```
*       This code was made to show a simple ADC read.
 *
 *      It was made from the example provided by TivaWare but it was a some
modifications
 * like the math
 *
 *
 * Lu?s Afonso
 *
 *
 */


#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>

volatile uint32_t millis=0;

void SycTickInt(){
  millis++;
}
void SysTickbegin(){
  SysTickPeriodSet(80000);
  SysTickIntRegister(SycTickInt);
  SysTickIntEnable();
   SysTickEnable();
}

void Wait(uint32_t time){
        uint32_t temp = millis;
        while( (millis-temp) < time){
```

```
        }
}

//*****************************************************************************
**
//
// 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_1
6MHZ);
          SysTickbegin();
          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;
            uint32_t TempValueF;


            //
            // 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);


            //ADCHardwareOversampleConfigure(ADC0_BASE,64);

            //
            // 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);

            //
            // 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);
            }



}
```

LED with PWM

```
#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/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 <string.h>

volatile uint32_t millis=0;
//about 2ms at 80Mhz
#define time 56666


void SycTickInt(){
  millis++;
}
void SysTickbegin(){
  SysTickPeriodSet(80000);
  SysTickIntRegister(SycTickInt);
  SysTickIntEnable();
   SysTickEnable();
}

void Wait(uint32_t _time){
        uint32_t temp = millis;
        while( (millis-temp) < _time){
        }
}


//PWM frequency in hz
uint32_t freq = 100000;

int main()
{

  //Set system clock to 80Mhz

SysCtlClockSet(SYSCTL_SYSDIV_2_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_1
6MHZ);
  SysTickbegin();

  uint32_t Period, dutyCycle;
  Period = SysCtlClockGet()/freq ;
  dutyCycle = Period-2;

  /*
    Configure PF1 as T0CCP1
    Configure PF2 as T1CCP0
    Configure PF3 as T1CCP1
  */

  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  SysCtlDelay(3);
  GPIOPinConfigure(GPIO_PF1_T0CCP1);
  GPIOPinConfigure(GPIO_PF2_T1CCP0);
  GPIOPinConfigure(GPIO_PF3_T1CCP1);
  GPIOPinTypeTimer(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
  //
```

```
  /*
    Configure timer 0 to split pair and timer B in PWM mode
    Set period and starting duty cycle.
  */
  SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
  SysCtlDelay(3);
  TimerConfigure(TIMER0_BASE, TIMER_CFG_SPLIT_PAIR|TIMER_CFG_B_PWM);
  TimerLoadSet(TIMER0_BASE, TIMER_B, Period -1);
  TimerMatchSet(TIMER0_BASE, TIMER_B, dutyCycle); // PWM


  /*
    Configure timer 1 to split pair and timer A and B in PWM mode
    Set period and starting duty cycle.
  */
  SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
  SysCtlDelay(3);
  TimerConfigure(TIMER1_BASE,
TIMER_CFG_SPLIT_PAIR|TIMER_CFG_A_PWM|TIMER_CFG_B_PWM);
  TimerLoadSet(TIMER1_BASE, TIMER_A, Period -1);
  TimerLoadSet(TIMER1_BASE, TIMER_B, Period -1);
  TimerMatchSet(TIMER1_BASE, TIMER_A, dutyCycle);
  TimerMatchSet(TIMER1_BASE, TIMER_B, dutyCycle);

  //Turn on both timers
  TimerEnable(TIMER0_BASE, TIMER_B);
  TimerEnable(TIMER1_BASE, TIMER_A|TIMER_B);

  int i;
  //Start by rising Red LED
  for(i=Period-2; i >  0;i--){
    TimerMatchSet(TIMER0_BASE, TIMER_B, i);
    SysCtlDelay(time);
  }
  while(1){

    //Blue brightness goes up - PF2
    for(i=Period-2; i > 0;i--){
      TimerMatchSet(TIMER1_BASE, TIMER_A, i);
      SysCtlDelay(time);
    }
    //Red brightness goes down - PF1
    for(i=1; i < Period-1; i++){
      TimerMatchSet(TIMER0_BASE, TIMER_B, i);
      SysCtlDelay(time);
    }
    //Green brightness goes up - PF3
    for(i=Period-2; i >  0;i--){
      TimerMatchSet(TIMER1_BASE, TIMER_B, i);
      SysCtlDelay(time);
    }
    //Blue brightness goes down - PF2
    for(i=1; i < Period-1; i++){
      TimerMatchSet(TIMER1_BASE, TIMER_A, i);
      SysCtlDelay(time);
    }
```

```
    //Red brightness goes up - PF1
    for(i=Period-2; i > 0;i--){
      TimerMatchSet(TIMER0_BASE, TIMER_B, i);
      SysCtlDelay(time);
    }
    //Green brightness goes down - PF3
    for(i=1; i < Period-1; i++){
      TimerMatchSet(TIMER1_BASE, TIMER_B, i);
      SysCtlDelay(time);
    }
  }

}
```

