# TivaC Lab 5 CPE 403

### **Checklist for Lab 5**

- ☑ A text/word document of the initial code with comments
- ☑ In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also include the comments.
- ☑ Provide a permanent link to all main and dependent source code files only (name them as LabXX-TYY, XX-Lab# and YY-task#)Screenshots of debugging process along with pictures of actual circuit
- **☑** *Video link of demonstration.*

#### **Code for Experiment**

### Task 1:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h" // include ADC driver
#define TARGET IS BLIZZARD RB1 // Symbol for the API's in ROM.
#include "driverlib/rom.h"
#ifdef DEBUG
void__error__(char *pcFilename, uint32_t ui32Line) {
#endif
int main() {
        uint32_t ui32ADC0Value[4];
                                                             // ADC FIFO
        volatile uint32_t ui32TempAvg;
                                                    // Store average
        volatile uint32_t ui32TempValueC;
                                                    // Temperature in C
                                                     // Temperature in F
        volatile uint32_t ui32TempValueF;
         // Use a 40 MHz
         ROM_SysCtlClockSet(
                          SYSCTL SYSDIV 5 | SYSCTL USE PLL | SYSCTL OSC MAIN
                                            SYSCTL_XTAL_16MHZ);
         ROM_SysCtlPeripheralEnable( SYSCTL_PERIPH_ADC0); // enable the ADC0
         ROM_ADCHardwareOversampleConfigure( ADCO_BASE, 64); // configure hardware averaging
        // Configure ADCO sequencer to use sample sequencer 1, and have the processor trigger the
sequence
        ROM_ADCSequenceConfigure( ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
         // Configure each step.
         ROM_ADCSequenceStepConfigure( ADC0_BASE, 1, 0, ADC_CTL_TS);
        ROM_ADCSequenceStepConfigure( ADC0_BASE, 1, 1, ADC_CTL_TS);
ROM_ADCSequenceStepConfigure( ADC0_BASE, 1, 2, ADC_CTL_TS);
         // Sample temperature
         ROM_ADCSequenceStepConfigure( ADC0_BASE, 1, 3,
                          ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
         ROM_ADCSequenceEnable( ADC0_BASE, 1); // Enable ADC sequencer 1
                 ROM_ADCIntClear(ADC0_BASE, 1); // Clear ADC0 interrupt flag.
                 ROM_ADCProcessorTrigger(ADC0_BASE, 1); // Trigger ADC conversion.
                 while (!ADCIntStatus(ADC0_BASE, 1, false))
                          ; // wait for conversion to complete.
                 ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value); // store data into ui32ADC0Value
                 // Average read values, and round.
                 // Each Value in the array is the result of the mean of 64 samples.
                 ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2]
                                  + ui32ADC0Value[3] + 2) / 4;
                 ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096) / 10; // calc temp in C
                 ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5; // convert from C to F
        }
```

```
<u>Task 2:</u>
```

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw memmap.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h" // include ADC driver
#include "driverlib/gpio.h" // include GPIO
#define TARGET_IS_BLIZZARD_RB1 // Symbol for the API's in ROM.
#include "driverlib/rom.h"
#ifdef DEBUG
void__error__(char *pcFilename, uint32_t ui32Line) {
}
#endif
int main() {
        uint32_t ui32ADC0Value[4];
                                                  // ADC FIFO
        volatile uint32_t ui32TempAvg;
                                                  // Store average
        volatile uint32_t ui32TempValueC;
                                                  // Temperature in C
                                                  // Temperature in F
        volatile uint32_t ui32TempValueF;
        // Use a 40 MHz Clock
        ROM_SysCtlClockSet(
        SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
        // GPIO Enable on port F
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); // enable port F
        // Set LEDs as outputs
        ROM_GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
        GPIO PIN 1 | GPIO PIN 2 | GPIO PIN 3);
        ROM_SysCtlPeripheralEnable( SYSCTL_PERIPH_ADC0); // enable the ADC0 peripheral
        ROM_ADCHardwareOversampleConfigure( ADC0_BASE, 64);
                                                                 // hardware averaging (64 samples)
        // Configure ADC0 sequencer to use sample sequencer 2
        ROM_ADCSequenceConfigure( ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
        ROM_ADCSequenceStepConfigure( ADC0_BASE, 2, 0, ADC_CTL_TS);
        ROM_ADCSequenceStepConfigure( ADC0_BASE, 2, 1, ADC_CTL_TS);
        ROM_ADCSequenceStepConfigure( ADC0_BASE, 2, 2, ADC_CTL_TS);
        // Sample temperature sensor
        ROM_ADCSequenceStepConfigure( ADC0_BASE, 2, 3,
        ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
        ROM_ADCSequenceEnable( ADCO_BASE, 2); // Enable ADC sequencer 2
        while (1) {
                 ROM ADCIntClear(ADCO_BASE, 2); // Clear ADCO interrupt flag.
                 ROM_ADCProcessorTrigger(ADC0_BASE, 2); // Trigger ADC conversion.
                while (!ROM ADCIntStatus(ADC0 BASE, 2, false))
                                 // wait for conversion to complete.
                 ADCSequenceDataGet(ADCO_BASE, 2, ui32ADCOValue); // store converted data.
                 // Average read values, and round.
                 // Each Value in the array is the result of the mean of 64 samples.
```

```
ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2]
                                 + ui32ADC0Value[3] + 2) / 4;
                 ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096) / 10; // calc temp in C
                 ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
                 // light LED 1 if temp > 80 deg-F
                 if (ui32TempValueF > 80) {
                         // Turn on LED at PF1
                         ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);
                } else {
                         // Turn off all LEDs
                         ROM_GPIOPinWrite(GPIO PORTF BASE,
                         GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);
                }
        }
}
Task 3:
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
                                 // include ADC driver
#include "driverlib/gpio.h"
                                 // include gpio APIs
#define TARGET_IS_BLIZZARD_RB1 // Symbol for the API's in ROM.
#include "driverlib/rom.h"
#include "driverlib/timer.h"
                                          // timer library
#include "driverlib/interrupt.h"
                                          // interrupt APIs and macros
#include "inc/tm4c123gh6pm.h"
                                         // Define interrupt macros for device
void__error__(char *pcFilename, uint32_t ui32Line) {
#endif
void IntTimer0Handler(void); // timer handler prototype
int main() {
        uint32_t ui32Period; // Period of timer
        // Use 40 MHz clock
        ROM_SysCtlClockSet(
                         SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN
                                          | SYSCTL_XTAL_16MHZ);
        // GPIO configuration
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); // enable port F
        // Set LEDs as outputs
        ROM_GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
                         GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
        ROM_SysCtlPeripheralEnable( SYSCTL_PERIPH_ADC0); // enable the ADC0 peripheral
        ROM ADCHardwareOversampleConfigure( ADCO BASE, 64); // hardware averaging (64 samples)
        // Configure ADC0 sequencer to use sample sequencer 2, and have the processor trigger the
```

