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);

}
```