Button debouncing

```
#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/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 <string.h>


#define Premido 0
#define NaoPremido GPIO_PIN_4

volatile uint32_t millis=0;

void SycTickInt(){
  millis++;
}
void SysTickbegin(){
  SysTickPeriodSet(80000);
  SysTickIntRegister(SycTickInt);
  SysTickIntEnable();
  SysTickEnable();
```

```
}

void Wait(uint32_t time){
        uint32_t temp = millis;
        while( (millis-temp) < time){
        }
}


int main(){

SysCtlClockSet(SYSCTL_SYSDIV_2_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_1
6MHZ);
          SysTickbegin();

          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
          SysCtlDelay(3);

          GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);

GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_4,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_S
TD_WPU);

          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1);
          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_INT_PIN_2);
          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3);


          uint8_t state=0;
          while(1){
                 // GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4);  // = 000x 0000

                    uint32_t value=0;

                     value = GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4);

                     while((value & GPIO_PIN_4)==NaoPremido) {
// espera que seja premido
                         value = GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4);

                     }


                         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1, GPIO_PIN_1);
                         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2, 0);
                         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3, 0);

                        do{
                                value =
GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4);
                        }while((value & GPIO_PIN_4) == Premido);



                    while((value & GPIO_PIN_4)==NaoPremido){
                       value = GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4);
```

```
                    }

                         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1, 0);
                         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2, GPIO_PIN_2);
                         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3, 0);


                         do{
                                                          value =
GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4);
                                                    }while((value & GPIO_PIN_4) ==
Premido);

                          while((value & GPIO_PIN_4)==NaoPremido){
                                                    value =
GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4);



GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1, 0);

GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2, 0);

GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3, GPIO_PIN_3);



                                               do{
                                                          value =
GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4);
                                                          }while((value &
GPIO_PIN_4) == Premido);



          }

}
```

Write a program to generate a clock signal 1Hz at PF1, and a clock signal 2Hz at PF2.

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include <stdint.h>
#include <stdbool.h>

volatile uint8_t count = 0;
volatile uint32_t led_status = GPIO_PIN_1|GPIO_PIN_2;
```

```
void Timer0IntHandle(void){
        //Clear Int Flag
        TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
        led_status ^= GPIO_PIN_2;
        if (count == 1){
                count = 0;
                led_status ^= GPIO_PIN_1;
        } else {
                count += 1;
        }

        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2, led_status);
}

int main(void) {
        static uint32_t ulPeriod;

        // Set system clock
        SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

        // Enable Peripheral GPIO port F and Timer0
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

        // GPIO F1 output
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2);

        //Set up Timer 0
        ulPeriod = SysCtlClockGet()/1000;
        TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
        TimerLoadSet(TIMER0_BASE, TIMER_A, ulPeriod);
        TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
        IntEnable(INT_TIMER0A);
        IntMasterEnable();
        TimerEnable(TIMER0_BASE, TIMER_A);
        while(1){

        }
}
```

Write a program to count a 8bit number and display on 8 single LEDs at port PB[7:0].

