EENG 3910: Project V – Digital Signal Processing System Design

Assignment 3

Due 22 February 2016

Nathan Ruprecht

UNT – Electrical Engineering

***Introduction***

The purpose of assignment 3 is to get us even more involved in changing the code to see results from the board. We were given code as a start, but then told to change it to achieve a certain end result. Then giving us conditions and told to (sort of) starting from scratch.

***Results and Discussion***

Problem 1: Sample on-board temperature sensor

The code for 3\_1 was given to us and fairly simple to follow. It had the usual list of defining global variables and functions. In main, it starts with setting the system clock and initializing the LEDs, ADC, UART, and timer. Enabled the interrupts and timer then the initial text to the screen. Then we have the infinite while loop. It checks for a user command to change the LEDs or print out the temp.

Problem 2: Sample on-board temperature sensor based on timer

What I changed about 3\_1 was adding in the needed code to use timer1 and moved everything dealing with temperature reading into the timer1 ISR.

Problem 3: Sample on-board temperature sensor with FPU enabled

The difference between the given code of 3\_3 and 3\_1 is a couple of lines so the output of temperature would be a float. Two lines in main enable the FPU and allow stacking. Then where it deals with temperature calculations, changing some of the numbers to include a decimal point so the final answer becomes includes a decimal.

Problem 4: Sample on-board temperature sensor based on timer with FPU enabled

I pretty much merged 3\_2 into 3\_3. Taking the given code in problem 3 and adding in the needed code for timer1, then moving the given temperature code into the timer1 ISR.

Problem 5: Voltage meter with ADC

When the voltage reach 1.81V, the digital logic switched from 0 to 1. It didn’t take long to get the idea of what needed to be done, and writing the code for the general idea. What really took me the longest was figuring out to change the ADCSequenceStepConfigure function. Thankfully, Nick was there to point it out. It took a little digging to figure out the math in order to present the data on UART correctly. The main head scratcher was the ADC function since I never thought to look there for my troubleshoot problem.

***Summary and Conclusion***

This assignment seemed significantly harder than the previous. It was a good wake up call to the class that we’re really taking off as far as speed and information covered. It was a great lab to really start giving us less guidance. Problem 5 was good in that it mimics real world a little more in that a customer will just have a list of needs, but how we fulfill them is on us. Very basic example but I’m sure it’s how most labs will get to be in the future. It was also a good introduction to building a circuit to interact with our board as well as the UART all together.

Modified Source Code:

Changes to get Lab3\_2

**#define** TIMER1\_FREQ 0.5 // Frequency in Hz

**void** **Timer1\_ISR**(**void**);

**TimerEnable**(TIMER1\_BASE, TIMER\_A);

**void** **init\_timer**(**void**)

{

// Enable and configure timer peripheral.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1);

// Configure Timer0 as a 32-bit timer in periodic mode.

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_PERIODIC);

// Initialize timer load register.

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, sys\_clock/TIMER0\_FREQ -1);

// Configure Timer1 as a 32-bit timer in periodic mode.

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

// Initialize timer load register.

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, sys\_clock/TIMER1\_FREQ -1);

// Registers a function to be called when the interrupt occurs.

**IntRegister**(INT\_TIMER0A, Timer0\_ISR);

// The specified interrupt is enabled in the interrupt controller.

**IntEnable**(INT\_TIMER0A);

// Enable the indicated timer interrupt source.

**TimerIntEnable**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

// Registers a function to be called when the interrupt occurs.

**IntRegister**(INT\_TIMER1A, Timer1\_ISR);

// The specified interrupt is enabled in the interrupt controller.

**IntEnable**(INT\_TIMER1A);

// Enable the indicated timer interrupt source.

**TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

}

// Timer1 interrupt service routine

**void** **Timer1\_ISR**(**void**)

{

// Clear the timer interrupt.

**TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

**ADCProcessorTrigger**(ADC0\_BASE, ADC0\_SEQ\_NUM);

**while**(!**ADCIntStatus**(ADC0\_BASE, ADC0\_SEQ\_NUM, false)) {

}

**ADCIntClear**(ADC0\_BASE, ADC0\_SEQ\_NUM);

**ADCSequenceDataGet**(ADC0\_BASE, ADC0\_SEQ\_NUM, &cur\_temp);

cur\_temp\_C = (1475 - (2475\*cur\_temp)/4096)/10; // Convert to degree C.

cur\_temp\_F = (cur\_temp\_C\*9 + 160)/5; // Convert to degree F.

// Print current temperature to UART0

**UARTprintf**("\nTemp = %dC = %dF\n> ", cur\_temp\_C, cur\_temp\_F);

}

Changes to get Lab3\_4

**#define** TIMER1\_FREQ 0.5 // Frequency in Hz

**void** **Timer1\_ISR**(**void**);

**TimerEnable**(TIMER1\_BASE, TIMER\_A);

**void** **init\_timer**(**void**)

{

// Enable and configure timer peripheral.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1);

// Configure Timer0 as a 32-bit timer in periodic mode.

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_PERIODIC);

// Initialize timer load register.

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, sys\_clock/TIMER0\_FREQ -1);

// Configure Timer1 as a 32-bit timer in periodic mode.

