupt in the interrupt flag register
IER |= M_INT1;                           // Enable legacy INT1 in the interrupt enable register
```

2. The default project creator code already given to you uses the following flow to sample the four optical encoder interface chips (LS7366) every 1ms:

   i. The Clock object (perhaps named CLK_StartADC) is setup to be triggered every 1ms.
   ii. CLK_StartADC’s function void start_dataCollection(void) is therefore called every 1ms and it simply calls the function start_SPI().
iii. `start_SPI()` sends an 8-bit command to the four LS7366 chips telling them to latch their encoder counts to the chip’s read register.

iv. After the 8-bit command has been received by the LS7366 chips, the SPI interrupts the TMS320F28335 indicating the transfer is complete and it is ready for more data to transfer.

v. Inside the SPI’s ISR, LS7366 chip #1 is selected and its count value is read over the SPI serial port. After optical encoder #1’s 32-bit count has been read the SPI again interrupts the processor and the procedure is repeated three more times to read chip #2, #3 and #4.

vi. At the final interrupt, indicating that the LS7366 #4’s count value has been read, the Swi object `swi_control` is posted. Then when priority permits, the SWI’s function `void control(void)` is called. Here is where you place your code that does something with the sampled sensor readings.

For this lab and remaining labs, you will also need to sample the ADC inputs every 1ms. Do this by inserting the sampling of the ADC inputs at the beginning of the LS7366 read process.

i. Inside CLK_StartADC’s function `void start_dataCollection(void)` instead of calling `start_SPI()`, set the appropriate ADC register bit to start the conversion sequence (Prelab question #1).

ii. Then when the conversion sequence is complete the ADC’s ISR will be called. Inside the ADC’s ISR, after you have read the 10 ADC channels, call `start_SPI()` to read the LS7366 chips. In step 3 you will create the ADC’s ISR.

3. Create the ISR function to be triggered by interrupt 1.6, ADCINT. Just like other SYS/BIOS functions, this function should have `void` parameters and a `void` return value.

   a. Inside this function, read the 10 ADC readings from the ADC’s Conversion Results Buffer Registers starting at address 0xB00. See Figure 2-14 in the ADC user’s guide [http://coecsle.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/ADC.pdf](http://coecsle.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/ADC.pdf). The TMS320F28335 header files name these result registers the ADC mirrored results registers and gives them the C structure name `AdcMirror`. So, for example, to read the first ADC sampled value you would use the following C statement:

   ```c
   raw_adc_A0 = AdcMirror.ADCRESULT0;
   ```

   Ten global integer variables, `raw_adc_A0` through `raw_adc_A4` and `raw_adc_B0` through `raw_adc_B4`, have already been created for you in the given code. They have already been created for you because the given source code also uses these variables to communicate the ADC readings to the OMAPL138 processor through the McBSP serial port. So only assign these variables the raw integer readings from the result registers and use different variables in later steps when converting the raw ADC reading to a voltage.

   b. After reading the 10 ADC results, start the read process of the LS7366 optical encoder chips by calling the `start_SPI();` function.

   c. Most peripheral interrupts have a status bit in one of its peripheral registers that indicate that this interrupt source has been triggered. In many cases this status bit needs to be cleared to allow another interrupt to be triggered. Most of the time the clearing of this status bit is done at the end of the ISR.
This is the case for the ADC interrupt so add the following code to reset the ADC and clear the interrupt flag.

```c
// Reinitialize for next ADC sequence
AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;     // Reset SEQ1
AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;   // Clear INT SEQ1 bit
PieCtrlRegs.PIEACK.all=PIEACK_GROUP1;//Acknowledge/CLR interrupt to PIE
```

4. With the ADCINT's interrupt service routine written, setup SYS/BIOS to call this function when the ADCINT interrupt occurs.
   a. Add a HWI object by right clicking on the Hwi and selecting New Hwi. Click on the new Hwi which was probably given the default name Hwi0 and the Hwi Instance tab will be selected. Change the Handle of this Hwi object to hwi37_ADC (you can pick a different name if you would like). In the ISR function, enter your ISR’s function name created in the step-3. The interrupt number is set to 37. Why 37? Change the default Masking options from “MaskingOption_SELF” to “MaskingOption_ALL”. The interrupt mask set to “all” disables all other interrupts while this ISR is running. On completion of this ISR all interrupts are re-enabled.

