Argenis Jimenez Aguirre

CPE 403

TIVAC LAB 07

Task 01: Continuously display the temperature of the device (internal temperature sensor) on the a) hyperterminal, and b) GUI Composer (Temp Sensor) using a timer interrupt every 0.5 secs.

```
// Argenis Jimenez Aguirre
// CPE 403
// TIVAC Lab 07
// Task 01
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/rom.h"
#include "driverlib/adc.h"

    // ADC FIFO data stored in array
    uint32_t ui32ADC0Value[4];

    // Variables for Average, Celsius and Fahrenheit Temperatures
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;
    // Variables used to convert to chars
    volatile uint32_t nF, nC;
    // Variables used to display the chars
    char tempF[2];
    char tempC[2];

void UARTIntHandler(void)
{
    uint32_t ui32Status;

    ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status

    UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts

    while(UARTCharsAvail(UART0_BASE)) //loop while there are chars
```

```
    {
        char temp = UARTCharGet(UART0_BASE);
        UARTCharPut(UART0_BASE, temp);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
        SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
    }
}

int main(void) {
    // Run 40MHz System Clock
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);


    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // Enable ADC0 peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);

    // Use highest priority and set ADC0 and SS1
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

    // Sample steps 0-2 on Sequencer 1
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);

    // Configure Interrupt flag and sample final step in Sequencer 1
    ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

    // Enable Sequencer 1
    ADCSequenceEnable(ADC0_BASE, 1);

    // Enable RX and TX
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
```

```
78      GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
79
80      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED
81      GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2); //enable pin for LED PF2
82
83      // Set rate of data to 115200
84      UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
85          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
86
87      IntMasterEnable(); //enable processor interrupts
88      IntEnable(INT_UART0); //enable the UART interrupt
89      UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX interrupts
90
91      while(1)
92      {
93          // ADC conversion complete when status flag is cleared
94          ADCIntClear(ADC0_BASE, 1);
95          // Trigger ADC conversion
96          ADCProcessorTrigger(ADC0_BASE, 1);
97
98          // Wait for end of conversion
99          while(!ADCIntStatus(ADC0_BASE, 1, false))
.00         {
.01         }
.02         // Copy samples available in FIFO to buffer
.03         ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
.04         // Calculate Average, Celsius and Fahrenheit temperature
.05         ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
.06         ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
.07         ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
.08
.09         nF = ui32TempValueF; // Get temperature
.10         tempF[0] = nF/10 + 0x30; // Divide by 10 and add 48 to convert 1st int to char
.11         tempF[1] = ui32TempValueF%10 + 0x30; // Get remainder and add 48 to convert 2nd int to char
.12
.13         nC = ui32TempValueC; // Get temperature
.14         tempC[0] = nC/10 + 0x30; // Divide by 10 and add 48 to convert 1st int to char
.15         tempC[1] = ui32TempValueC%10 + 0x30; // Get remainder and add 48 to convert 2nd int to char
.16
```

```
117         SysCtlDelay(10000000);
118
119         // Display Temperature in F and C
120         UARTCharPut(UART0_BASE, 'T');
121         UARTCharPut(UART0_BASE, 'e');
122         UARTCharPut(UART0_BASE, 'm');
123         UARTCharPut(UART0_BASE, 'p');
124         UARTCharPut(UART0_BASE, ':');
125         UARTCharPut(UART0_BASE, ' ');
126         UARTCharPut(UART0_BASE, tempF[0]);
127         UARTCharPut(UART0_BASE, tempF[1]);
128         UARTCharPut(UART0_BASE, 'F');
129         UARTCharPut(UART0_BASE, ' ');
130         UARTCharPut(UART0_BASE, 'o');
131         UARTCharPut(UART0_BASE, 'r');
132         UARTCharPut(UART0_BASE, ' ');
133         UARTCharPut(UART0_BASE, tempC[0]);
134         UARTCharPut(UART0_BASE, tempC[1]);
135         UARTCharPut(UART0_BASE, 'C');
136         UARTCharPut(UART0_BASE, '\r');
137         UARTCharPut(UART0_BASE, '\n');
138
139         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
140         SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
141         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
142     }
143 }
```

Task 02: Interaction/User Interface: Develop a user interface using UART to perform the following:

Enter the cmd: R: Red LED, G: Green LED, B: Blue LED, T: Temperature:

