Introduction:

Laboratory three is another extension of the lab #1 oscilloscope. The goal of this lab is to learn how to use the controller area network in order to extend the functionality of our oscilloscope.The objective is to use the LM3S2110 board to read in the input signal and use two 16-bit timers and an analog comparator in order to calculate the frequency of the signal and send it to the LM3S8962 board. The second part of this lab is to set up the pre-existing ADC on the RTOS to take advantage of the 8-sample hardware FIFO.

Discussion and Results:

Setup

In order to avoid using up more of the CPU load on the main board, we are using the LM3S2110 to calculate the frequency of the input signal. The signal is received by the comparator-0 input pin who converts it to a square wave. The square wave is output by the comparator output pin which we connect to the timer A input pin. Once the frequency calculations are finished, this value is sent as an unsigned long via the CAN interface to the LM3S8962. This board then interrupts on the reception of the message and reads in the unsigned long value. This unsigned long value is then converted into a string and displayed onto the OLED.

Analog Comparator Setup

The first step in calculating the frequency of the input signal is to convert it to a square wave. First of all, the analog comparator pins C0 and C0-o are configured in peripheral mode. By using a 1.5125V reference voltage we can create a square wave that oscillates around the

simulated 0V point of our sine wave. We then configure the comparator output pin to output the square wave so that it can be picked up by the capture timer.

Capture Timer and Periodic Timer

Timer 0, A and B, are our frequency calculation timers. The CCP0 pin is set up in peripheral mode to read in the square wave. Timer 0A is the capture timer, used to measure the period of the input square wave. It is setup to interrupt on the positive edge event of the input wave. This means that every interrupt captures the next positive edge of the waveform. We can use this to measure the period of the input signal. The following line shows how the period is captured:

121: accum_period += (last_capture - capture) & 0xffff;

The value *capture* is the count of the timer when it captured at the event. The value of *last_capture* is the captured value of the timer the last time it interrupted. By doing a bitwise and of the difference of these two values we can catch the overflow from when the two values are captured on separate slopes of the timer. This timer also keeps count of the amount of times the timer has interrupted which timer B uses in its calculations.

Timer 0B is a periodic timer set to run every 100ms. Inside of the timer B ISR we get the accumulated period, shown in the line above, along with the total number of periods captured in the accumulated period. Using these two values we can calculate the average period as the accumulated period divided by the total number of periods. From here in order to calculate the period we need to know the time of each timer tick.Since the timer runs at the CPU frequency, 25 MHz, we know that each timer A tick is 40ns. The period calculation is shown below:

*140: frequency = 1/(period * (0.00000004))*1000;*

We see here that the period (measured in clock ticks) multiplied by the time in seconds of each clock tick gives us the period in seconds. One over this period gives us the frequency in hertz. Finally we must multiply this value by 1000 in order to have three decimal points of precision when we transmit the unsigned integer on the CAN.

CAN Interface

The files network.c and network.h need to be integrated into the project in order to implement CAN communication. The main thing that has to be edited within network.h is selecting the Tx and Rx message IDs. Then, the computed frequency can be sent over the CAN interface as a 32-bit integer. Next, network.c and network.h have to be integrated into the code from the previous lab session's oscilloscope function in order to receive the data that is to be sent. The Rx and Tx message ID's must then be swapped. A CAN Hwi object is added and configured to call CAN_ISR(). Then, the function removes calls that configure the interrupt controller, and then customize NetworkRxCallback() to save the frequency in a global variable. Then, the received frequency is used as the parameter for the frequency display function

Frequency Display

The Capture ISR takes the previous and current captured Timer0A counts and calculates the difference between the two. This value is equivalent to the period. Once the period value has been accounted for, the frequency can then be solved using the equation:

```
frequency = 1/(period * (0.00000004))*1000;
```

Once this is accomplished, the frequency is set to be printed to the display screen. However, it is first broken up and stored as two separate parts: one part for the integer value of the frequency, and the other portion for the floating point decimal value. The code used to accomplish this is:

```
frequency = g_ulfrequency;
frequency_dp = frequency%1000;
frequency /= 1000;
```

Once the frequency is properly represented, it is then printed by using the usprint() function, printing first the integer value, then a decimal point, then the decimal point value.

ADC 8-Sample Hardware FIFO

In the final part of the Lab, the ADC's hardware FIFO was used to sample a waveform at 500 ksps or 500 Khz. In order to set up the FIFO, the 8 steps in the ADC sequence 0 were set to the same channel. Step 3 was configured as an interrupt using the code:

```
ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_CH0|ADC_CTL_IE);
```

Step 7 was also configured as an interrupt, and configured as the end of sequence, using the code:

ADCSequenceStepConfigure(ADC0_BASE, 0, 7, DC_CTL_CH0|ADC_CTL_IE|ADC_CTL_END);

Next, a while loop was placed in the code to read samples in from the FIFO. This while statement was set to continue looping until the ADC0_SSFSTAT0_R register empty flag becomes set. The ADC ISR is converted into a normal SYS/BIOS Hwi. The function kept track of errors by incrementing an error counter every time the FIFO overflowed. Once this occurred, the overflow condition was cleared, ensuring that the error counter did not increment continuously.

Conclusion:

This lab built off of the oscilloscope device that was first implemented in Lab 1, and then rebuilt in Lab 2. However, while the main purpose of Lab 2 was to take the original oscilloscope and change how it was run, (with an additional objective to add a new utility mode), Lab 3 used the original oscilloscope but focused mostly on building the components of a frequency meter that would use the signal from the oscilloscope to calculate and print the frequency value. This lab explored concepts that had been covered in the lectures previous to the lab's assignment. The increase in project scale and the new requirements made this lab more challenging than the previous lab. By the end of the lab period , the device was able to meet the requirements of the project guidelines, and ran without errors. So, this lab was indeed a success.