5. Now that the conversion of the ADC channels has been inserted into the data collection sequence, you can use these sampled ADC values along with the optical encoder readings. The best location to put this code is in the swi_control's function `control()`. Here you have all the ADC readings and all the optical encoder readings.

   Similar to what we did in the first exercise of Lab 3, echo the value sampled on ADCINB1 and ADCINB2 to DAC outputs 1 and 2. To do this you will need to scale the raw ADCIN reading that is in the range 0 to 4095 to a volt ranging from 0V to 3.0V. Then write this scaled value to the DACs using the `writeDAC7564()` function. ADCINB1 and ADCINB2 are connected to the blue and orange banana jacks on the robot’s left handle support and DAC1 and DAC2 are connected to the white and gray banana jack on the right support.

6. Build and load your code to the TMS320F28335. Check that the code is working by inputting the same voltage signal to both ADC channels and scoping both this input and the DAC channel outputs. Your instructor will show you how to setup the function generator at your bench to produce a sine wave that varies from 0V to 2.5V. Compare how the input looks compared to the DAC output. Start first with a slow frequency and then gradually increase the frequency of the sine wave. At slow frequencies the output should compare pretty well with the input. At higher frequencies the output starts having a “stair step” look. If you go above the Nyquist frequency you will start to see aliasing. Demonstrate this working to your instructor.

**Exercise 2: Filter Design and implementation**

The Butterworth filter is a common filter used in signal processing. In the discrete time domain, the transfer function of the Butterworth filter is given by:

\[
\frac{Y(z)}{X(z)} = \frac{b_1 + b_2 z^{-1} + \cdots + b_{n+1} z^{-n}}{1 + a_2 z^{-1} + \cdots + a_{n+1} z^{-n}} \quad (1)
\]

where \(n\) represents the order of the filter. If we assume that the filter is linear time invariant, then we can represent Equation (1) using a linear constant-coefficient difference equation:
In Equation 2, the current output of the Butterworth filter, \( y(0) \), is determined based on the filter coefficients \( a_k \), \( b_k \), past outputs \( y(-k) \) and past and present inputs \( x(-k) \). These coefficients can be determined using the ‘butter’ command in MATLAB. To verify the performance of the Butterworth filter, ‘freqz’ is used with these coefficients to produce a frequency response plot. A command written for MATLAB, `arraytoCformat`, is used to help copy the \( a_k \) and \( b_k \) coefficients to your C program. Verification of the implemented Butterworth filter should be conducted using the oscilloscope.

**An Example:**

Let’s assume our robot is using an ADC with a sampling rate of 100Hz and we want to design a 3rd order low pass filter with a cutoff frequency of 20Hz and implement it on our robot. We can design the filter using MATLAB by adhering to the following procedure:

1.) Determine the Butterworth filter coefficients using the MATLAB `butter` command:

\[
[b,a] = \text{butter}(3,.4,'low');
\]

Since we have a sample rate of 100Hz, the Nyquist Frequency is 50Hz and the cutoff frequency is 40% of the Nyquist Frequency.

\[
b =
\begin{bmatrix}
0.0985 & 0.2956 & 0.2956 & 0.0985
\end{bmatrix}
\]

\[
a =
\begin{bmatrix}
1.0000 & -0.5772 & 0.4218 & -0.0563
\end{bmatrix}
\]

2.) To see the transfer function of the Butterworth filter, use MATLAB’s `filterf` command:

\[
\text{filterf} = \text{tf}(b,a,.01);
\]

Again, we have a sample rate of 100Hz. The output of this command can be used as a reference for the difference equation below:

\[
y_k = b_0x_k + b_1x_{k-1} + b_2x_{k-2} + b_3x_{k-3} - a_1y_{k-1} - a_2y_{k-2} - a_3y_{k-3} \quad (3)
\]

\[
\text{filterf} =
0.09853 z^3 + 0.2956 z^2 + 0.2956 z + 0.09853
\]

\[
\frac{z^3 - 0.5772 z^2 + 0.4218 z - 0.0563}{z^3 + 0.2956 z^2 + 0.2956 z + 0.09853}
\]

Sample time: 0.01 seconds

3.) To see the expected frequency response of the filter, invoke MATLAB’s `freqz` command:

\[
\text{freqz}(b,a);
\]

The plot corresponding to the Butterworth filter above is included below:
Later we can compare this plot with the output on our oscilloscope once the filter has been implemented on the robot.

