**Lab07:**

Task01:

To calculate the temperature using the on-board temperature sensor, the code for task 2 of lab 5 was used. Then, each digit of the calculated temperature was converted to a character and displayed via the UART commands.

Code:

#include <stdint.h>

#include <stdbool.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/uart.h"

#include "inc/hw\_ints.h"

#include "driverlib/interrupt.h"

#include "driverlib/adc.h"

#define TARGET\_IS\_BLIZZARD\_RB1

#include "driverlib/rom.h"

#include "driverlib/timer.h"

#include <stdio.h>

int main(void)

{

// used to clear previous UART screen

int backspace;

SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

UARTConfigSetExpClk(UART0\_BASE, SysCtlClockGet(), 115200,

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE | UART\_CONFIG\_PAR\_NONE));

// Use to define the period of ADC conversion

uint32\_t ui32Period;

// Enable ADC Peripheral

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

// Average 64 values before putting in the ADC FIFO

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 64);

// Configure ADC sequencer to use ADC0 and SS1

ROM\_ADCSequenceConfigure(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);

// Setup steps for ADC sequencer. The temp value will show the average of 4 temp values

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 temp again and enable the interrupt flag to say that calculation has been completed

ROM\_ADCSequenceStepConfigure(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

// Enable ADC sequence

ROM\_ADCSequenceEnable(ADC0\_BASE, 1);

// Enable timer1

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER1);

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

// Clock period of 2Hz

ui32Period = SysCtlClockGet() / 2;

TimerLoadSet(TIMER1\_BASE, TIMER\_A, ui32Period -1);

// Enable timer1 overload as interrupts

IntEnable(INT\_TIMER1A);

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

IntMasterEnable();

// Start timer

TimerEnable(TIMER1\_BASE, TIMER\_A);

// Erase any previous "Fahrenheit: ##"

for (backspace = 0; backspace < 15; backspace++)

UARTCharPut(UART0\_BASE, '\b');

UARTCharPut(UART0\_BASE, 'F');

UARTCharPut(UART0\_BASE, 'a');

UARTCharPut(UART0\_BASE, 'h');

UARTCharPut(UART0\_BASE, 'r');

UARTCharPut(UART0\_BASE, 'e');

UARTCharPut(UART0\_BASE, 'n');

UARTCharPut(UART0\_BASE, 'h');

UARTCharPut(UART0\_BASE, 'e');

UARTCharPut(UART0\_BASE, 'i');

UARTCharPut(UART0\_BASE, 't');

UARTCharPut(UART0\_BASE, ':');

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, ' ');

while(1)

{

}

}

void Timer1IntHandler(void)

{

UARTCharPut(UART0\_BASE, '\b');

UARTCharPut(UART0\_BASE, '\b');

int Tens;

int Ones;

// Define variable to store data from the ADC FIFO

uint32\_t ui32ADC0Value[4];

// Define variables for use in later calculations

volatile uint32\_t ui32TempAvg;

volatile uint32\_t ui32TempValueC;

volatile uint32\_t ui32TempValueF;

// Clear ADC calculation complete interrupt

ROM\_ADCIntClear(ADC0\_BASE, 1);

// Start ADC conversion

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1);

// Wait for conversion to complete

while(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false))

{

}

// Copy ADC value

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value);

// Calculate average temperature reading

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

// Convert temp to Celsius

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

// Convert temp to Fahrenheit

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

Tens = ui32TempValueF - ui32TempValueF % 10;

Ones = ui32TempValueF - Tens;

UARTCharPut(UART0\_BASE, Tens / 10 + '0');

UARTCharPut(UART0\_BASE, Ones + '0');

}

Task02:

To implement the desired functionality, whenever a button is pressed, the program checks to see if any of the commands was inputted. Regardless, the program does not print the inputted character into the console. When the temperature input ‘T’ is inputted, the console will show the temperature measured by the ADC sequence every 0.5 seconds. When ‘t’ is inputted, the console will erase the shown temperature and stop the ADC sequence until the sequence is reinitiated.