Based on the command (cmd) the program should turn ON Red LED when R is entered in the terminal, etc. Command of ‘r’ will turn off the Red LED.

```c
// Argenis Jimenez Aguirre
// CPE 403
// TIVAC Lab 07
// Task 02
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/rom.h"
#include "driverlib/adc.h"

int main(void) {
    // Run 40MHz System Clock
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);


    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // Enable ADC0 peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);

    // Use highest priority and set ADC0 and SS1
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

    // Sample steps 0-2 on Sequencer 1
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);

    // Configure Interrupt flag and sample final step in Sequencer 1
    ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
```

```
40
41      // Enable Sequencer 1
42      ADCSequenceEnable(ADC0_BASE, 1);
43
44      // Enable RX and TX
45      GPIOPinConfigure(GPIO_PA0_U0RX);
46      GPIOPinConfigure(GPIO_PA1_U0TX);
47      GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
48
49      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED
50      GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3); //enable pin for LED PF2
51
52      // Set rate of data to 115200
53      UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
54          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
55
56      //IntMasterEnable(); //enable processor interrupts
57      IntEnable(INT_UART0); //enable the UART interrupt
58      UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX interrupts
59
60      // Display starting commands
61      UARTCharPut(UART0_BASE, 'E');
62      UARTCharPut(UART0_BASE, 'n');
63      UARTCharPut(UART0_BASE, 't');
64      UARTCharPut(UART0_BASE, 'e');
65      UARTCharPut(UART0_BASE, 'r');
66      UARTCharPut(UART0_BASE, ' ');
67      UARTCharPut(UART0_BASE, 'R');
68      UARTCharPut(UART0_BASE, ',');
69      UARTCharPut(UART0_BASE, ' ');
70      UARTCharPut(UART0_BASE, 'G');
71      UARTCharPut(UART0_BASE, ',');
72
73      UARTCharPut(UART0_BASE, ' ');
74      UARTCharPut(UART0_BASE, 'B');
75      UARTCharPut(UART0_BASE, ' ');
76      UARTCharPut(UART0_BASE, 'o');
77      UARTCharPut(UART0_BASE, 'r');
78      UARTCharPut(UART0_BASE, ' ');
```

```
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, '\r');

    while(1)
    {
        char cmd = UARTCharGet(UART0_BASE);

        if (cmd == 'T' || cmd =='t' )
        {
            // ADC FIFO data stored in array
            uint32_t ui32ADC0Value[4];

            // Variables for Average, Celsius and Fahrenheit Temperatures
            volatile uint32_t ui32TempAvg;
            volatile uint32_t ui32TempValueC;
            volatile uint32_t ui32TempValueF;
            // Variables used to convert to chars
            volatile uint32_t nF, nC;
            // Variables used to display the chars
            char tempF[2];
            char tempC[2];

            // ADC conversion complete when status flag is cleared
            ADCIntClear(ADC0_BASE, 1);
            // Trigger ADC conversion
            ADCProcessorTrigger(ADC0_BASE, 1);

            // Wait for end of conversion
            while(!ADCIntStatus(ADC0_BASE, 1, false))
            {
            }
            // Copy samples available in FIFO to buffer
            ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
            // Calculate Average, Celsius and Fahrenheit temperature
            ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
            ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
            ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

```

```
            nF = ui32TempValueF; // Get temperature
            tempF[0] = nF/10 + 0x30; // Divide by 10 and add 48 to convert 1st int to char
            tempF[1] = ui32TempValueF%10 + 0x30; // Get remainder and add 48 to convert 2nd int to char

            nC = ui32TempValueC; // Get temperature
            tempC[0] = nC/10 + 0x30; // Divide by 10 and add 48 to convert 1st int to char
            tempC[1] = ui32TempValueC%10 + 0x30; // Get remainder and add 48 to convert 2nd int to char

            // Display Temperature in F and C
            UARTCharPut(UART0_BASE, 'T');
            UARTCharPut(UART0_BASE, 'e');
            UARTCharPut(UART0_BASE, 'm');
            UARTCharPut(UART0_BASE, 'p');
            UARTCharPut(UART0_BASE, ':');
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, tempF[0]);
            UARTCharPut(UART0_BASE, tempF[1]);
            UARTCharPut(UART0_BASE, 'F');
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, 'o');
            UARTCharPut(UART0_BASE, 'r');
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, tempC[0]);
            UARTCharPut(UART0_BASE, tempC[1]);
            UARTCharPut(UART0_BASE, 'C');
        }
        // Turn ON Red LED with command 'R'
        else if (cmd == 'R')
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
            UARTCharPut(UART0_BASE, cmd);
        }
        // Turn ON Blue LED with command 'B'
        else if (cmd == 'B')
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
```

```
157             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
158             UARTCharPut(UART0_BASE, cmd);
159         }
160         // Turn ON Green LED with command 'G'
161         else if (cmd == 'G')
162         {
163             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
164             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
165             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 8);
166             UARTCharPut(UART0_BASE, cmd);
167         }
168         // Turn OFF Red LED with command 'r'
169         else if (cmd == 'r')
170         {
171             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
172             UARTCharPut(UART0_BASE, cmd);
173         }
174         // Turn OFF Blue LED with command 'b'
175         else if (cmd == 'b')
176         {
177             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
178             UARTCharPut(UART0_BASE, cmd);
179         }
180         // Turn OFF Green LED with command 'g'
181         else if (cmd == 'g')
182         {
183             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
184             UARTCharPut(UART0_BASE, cmd);
185         }
186
187         // Display greeting after action
188         UARTCharPut(UART0_BASE, '\r');
189         UARTCharPut(UART0_BASE, '\n');
190         UARTCharPut(UART0_BASE, 'E');
191         UARTCharPut(UART0_BASE, 'n');
192         UARTCharPut(UART0_BASE, 't');
```

```
193         UARTCharPut(UART0_BASE, 'e');
194         UARTCharPut(UART0_BASE, 'r');
195         UARTCharPut(UART0_BASE, ' ');
196         UARTCharPut(UART0_BASE, 'R');
197         UARTCharPut(UART0_BASE, ',');
198         UARTCharPut(UART0_BASE, ' ');
199         UARTCharPut(UART0_BASE, 'G');
200         UARTCharPut(UART0_BASE, ',');
201         UARTCharPut(UART0_BASE, ' ');
202         UARTCharPut(UART0_BASE, 'B');
203         UARTCharPut(UART0_BASE, ' ');
204         UARTCharPut(UART0_BASE, 'o');
205         UARTCharPut(UART0_BASE, 'r');
206         UARTCharPut(UART0_BASE, ' ');
207         UARTCharPut(UART0_BASE, 'T');
208         UARTCharPut(UART0_BASE, ':');
209         UARTCharPut(UART0_BASE, '\r');
210         UARTCharPut(UART0_BASE, '\n');
211     }
212 }
```