4.) Now we can start to implement the filter on our robot. To put the filter coefficients in an array format suitable for C code, invoke the following command:

arraytoCformat(b);
arraytoCformat(a);

Copy the output into the appropriate space in your Code Composer Studio Project to create global variables for the coefficient arrays.

```
float a[4]={ 1.0000000000e+00, -5.7724052481e-01, 4.2178704869e-01, -5.6297236492e-02};
```

5.) The last step is to code the filter in your Code Composer Studio Project. First, create global variables representing the filter inputs and outputs:

```
x[4] = {0,0,0,0};
y[4] = {0,0,0,0};
```
Then, within the void control(void) function we will add code similar to this:

```c
x[0] = adcB1; // adc voltage value is the current sample

// calculate filter value

dac1 = y[0];
dac2 = <whatever you want>;
writedac7564(dac1,dac2);

// save past states so ready for next sample
x[3] = x[2];
x[2] = x[1];
x[1] = x[0];

y[3] = y[2];
y[2] = y[1];
y[1] = y[0];
```

When you implement your filter, the equation to calculate the filter value should be done within a for loop (to mitigate mistakes when using filters of higher order). Also, save past states within a for loop.

Filter Assignment:

Using the robot’s sample rate of 1000Hz, design a simple 8-point averaging filter (take the average of the current reading and 7 past readings). At first, to check out that you understand the filtering equations, you do not need to use for loops to implement the filter, but to get checked off for the averaging filter you must change your code to use for loops to implement the averaging and the saving of past states. (The reason for this is what if I ask you to implement a 50-point averaging filter?) Use the “freqz” Matlab function and your b coefficients to plot a frequency (bode) plot of your filter. (What are the b coefficients of an averaging filter?) Verify that your actual filter works as shown in the Matlab frequency plot by picking a frequency in the plot and checking that magnitude and phase shift with the magnitude of phase shift of the sine wave on the Oscilloscope.

Once your averaging filter is checked off, design a 7th order low pass Butterworth filter with a 50Hz cutoff frequency using the “butter” command in Matlab. Just like in the averaging filter, verify that the frequency plot of the Butterworth filter matches the actual response of the filter by again picking a point and checking the magnitude and phase of the implemented filter using the oscilloscope.

References:
MATLAB – ‘butter’ command to create the coefficients of the butterworth filter.
MATLAB – ‘freqz’ command to create a frequency response plot of a digital filter.
Exercise 3: LABVIEW Application Starter Help

The goal of this assignment is to get you up to speed faster using LABVIEW to program a host interface to your robot. This quick example shows you how to both download data from the PC to the DSP and upload data from the DSP to the PC. It also shows you how to move an object inside a 2D picture item.

When you are finished with both the LABVIEW code and the DSP code you will be able to type values into the numeric controls and click SendDAC and the values you entered will be output on the DSP’s DACs 1 and 2. Also every 250ms the DSP will send your LABVIEW application the values of ADC1 and ADC2. The values will be displayed in two indicators and a circle inside the picture box will move to the x, y coordinates of (adc1, adc2). If the input into ADC1 and ADC2 varies, you will see the circle move on the screen.
1. Open up LABVIEW, create a blank project and add a single VI to the project. You may want to save this project and VI in your personal repository like we did in Lab 2.

2. Add a 2D Picture to your front panel and make its dimensions 800 by 800. Later we will be equating 50 pixels to 1 tile. This will then make your course 16 tiles by 16 tiles.

3. Reproduce the following block diagram along with the front panel above. It prints a square at the coordinates sent from Linux.
Code on the TMS320F28335 to check that your LABVIEW application is working correctly.

The code on the TMS320F28335 processor side to test that your LABVIEW application has been built correctly is mostly given to you. The OMAPL138’s DSP is already flashed with a program that takes care of receiving and sending data to and from LABVIEW. For sending data to LABVIEW you simply write the two ADC readings to two float variables already created for you called F28335_Extra1 and F28335_Extra2. So in the control() function, before the sendData() function call, assign F28335_Extra1 to the first ADC voltage reading and F28335_Extra2 to the second ADC reading.

