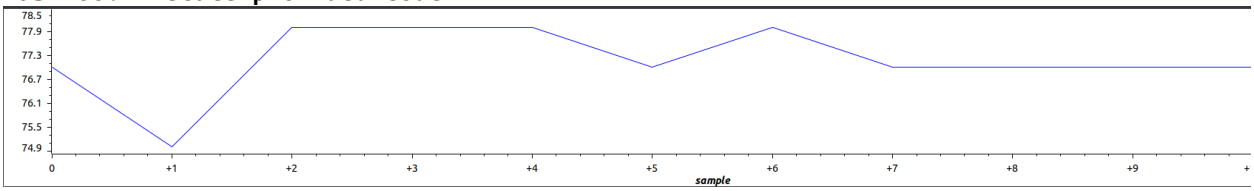
Brian West
Github root directory: https://github.com/westbrian2/Fall2019

Date Submitted:

Task 00: Execute provided code




```
//#define PART_TM4C123GH6PM


#include <stdint.h>
#include <stdbool.h>
#include "stdlib.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"
#include "inc/hw_pwm.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/udma.h"
#include "driverlib/pwm.h"
#include "driverlib/ssi.h"
```

```
#include "driverlib/systick.h"
#include "driverlib/adc.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"
#include <string.h>

//*****************************************************************************
//
// This function sets up UART0 to be used for a console to display information
// as the example is running.
//
//*****************************************************************************
void
InitConsole(void)
{
  //
  // Enable GPIO port A which is used for UART0 pins.
  // TODO: change this to whichever GPIO port you are using.
  //
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

  //
  // Configure the pin muxing for UART0 functions on port A0 and A1.
  // This step is not necessary if your part does not support pin muxing.
  // TODO: change this to select the port/pin you are using.
  //
  GPIOPinConfigure(GPIO_PA0_U0RX);
  GPIOPinConfigure(GPIO_PA1_U0TX);
```

```
    //
    // Enable UART0 so that we can configure the clock.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    //
    // Select the alternate (UART) function for these pins.
    // TODO: change this to select the port/pin you are using.
    //
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);
}

int main(){
    SysCtlClockSet(SYSCTL_SYSDIV_2_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    InitConsole();
     //
     // This array is used for storing the data read from the ADC FIFO. It
     // must be as large as the FIFO for the sequencer in use.  This example
     // uses sequence 3 which has a FIFO depth of 1.  If another sequence
```

```
// was used with a deeper FIFO, then the array size must be changed.
//
uint32_t ADCValues[1];


//
// These variables are used to store the temperature conversions for
// Celsius and Fahrenheit.
//
uint32_t TempValueC = 0 ;
uint32_t TempValueF = 0 ;



//
// Display the setup on the console.
//
UARTprintf("ADC ->\n");
UARTprintf("  Type: Internal Temperature Sensor\n");
UARTprintf("  Samples: One\n");
UARTprintf("  Update Rate: 250ms\n");
UARTprintf("  Input Pin: Internal temperature sensor\n\n");

//
// The ADC0 peripheral must be enabled for use.
//
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
SysCtlDelay(3);

//
// Enable sample sequence 3 with a processor signal trigger.  Sequence 3
```

```
// will do a single sample when the processor sends a singal to start the
// conversion.  Each ADC module has 4 programmable sequences, sequence 0
// to sequence 3.  This example is arbitrarily using sequence 3.
//
ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);

//
// Configure step 0 on sequence 3.  Sample the temperature sensor
// (ADC_CTL_TS) and configure the interrupt flag (ADC_CTL_IE) to be set
// when the sample is done.  Tell the ADC logic that this is the last
// conversion on sequence 3 (ADC_CTL_END).  Sequence 3 has only one
// programmable step.  Sequence 1 and 2 have 4 steps, and sequence 0 has
// 8 programmable steps.  Since we are only doing a single conversion using
// sequence 3 we will only configure step 0.  For more information on the
// ADC sequences and steps, reference the datasheet.
//
ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_IE |
                         ADC_CTL_END);

//
// Since sample sequence 3 is now configured, it must be enabled.
//
ADCSequenceEnable(ADC0_BASE, 3);

//
// Clear the interrupt status flag.  This is done to make sure the
// interrupt flag is cleared before we sample.
//
ADCIntClear(ADC0_BASE, 3);
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
//
// Sample the temperature sensor forever.  Display the value on the
// console.
//
while(1)
{
    //
    // Trigger the ADC conversion.
    //
    ADCProcessorTrigger(ADC0_BASE, 3);

    //
    // Wait for conversion to be completed.
    //
    while(!ADCIntStatus(ADC0_BASE, 3, false))
    {
    }

    //
    // Clear the ADC interrupt flag.
    //
    ADCIntClear(ADC0_BASE, 3);

    //
    // Read ADC Value.
    //
    ADCSequenceDataGet(ADC0_BASE, 3, ADCValues);
```

```
    //
    // Use non-calibrated conversion provided in the data sheet. I use floats in intermediate
    // math but you could use intergers with multiplied by powers of 10 and divide on the end
    // Make sure you divide last to avoid dropout.
    //
    TempValueC = (uint32_t)(147.5 - ((75.0*3.3 *(float)ADCValues[0])) / 4096.0);


    //
    // Get Fahrenheit value.  Make sure you divide last to avoid dropout.
    //
    TempValueF = ((TempValueC * 9) + 160) / 5;


    //
    // Display the temperature value on the console.
    //
    UARTprintf("Temperature = %3d*C or %3d*F\r", TempValueC,
          TempValueF);


    //
    // This function provides a means of generating a constant length
    // delay.  The function delay (in cycles) = 3 * parameter.  Delay
    // 250ms arbitrarily.
    //
    SysCtlDelay(80000000 / 12);
}
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Brian West
Github root directory: https://github.com/westbrian2/Fall2019

Task 01:

Youtube Link: https://youtu.be/I8tg5kaL8qw

Modified Code:

```
//#define PART_TM4C123GH6PM

#include <stdint.h>
#include <stdbool.h>
#include "stdlib.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"
#include "inc/hw_pwm.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/udma.h"
#include "driverlib/pwm.h"
#include "driverlib/ssi.h"
#include "driverlib/systick.h"
#include "driverlib/adc.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"
#include <string.h>

//*****************************************************************************
//
// This function sets up UART0 to be used for a console to display information
// as the example is running.
//
//*****************************************************************************
void
InitConsole(void)
{
    //
    // Enable GPIO port A which is used for UART0 pins.
```

```
    // TODO: change this to whichever GPIO port you are using.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Configure the pin muxing for UART0 functions on port A0 and A1.
    // This step is not necessary if your part does not support pin muxing.
    // TODO: change this to select the port/pin you are using.
    //
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);

    //
    // Enable UART0 so that we can configure the clock.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    //
    // Select the alternate (UART) function for these pins.
    // TODO: change this to select the port/pin you are using.
    //
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);
}

void configureTimer1A(void){

    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    IntMasterEnable();
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    //ui32period = (SysCtlClockGet()/2);
    TimerLoadSet(TIMER1_BASE, TIMER_A, 40000000-1);

    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER1_BASE,TIMER_TIMA_TIMEOUT);
    TimerEnable(TIMER1_BASE, TIMER_A);
}



int main(){

SysCtlClockSet(SYSCTL_SYSDIV_2_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
      InitConsole();

        // Display the setup on the console.
```

```
        //
        UARTprintf("ADC ->\n");
        UARTprintf("  Type: Internal Temperature Sensor\n");
        UARTprintf("  Samples: One\n");
        UARTprintf("  Update Rate: 250ms\n");
        UARTprintf("  Input Pin: Internal temperature sensor\n\n");

        //
        // The ADC0 peripheral must be enabled for use.
        //
        SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
        SysCtlDelay(3);
        ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
        ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_IE |
ADC_CTL_END);
        ADCSequenceEnable(ADC0_BASE, 3);
        ADCIntClear(ADC0_BASE, 3);

        configureTimer1A();
        while(1)
        {

        }



}

void Timer1IntHandler(void){
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    uint32_t ui32TempValueC = 0;
    uint32_t ui32TempValueF = 0;
    uint32_t ADCValues[1];

    ADCIntClear(ADC0_BASE, 3);
    ADCProcessorTrigger(ADC0_BASE, 3);

    while(!ADCIntStatus(ADC0_BASE,3,false)){
        ADCSequenceDataGet(ADC0_BASE, 3, ADCValues);
        ui32TempValueC = (uint32_t)(147.5 - ((75.0*3.3 *(float)ADCValues[0])) /
4096.0);
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
    }

    UARTprintf("Temperature = %3d*C or %3d*F\r", ui32TempValueC,ui32TempValueF);
}
```

Task 02:

Youtube Link: https://youtu.be/_5iSyULb90c

Modified Code:

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

```
#include <stdbool.h>
#include "stdlib.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"
#include "inc/hw_pwm.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/udma.h"
#include "driverlib/pwm.h"
#include "driverlib/ssi.h"
#include "driverlib/systick.h"
#include "driverlib/adc.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"
#include <string.h>

int32_t blue = (int32_t) 'b';
int32_t red = (int32_t) 'r';
int32_t green = (int32_t) 'g';
int32_t temp = (int32_t) 't';

uint8_t uiLED =0;
void UARTIntHandler(void)
{
    uint32_t ui32Status;
    int32_t command; //will hold the value of the userinput
    //uint8_t uiLED =0;

    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 = UARTCharGetNonBlocking(UART0_BASE); //echo character
        if(command == blue)
            uiLED = uiLED^4;
            //GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2^0x04); //toggle
LED
        if(command == red)
            uiLED = uiLED^2;
            //GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1^0x02); //toggle
LED
        if(command == green)
            uiLED = uiLED^8;
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
            //GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3^0x08); //toggle
LED
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3,uiLED);
        if(command == temp){
            uint32_t ui32TempValueC = 0;
            uint32_t ui32TempValueF = 0;
            uint32_t ADCValues[1];

            ADCIntClear(ADC0_BASE, 3);
            ADCProcessorTrigger(ADC0_BASE, 3);

            while(!ADCIntStatus(ADC0_BASE,3,false)){
            }
                ADCSequenceDataGet(ADC0_BASE, 3, ADCValues);
                ui32TempValueC = (uint32_t)(147.5 - ((75.0*3.3 *(float)ADCValues[0]))
/ 4096.0);
                ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

                UARTprintf("Temperature = %3d*C or %3d*F\r",
ui32TempValueC,ui32TempValueF);
        }

        SysCtlDelay(80000000 / 12);

    }
}

int main(void) {

    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3); //enable
pin for LED PF2

    UARTStdioConfig(0, 115200, 16000000);
   //UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
        //(UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    IntMasterEnable(); //enable processor interrupts
    IntEnable(INT_UART0); //enable the UART interrupt
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX
interrupts

    UARTCharPut(UART0_BASE, 'E');
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'x');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlDelay(3);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntClear(ADC0_BASE, 3);
    while (1) //let interrupt handler do the UART echo function
    {
//      if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE,
UARTCharGet(UART0_BASE));
    }

}
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Brian West
Github root directory: https://github.com/westbrian2/Fall2019

Task 03:

Youtube Link:

Modified Schematic (if applicable):

Modified Code:

```
// Insert code here
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.