Code:

#include <stdint.h>

#include <stdbool.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/uart.h"

#include "inc/hw\_ints.h"

#include "driverlib/interrupt.h"

#include "driverlib/adc.h"

#define TARGET\_IS\_BLIZZARD\_RB1

#include "driverlib/timer.h"

#include "driverlib/rom.h"

void UARTIntHandler(void)

{

uint32\_t ui32Status;

uint8\_t backspace;

ui32Status = UARTIntStatus(UART0\_BASE, true); //get interrupt status

UARTIntClear(UART0\_BASE, ui32Status); //clear the asserted interrupts

// Save the current UART input

char inputChar;

inputChar = UARTCharGetNonBlocking(UART0\_BASE);

// When 'R' is input, turn on red LED

if (inputChar == 'R')

{

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 2);

}

// When 'r' is input, turn off red LED

else if (inputChar == 'r')

{

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0);

}

// When 'G' is input, turn on green LED

else if (inputChar == 'G')

{

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 8);

}

// When 'g' is input, turn off green LED

else if (inputChar == 'g')

{

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0);

}

// When 'B' is input, turn on blue LED

else if (inputChar == 'B')

{

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 4);

}

// When 'b' is input, turn off blue LED

else if (inputChar == 'b')

{

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0);

}

// When 'T' is input, display temperature in console

else if (inputChar == 'T')

{

// Erase any previous "Fahrenheit: ##"

for (backspace = 0; backspace < 15; backspace++)

UARTCharPut(UART0\_BASE, '\b');

UARTCharPut(UART0\_BASE, 'F');

UARTCharPut(UART0\_BASE, 'a');

UARTCharPut(UART0\_BASE, 'h');

UARTCharPut(UART0\_BASE, 'r');

UARTCharPut(UART0\_BASE, 'e');

UARTCharPut(UART0\_BASE, 'n');

UARTCharPut(UART0\_BASE, 'h');

UARTCharPut(UART0\_BASE, 'e');

UARTCharPut(UART0\_BASE, 'i');

UARTCharPut(UART0\_BASE, 't');

UARTCharPut(UART0\_BASE, ':');

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, ' ');

// Start timer to begin ADC conversion process

TimerEnable(TIMER1\_BASE, TIMER\_A);

IntEnable(INT\_TIMER1A);

}

// When 't' is input, stop displaying temperature in console

else if (inputChar == 't')

{

// Stop timer to stop ADC conversion process

TimerDisable(TIMER1\_BASE, TIMER\_A);

IntDisable(INT\_TIMER1A);

// Erase any previous "Fahrenheit: ##"

for (backspace = 0; backspace < 16; backspace++)

{

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, '\b');

UARTCharPut(UART0\_BASE, '\b');

}

}

}

int main(void) {

SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

UARTConfigSetExpClk(UART0\_BASE, SysCtlClockGet(), 115200,

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE | UART\_CONFIG\_PAR\_NONE));

IntMasterEnable();

IntEnable(INT\_UART0);

UARTIntEnable(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT);

// Enable clock usage for pin F

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

// Set the 3 on-board LEDs to outputs

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);

// Use to define the period of ADC conversion

uint32\_t ui32Period;

// Enable ADC Peripheral

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

// Average 64 values before putting in the ADC FIFO

ROM\_ADCHardwareOversampleConfigure(ADC0\_BASE, 64);

// Configure ADC sequencer to use ADC0 and SS1

ROM\_ADCSequenceConfigure(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);

// Setup steps for ADC sequencer. The temp value will show the average of 4 temp values

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 temp again and enable the interrupt flag to say that calculation has been completed

ROM\_ADCSequenceStepConfigure(ADC0\_BASE,1,3,ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

// Enable ADC sequence

ROM\_ADCSequenceEnable(ADC0\_BASE, 1);

// Enable timer1

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER1);

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

// Clock period of 2Hz

ui32Period = SysCtlClockGet() / 2;

TimerLoadSet(TIMER1\_BASE, TIMER\_A, ui32Period -1);

// Enable timer1 overload as interrupts

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

IntMasterEnable();

while (1)

{

}

}

void Timer1IntHandler(void)

{

UARTCharPut(UART0\_BASE, '\b');

UARTCharPut(UART0\_BASE, '\b');

int Tens;

int Ones;

// Define variable to store data from the ADC FIFO

uint32\_t ui32ADC0Value[4];

// Define variables for use in later calculations

volatile uint32\_t ui32TempAvg;

volatile uint32\_t ui32TempValueC;

volatile uint32\_t ui32TempValueF;

// Clear ADC calculation complete interrupt

ROM\_ADCIntClear(ADC0\_BASE, 1);

// Start ADC conversion

ROM\_ADCProcessorTrigger(ADC0\_BASE, 1);

// Wait for conversion to complete

while(!ROM\_ADCIntStatus(ADC0\_BASE, 1, false))

{

}

// Copy ADC value

ROM\_ADCSequenceDataGet(ADC0\_BASE, 1, ui32ADC0Value);

// Calculate average temperature reading

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

// Convert temp to Celsius

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

// Convert temp to Fahrenheit

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

Tens = ui32TempValueF - ui32TempValueF % 10;

Ones = ui32TempValueF - Tens;

UARTCharPut(UART0\_BASE, Tens / 10 + '0');

UARTCharPut(UART0\_BASE, Ones + '0');

}