To test if data is being received correctly from LABVIEW, write the transmitted float values to the two DAC channels and then monitor with the oscilloscope the change in the DAC voltage output. So in your code, instead of echoing the sampled ADC of filtered values to the two DAC channels, change the code in the control() function to pass the variables omap_dac1 and omap_dac2 to the writeDAC7564() function. omap_dac1 and omap_dac2 are variables already created for you and are assigned the value transmitted from LABVIEW in the given OMAPL138’s DSP code. To test this LABVIEW application, build and run your TMS320F28335 code. Then at the Linux command prompt run the executable “./LVDSPComm”. Finally run your LABVIEW application. If you input a .1 Hz Sine wave that varies from 0 to 2.5 into both ADC channels, you will see the square in the LABVIEW application moving along a straight diagonal line. Enter two numbers between 0 and 2.5 into your LABVIEW application’s numeric controls and click the Send button. Viewing on the oscilloscope, you should see the DAC voltage change to the sent values. Demonstrate this working to your instructor.

As a final Step

Add horizontal and vertical grid lines to the 2D Picture. The grid lines should be spaced every tile or 50 pixels. Think about where you should place the code to draw the grid lines. Do you want to draw them each time you receive new coordinates?

Exercise 4: Circuit Board Design and Fabrication using Eagle CAD

In the second week of lab you will become familiar with EagleCAD a circuit design software package, by going through a tutorial. For the take home exercise assigned below you will be asked to use EagleCAD to design your circuit and create the circuit board. You of course will not be asked to purchase your fabricated board from a printed circuit board company, but when you have completed the take home exercise you will have generated the files that would allow you to purchase the boards if you so desired. The tutorial is located at http://www.sparkfun.com/tutorials/108. You will be given instructions at the beginning of lab what is required for a check off of this section. To download the latest version of EagleCAD go to http://www.cadsoftusa.com/download-eagle/?language=en or http://www.cadsoft.de if the first link does not work.

TAKE HOME EXERCISE (must be completed to get a check off for this lab)

This take home exercise should be completed with your lab partner. When a section is complete, make an appointment with your instructor where both you and your partner will be asked to explain the work you have completed. This exercise has two sections. In the first section you are given a circuit board schematic and the C code that communicates with the IC’s of the circuit. Then by looking at the schematic and the source code, your assignment is to explain what the given C code is accomplishing. After getting this section of the take home exercise completed you will have a good idea of what is required for the second section. In the second section you will design your own circuit board.
You will create your circuit board in EagleCAD, develop the C code that communicates with one of the SPI chips of your design and, as in the first section, write an explanation of the C source code you developed.

**Section #1**

The below schematic is taken from the schematic of the robot’s processor board. It shows how the TMS320F28335 connects through its SPI serial port to 4 LS7366 ICs. The source file “28335_spi.c” contains the code for initializing the SPI port, initializing the 4 LS7366 chips and reading the LS7366 count values. For this assignment cut and paste the entire source file “28335_spi.c” into a Word document or whatever word processor you choose. There are four functions in this source file, init_SPI, SPI_RXint, start_SPI and writeDAC7564. You will NOT be explaining the writeDAC7564 so just leave it uncommented. Then for every 10 or so lines of code, excluding the writeDAC7564 function, I would like you to write a paragraph (not little short one liners and not line by line comments) explaining what this code is accomplishing.

**NOTES:**

1. You will want to look through the following User’s Guides and Data Sheets:
   a. LS7366 DataSheet, [http://coecsl.ece.illinois.edu/ge423/datasheets/LS7366.pdf](http://coecsl.ece.illinois.edu/ge423/datasheets/LS7366.pdf)
   b. SPI peripheral, [http://coecsl.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/SPI.pdf](http://coecsl.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/SPI.pdf)
   c. GPIO registers, Chapter 6, [http://coecsl.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/syscontrolandinterrupts.pdf](http://coecsl.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/syscontrolandinterrupts.pdf)
   d. GPIO MUX Tables, pages 75 and 76 [http://coecsl.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/syscontrolandinterrupts.pdf](http://coecsl.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/syscontrolandinterrupts.pdf)

2. EALLOW and EDIS are special macros that enable access to “protected” registers. Protected registers are important registers that if set in error cause the processor to be configured differently. These registers are usually only set once in main(). In the case of the given code, GPADIR and GPAMUX are protected.