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

// Initialize timer load register.

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, sys\_clock/TIMER1\_FREQ -1);

// Registers a function to be called when the interrupt occurs.

**IntRegister**(INT\_TIMER0A, Timer0\_ISR);

// The specified interrupt is enabled in the interrupt controller.

**IntEnable**(INT\_TIMER0A);

// Enable the indicated timer interrupt source.

**TimerIntEnable**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

// Registers a function to be called when the interrupt occurs.

**IntRegister**(INT\_TIMER1A, Timer1\_ISR);

// The specified interrupt is enabled in the interrupt controller.

**IntEnable**(INT\_TIMER1A);

// Enable the indicated timer interrupt source.

**TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

}

// Timer1 interrupt service routine

**void** **Timer1\_ISR**(**void**)

{

// Clear the timer interrupt.

**TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

**ADCProcessorTrigger**(ADC0\_BASE, ADC0\_SEQ\_NUM);

**while**(!**ADCIntStatus**(ADC0\_BASE, ADC0\_SEQ\_NUM, false)) {

}

**ADCIntClear**(ADC0\_BASE, ADC0\_SEQ\_NUM);

**ADCSequenceDataGet**(ADC0\_BASE, ADC0\_SEQ\_NUM, &cur\_temp);

cur\_temp\_C = 147.5 - (247.5\*cur\_temp)/4096; // Convert to degree C.

cur\_temp\_F = (cur\_temp\_C\*9 + 160)/5; // Convert to degree F.

**snprintf**(temp\_str, TEMP\_STR\_LEN, "%.1fC = %.1fF", cur\_temp\_C, cur\_temp\_F);

// Print current temperature to UART0

**UARTprintf**("\nTemp = %s\n> ", temp\_str);

}

Main changes to get Lab3\_5

**#define** TIMER0\_FREQ 2 // Freqency in Hz

**#define** TIMER1\_FREQ 1 // Frequency in Hz

**#define** UART0\_BAUDRATE 115200 // UART baudrate in bps

**#define** ADC0\_SEQ\_NUM 0 // ADC Sample Sequence Number

**#define** MAX\_ADC\_SAMP\_VAL 0x0FFF // max value for 12-bit ADC

**#define** DISP\_TEXT\_LINE\_NUM 4

**#define** DISP\_DATA\_STR\_LEN 20

// global variables

uint32\_t sig\_data=0, sig\_data\_d;

**float** sig\_data\_f;

**char** disp\_data\_str[DISP\_DATA\_STR\_LEN];

**void** **init\_timer**(**void**)

{

// Enable and configure timer peripheral.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1);

// Configure Timer0 as a 32-bit timer in periodic mode.

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_PERIODIC);

// Initialize timer load register.

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, sys\_clock/TIMER0\_FREQ -1);

// Configure Timer1 as a 32-bit timer in periodic mode.

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

// Initialize timer load register.

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, sys\_clock/TIMER1\_FREQ -1);

// Registers a function to be called when the interrupt occurs.

**IntRegister**(INT\_TIMER0A, Timer0\_ISR);

// The specified interrupt is enabled in the interrupt controller.

**IntEnable**(INT\_TIMER0A);

// Enable the indicated timer interrupt source.

**TimerIntEnable**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

// Registers a function to be called when the interrupt occurs.

**IntRegister**(INT\_TIMER1A, Timer1\_ISR);

// The specified interrupt is enabled in the interrupt controller.

**IntEnable**(INT\_TIMER1A);

// Enable the indicated timer interrupt source.

**TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

}

**void** **init\_ADC**(**void**)

{

// Enable and configure ADC0

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

**ADCSequenceConfigure**(ADC0\_BASE, ADC0\_SEQ\_NUM, ADC\_TRIGGER\_PROCESSOR, 0);

**ADCSequenceStepConfigure**(ADC0\_BASE, ADC0\_SEQ\_NUM, 0, ADC\_CTL\_IE|ADC\_CTL\_CH7|ADC\_CTL\_END);

**ADCSequenceEnable**(ADC0\_BASE, ADC0\_SEQ\_NUM);

}

// Timer1 interrupt service routine

**void** **Timer1\_ISR**(**void**)

{

// Clear the timer interrupt.

**TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

**ADCProcessorTrigger**(ADC0\_BASE, ADC0\_SEQ\_NUM);

**while**(!**ADCIntStatus**(ADC0\_BASE, ADC0\_SEQ\_NUM, false)) {

}

**ADCIntClear**(ADC0\_BASE, ADC0\_SEQ\_NUM);

**ADCSequenceDataGet**(ADC0\_BASE, ADC0\_SEQ\_NUM, &sig\_data);

sig\_data\_f = 3.3\*sig\_data/MAX\_ADC\_SAMP\_VAL;

**if**(**GPIOPinRead**(GPIO\_PORTE\_BASE, GPIO\_PIN\_4))

sig\_data\_d = 1;

**else**

sig\_data\_d = 0;

// Print data to UART0

**snprintf**(disp\_data\_str, DISP\_DATA\_STR\_LEN, "v=%.2f, d=%d", sig\_data\_f, sig\_data\_d);

**UARTprintf**("\n%s\n> ", disp\_data\_str);

}