sequence.

```
ROM_ADCSequenceConfigure( ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
        // Configure each step.
        ROM_ADCSequenceStepConfigure( ADC0_BASE, 2, 0, ADC_CTL_TS);
        ROM_ADCSequenceStepConfigure( ADC0_BASE, 2, 1, ADC_CTL_TS);
        ROM_ADCSequenceStepConfigure( ADC0_BASE, 2, 2, ADC_CTL_TS);
        // Sample temperature sensor
        ROM_ADCSequenceStepConfigure( ADC0_BASE, 2, 3,
                        ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
        ROM_ADCSequenceEnable( ADC0_BASE, 2); // Enable ADC sequencer 2
        // Timer configuration
        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); // enable clock to TIMER0
        TimerConfigure(TIMER@ BASE, TIMER_CFG_PERIODIC); // Configure TIMER@ as 32 bit timer
        // Calculate and set delay
        ui32Period = SysCtlClockGet() / 2; // set frequency of interrupt to 2 Hz.
        TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period - 1);
        // Enable interrupt
        IntEnable(INT_TIMER0A); // enable vector associated with TIMER0A
        TimerIntEnable(TIMERO_BASE, TIMER_TIMA_TIMEOUT); // Enable event to generate interrupt
        IntMasterEnable();
                                     // Master int enable for all interrupts
        // Enable the timer
        TimerEnable(TIMER0_BASE, TIMER_A);
        while (1)
}
void IntTimerOHandler(void) {
        uint32_t ui32ADC0Value[4];
                                                 // ADC FIFO
        volatile uint32_t ui32TempAvg;
                                                // Store average
        volatile uint32 t ui32TempValueC;
                                                // Temperature in C
        volatile uint32_t ui32TempValueF;
                                                // Temperature in F
        // Clear the timer interrupt.
        ROM TimerIntClear(TIMERO BASE, TIMER TIMA TIMEOUT);
        ROM_ADCIntClear(ADC0_BASE, 2); // Clear ADC0 interrupt flag.
        ROM_ADCProcessorTrigger(ADC0_BASE, 2); // Trigger ADC conversion.
        while (!ROM_ADCIntStatus(ADC0_BASE, 2, false))
                ; // wait for conversion to complete.
        ADCSequenceDataGet(ADCO_BASE, 2, ui32ADCOValue); // store converted d/ Average read values, and
round.
        // Each Value in the array is the result of the average of 64 samples
        ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2]
                        + ui32ADC0Value[3] + 2) / 4;
        ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096) / 10; // calc temp in C
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
        // Read the current temperature. Light LED 1 if temp > 80 deg-F
        if (ui32TempValueF > 80) {
                ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);
        } else {
                ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
                                0);
        }
```

```
ROM_TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
}
```

## Video Link to Demo

Task 2: https://www.youtube.com/watch?v=hlk4ggL-7LQ

Task 3: https://www.youtube.com/watch?v=qmEybHfJPR4