3. In SPI_RXint(), case 6 and 7 are for the DAC7564, so you DO NOT have to comment these lines.

4. Start_SPI() is called every 1 millisecond inside the ADC’s ISR.

5. SPI_RXint() is the SPI’s receive ISR. Remember that the SPI serial port sends and receives at the same time so when the receive interrupt occurs the SPI has also completed transmitting.

6. Ignore the use of GPIO6 and GPIO19. GPIO6 is setup in init_SPI() for debug purposes only and GPIO19 is used by the DAC7564.
Section #2

Design a new “shield” circuit board, along with its interface C code, for the TMS320F28335. The current “shield” board on the robot’s processor board is the green board to the left of the color LCD. The design will actually be a partial design, because I only want you focusing on the SPI connections from the TMS320F28335 to the external chips. I do not want you designing the external connections to these chips. Even though this is a partial design, I still would like you to both create an EagleCAD schematic and an EagleCAD board layout. This will give you some extra experience working with circuit board fabrication. You are even going to go as far as sending your board files to a PCB company’s automated board file check website giving you some initial experience in ordering your designed board.

Assignment:

1. Design a new “shield” circuit board that interfaces the TMS320F28335 to five integrated circuits that use the SPI serial interface for communication. The five chips are the DS1394, MAX5436, MAX3100, MAX6627 and MCP23S08. So your new circuit board will have a real-time clock chip (DS1394), a digital potentiometer (MAX5436), an additional RS-232 UART serial port (MAX3100), a digital temperature sensor (MAX6627) and an 8-bit I/O Expander (MSP23S08). You will only be writing C code to interface with the MCP23S08 chip for this
assignment. I mention this here so you don’t spend a lot of time reading the datasheets of the other 4 chips. This is only a partial design, so you will only be connecting the SPI side of these chips to the TMS320F28335 and not worrying about the external connections of each of these chips. Of course this makes the board unusable (except for maybe the real-time clock chip), but the focus of this exercise is to give you some initial board fabrication experience. A note below specifies which pins to not wire on each of these chips.

For this design you can only use the TMS320F28335’s four SPI pins GPIO16_SPISIMOA_SV17, GPIO17_SPISIMOB_SV17, GPIO18_SPICLKA_SV17 and GPIO19_SPISTEA_SV17, and three general purpose I/O pins GPIO48_SV17, GPIO49_SV17 and GPIO58_MCLKRA_SV17.

Design the schematic for the circuit board in EagleCAD. All the needed chips and other components will be shown to you in the second week of Lab 4. (See also http://coecsl.ece.illinois.edu/ge423/EagleCAD%20Tips.pdf). When finished with the schematic, generate a “board” file for your design. In the board design software, move the ICs and components to desired locations and then have EagleCAD “auto-route” the traces for the board. Once your board file is complete, generate the fabrication files (gerber files). As a final step in the board design, send the fabrication files to www.4pcb.com to see if you have any errors that would cause problems in fabrication. Again see http://coecsl.ece.illinois.edu/ge423/EagleCAD%20Tips.pdf for steps in sending files to www.4pcb.com.

2. Write software for the TMS320F28335’s SPI serial port to interface with the MCP23S08 chip first without using the SPI’s RX interrupt. You will be polling on the FIFO status bits instead of using the RX interrupt. Create the following functions:

a. void Init_MCP23S08(unsigned int IODIRvalue, unsigned int IPOLvalue, unsigned int GPPUvalue, unsigned int IOCONvalue, unsigned int OLATvalue) — Write code to have this function initialize the MCP23S08 chip depending on what parameters are passed. Poll on the RXFFST status bits to determine when the SPI transmit and receive have completed before returning from this function. This way the function could be called in main() where interrupts are disabled.

b. void SetPortLatch(unsigned int byte) — Only the least significant 8 bits are sent to the MCP23S08’s OLAT register. Only the bits setup as outputs will be changed by writing to the OLAT register. Poll on the RXFFST status bits to determine when the SPI transmit and receive have completed before returning from this function.

c. unsigned int ReadPort(void) — Read and return the 8 bit value of the GPIO register. Poll on the RXFFST status bits to determine when the SPI transmit and receive have completed before returning from this function.

