Phillip Sortomme

Date Submitted: 10/12/2018

Task 01:

Youtube Link: https://youtu.be/Z4FvgDUjaEM

```
#include <stdint.h>
#include <stdio.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"


// ===========================================================
// global variables
// ===========================================================

volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

// ===========================================================
// interrupt handlers
// ===========================================================

void Timer1IntHandler(void)
{
   uint32_t ui32ADC0Value[4];
   char tempconvert[10];
   int i;

   // Clear the timer interrupt
   ROM_TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
   ROM_ADCIntClear(ADC0_BASE, 2);
   ROM_ADCProcessorTrigger(ADC0_BASE, 2);

   while(!ROM_ADCIntStatus(ADC0_BASE, 2, false)){}
   ROM_ADCSequenceDataGet(ADC0_BASE, 2, ui32ADC0Value);
   ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +ui32ADC0Value[3] + 2)/4;
   ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
   ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

   i = 0;
   while (ui32TempValueF != 0)
   {
      tempconvert[i++] = (ui32TempValueF % 10) + '0';
      ui32TempValueF = ui32TempValueF / 10;
   }

// for the purpose of this task we only need the lower 16 bits of ui32TempValueF
// to output the temperature to UART.
   UARTCharPut(UART0_BASE, tempconvert[1]);
   UARTCharPut(UART0_BASE, tempconvert[0]);
   UARTCharPut(UART0_BASE, ' ');
```

```
    UARTCharPut(UART0_BASE, 'F');
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');

    // reload timer1 to begin 0.5s delay
    ROM_TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()/2);
}

// ============================================================
// main function
// ============================================================

int main(void)
{

    ROM_SysCtlClockSet(
          SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN
                | SYSCTL_XTAL_16MHZ);

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);

    ROM_UARTConfigSetExpClk(
          UART0_BASE, SysCtlClockGet(), 115200,
          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    // timer and ADC set up

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    ROM_TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 8);

    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);
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    ROM_ADCSequenceEnable(ADC0_BASE, 2);
    // set timer to 2Hz (0.5s)
    ROM_TimerLoadSet(TIMER1_BASE, TIMER_A, (SysCtlClockGet()/2));

    ROM_IntMasterEnable();
    ROM_IntEnable(INT_UART0);

    ROM_IntEnable(INT_TIMER1A);
    ROM_TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    ROM_UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

    ROM_UARTCharPut(UART0_BASE, 'T');
    ROM_UARTCharPut(UART0_BASE, 'e');
    ROM_UARTCharPut(UART0_BASE, 'm');
    ROM_UARTCharPut(UART0_BASE, 'p');
    ROM_UARTCharPut(UART0_BASE, 'e');
    ROM_UARTCharPut(UART0_BASE, 'r');
    ROM_UARTCharPut(UART0_BASE, 'a');
    ROM_UARTCharPut(UART0_BASE, 't');
    ROM_UARTCharPut(UART0_BASE, 'u');
    ROM_UARTCharPut(UART0_BASE, 'r');
```

```
    ROM_UARTCharPut(UART0_BASE, 'e');
    ROM_UARTCharPut(UART0_BASE, ':');
    ROM_UARTCharPut(UART0_BASE, '\n');
    ROM_UARTCharPut(UART0_BASE, '\r');

    ROM_TimerEnable(TIMER1_BASE, TIMER_A);

    while (1)
    {
    }
}
```

Task 02:

Youtube Link: https://youtu.be/P8RTJ-DoJho

```
#include <stdint.h>
#include <stdio.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"

// ============================================================
// global variables
// ============================================================
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;
volatile char command;
volatile bool UFlag;

// ============================================================
// interrupt handlers
// ============================================================
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
        command = ROM_UARTCharGetNonBlocking(UART0_BASE); // load the character into the command variable
        ROM_UARTCharPutNonBlocking(UART0_BASE, command); //echo character
        UFlag = true;
    }
}

// ============================================================
// main function
// ============================================================
int main(void)
{
    // ************ variable declaration and system clock setup************
```

```
    int i;      // used for temperature calculations
    uint32_t ui32ADC0Value[4];
    char tempconvert[10];
    char intro[66] =
          "Please enter a Command. Valid commands are: R, r, G, g, B, b, T.\n\r";
    char instruction[42] = "Valid commands are: R, r, G, g, B, b, T.\n\r";
    ROM_SysCtlClockSet(
          SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

// ************ peripheral enables and pin configuration************
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
    GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

// ************ ADC setup************
    ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 8);
    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);
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 3,
    ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
    ROM_ADCSequenceEnable(ADC0_BASE, 2);

// ************ Interrupt setup************
    ROM_IntMasterEnable();
    ROM_IntEnable(INT_UART0);

// ************ UART setup and initial message************
    ROM_UARTConfigSetExpClk(
          UART0_BASE, SysCtlClockGet(), 115200,
          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
    ROM_UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
    // print the introductory message to the terminal giving instructions on what to do
    i = 0;
    while (i < sizeof(intro))
       ROM_UARTCharPut(UART0_BASE, intro[i++]);
    UFlag = false;      // set the UART flag to 0

    //wait in the while loop for user input then execute the corresponding command
    while (1)
    {
       if (UFlag == true)
       {
          UFlag = false;  // reset the UFlag
          if (command == 'R')            // light up the red LED
             ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);
          else if (command == 'r')       // turn off the red LED
             ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
          else if (command == 'G')       // light up the green LED
             ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 8);
          else if (command == 'g')       // turn off the green LED
             ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
          else if (command == 'B')       // turn the the blue LED
             ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
          else if (command == 'b')    // turn of the, you guessed it, blue LED
             ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
          else if (command == 'T')
          {
             // collect the temperature data from ADC0 and output to terminal
```

```
            ROM_ADCIntClear(ADC0_BASE, 2);
            ROM_ADCProcessorTrigger(ADC0_BASE, 2);
            while (!ROM_ADCIntStatus(ADC0_BASE, 2, false))
            {
            }
            ROM_ADCSequenceDataGet(ADC0_BASE, 2, ui32ADC0Value);
            ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1]
                  + ui32ADC0Value[2] + ui32ADC0Value[3] + 2) / 4;
            ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096) / 10;
            ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
            //convert temperature reading into a char format for UART transmission
            i = 0;
            while (ui32TempValueF != 0)
            {
               tempconvert[i++] = (ui32TempValueF % 10) + '0';
               ui32TempValueF = ui32TempValueF / 10;
            }

            // for the purpose of this task we only need the lower 16 bits of ui32TempValueF to output the temperature to UART.
            ROM_UARTCharPut(UART0_BASE, tempconvert[1]);
            ROM_UARTCharPut(UART0_BASE, tempconvert[0]);
            ROM_UARTCharPut(UART0_BASE, ' ');
            ROM_UARTCharPut(UART0_BASE, 'F');
            ROM_UARTCharPut(UART0_BASE, '\n');
            ROM_UARTCharPut(UART0_BASE, '\r');
         }
         else
         { // you're doing it wrong... heres a reminder of what you should do.
            i = 0;
            while (i < sizeof(instruction))
               ROM_UARTCharPut(UART0_BASE, instruction[i++]);
         }
      }
   }
}
```