3. As a second coding exercise, insert code into the void SPI_RXint(void) interrupt function to read the 8 bit GPIO register after the four LS7366 chips have been read. In other words you are going to add to the LS7366 code you studied in Section #1 of this take home assignment in order to read the MCP23S08’s GPIO register. Here you will be using the SPI’s RX interrupt so you will not have to poll on the RXFFST status bits to determine when the SPI transmit and receive have finished.

4. As you did in section #1, write a paragraph every 10 or so lines of C code explaining what your code is performing.
5. Make an appointment with your instructor where both you and your partner will be asked to explain the work you have completed.

NOTES:

Datasheets

1. MCP23S08, read this pdf first. It has a link to the MCP23S08 datasheet.
   http://coecsl.ece.illinois.edu/ge423/datasheets/MCP23S08_Notes.pdf
2. 74F138, look at this decoder chip to help you with the 5 CS (one per chip) that you need to create given only one SS (SPISTEA) and three GPIO pins. http://coecsl.ece.illinois.edu/ge423/datasheets/en74f138.pdf

These remaining Datasheets are mainly for you to view the pin out of the chip’s pins and the purpose of each pin. See list below to find out which pins of each chip you will need to connect to. You are only going to wire these chips to the SPI interface and not write any source code for these chips.

5. MAX3100, http://coecsl.ece.illinois.edu/ge423/datasheets/MAX3100datasheet.pdf

What pins on each chip to ignore (leave unconnected) and which pins to use for this exercise

1. MCP23S08:
   Ignore: RESET, INT, GP0, GP1, GP2, GP3, GP4, GP5, GP6, GP7
   Connect: SCK, SI, SO, A1 (Connect to GND, Always Low), A0 (Connect to GND, Set Always Low), CS, VSS (GND), VDD (3.3V).

2. DS1394:
   Ignore: X1, X2, Vbackup, SQW/INT
   Connect: CS, GND, Vcc (3.3V), SCLK, DOUT, DIN

3. MAX5436:
   Ignore: VSS, L, W, H, VDD
   Connect: SCLK, DIN, CS, GND, VCC (3.3V)

4. MAX3100:
   Ignore: N.C., IRQ, SHDN, TX, RX, RTS, N.C., CTS, X1, X2
   Connect: DIN, DOUT, SCLK, CS, GND, VCC (3.3V)

5. MAX6627:
   Ignore: DXN, DXP, N.C.
   Connect: GND, VCC (3.3V), SDO, CS, SCK

6. F28335_SHIELD_CONNECTOR
   Ignore: SPI1_SCSN1_SV2 (OMAP SPI), SPI1_SIMO_SV2 (OMAP SPI), SPI1_SOMI_SV2 (OMAP SPI), SPI1_CLK_SV2 (OMAP SPI), CANTXA_SV16, CANRXA_SV16, GPIO63_SCITXDC_SV17, GPIO62_SCRXDC_SV17, GPIO59_MFSRA_SV17, +5V_SV17, ADCA0_SV19, ADCA1_SV19, ADCA2_SV19, ADCA3_SV19, ADCA4_SV19,
ADCA5_SV19, ADCA6_SV19, ADCA7_SV19, ADCB0_5V_SV18, ADCB1_5V_SV18, 
ADCB2_5V_SV18, ADCB3_5V_SV18, ADCB4_SV18, ADCB5_SV18, ADCB6_SV18, 
ADCB7_SV18.

Connect: GND_SV2, +3.3V_SV2, +3.3V1_SV16, GND1_SV16, GND2_SV16, 
+3.3V2_SV16, GPIO16_SPISIMOAOA_SV17, GPIO17_SPISOMIA_SV17, 
GPIO18_SPICLKA_SV17, GPIO19_SPISTEA_SV17, GPIO58_MCLKRA_SV17, 
GPIO49_SV17, GPIO48_SV17, GND_SV17, GND2_SV19, +3.3V2_SV19, +3.3V1_SV19, 
GND1_SV19, GND2_SV18, +3.3V2_SV18, +3.3V1_SV18, GND1_SV18.

In Lab Check Off:

1. Demonstrate your ADC to DAC echoing program and show a signal aliasing example.
2. Demonstrate both of your filters working.
3. Demonstrate your LABVIEW application working.
4. Finish the Eagle CAD tutorial for both the schematic editor and the board editor.
5. Section #1 and #2 of Take Home exercise.