```
#include <stdint.h>

#include <stdio.h>

#include <stdbool.h>

#include "inc/hw_memmap.h"

#include "inc/hw_types.h"

#include "driverlib/debug.h"

#include "driverlib/sysctl.h"
```

```
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"

unsigned int i;

int main(void) {
        // Set system clock
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

        // Enable Peripheral GPIO port B
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED

   GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7);
//enable pin PB0 -> PB7, Output mode
   while (1){
        printf ("Enter a hexadecimal number: ");
        scanf ("%x",&i);
     GPIOPinWrite(GPIO_PORTB_BASE,
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7, i);
   }
}
```

Write a program to measure the voltage values at the ADC0, and transmit them to UART after every second.

```
#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
```

```
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
#include "driverlib/uart.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/interrupt.h"
#include "inc/hw_ints.h"

unsigned long ulADC0Value[4];
char voltage_data[4];
volatile unsigned char output_data[16] = "The voltage is: ";
volatile unsigned long  ulADC0ValueAvg;
volatile unsigned long  ulVoltageValue;

void GetVoltage(void);
void SendVoltage(void);

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
	{
		UARTCharPutNonBlocking(UART0_BASE,UARTCharGetNonBlocking(UART0_BASE)); //echo
character
	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
	}
}
```

```
void SendVoltage(void){
	uint8_t i = 0;
	for(i=0;i<16;i++) {
		UARTCharPut(UART0_BASE, output_data[i]);
	}

	GetVoltage();
	UARTCharPut(UART0_BASE, voltage_data[0]);
	UARTCharPut(UART0_BASE, '.');
	UARTCharPut(UART0_BASE, voltage_data[1]);
	UARTCharPut(UART0_BASE, voltage_data[2]);
	UARTCharPut(UART0_BASE, voltage_data[3]);
	UARTCharPut(UART0_BASE, ulVoltageValue%10 + 48);
	UARTCharPut(UART0_BASE, ' ');
	UARTCharPut(UART0_BASE, 'V');
	UARTCharPut(UART0_BASE, '\n');
	UARTCharPut(UART0_BASE, '\r');
}
void GetVoltage(void){
	ADCIntClear(ADC0_BASE, 1);
	ADCProcessorTrigger(ADC0_BASE, 1);
	while(!ADCIntStatus(ADC0_BASE, 1, false))
	{
	}
	ADCSequenceDataGet(ADC0_BASE, 1, ulADC0Value);
	ulADC0ValueAvg = (ulADC0Value[0] + ulADC0Value[1] + ulADC0Value[2] + ulADC0Value[3] + 2)/4;
	ulVoltageValue = (3300 * ulADC0ValueAvg) / 4096;

	voltage_data[3] = ulVoltageValue%10 + 48;
	ulVoltageValue = ulVoltageValue/10 ;
```

```
        voltage_data[2] = ulVoltageValue%10 + 48;
        ulVoltageValue = ulVoltageValue/10 ;
        voltage_data[1] = ulVoltageValue%10 + 48;
        voltage_data[0] = ulVoltageValue/10 + 48;
}

int main(void) {
        // Set system clock
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

        // Enable Peripheral GPIO port F, ADC0, UART0 and Timer0
        SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

   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_2); //enable pin for LED PF2
   GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);

        // Set up ADC0
        SysCtlADCSpeedSet(SYSCTL_ADCSPEED_250KSPS);
        ADCSequenceDisable(ADC0_BASE, 1);
        ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
        ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH1);
        ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH1);
```

```
        ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH1);
        ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_CH1|ADC_CTL_IE|ADC_CTL_END);
        ADCSequenceEnable(ADC0_BASE, 1);

        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, 'S');
   UARTCharPut(UART0_BASE, 'T');
   UARTCharPut(UART0_BASE, 'A');
   UARTCharPut(UART0_BASE, 'R');
   UARTCharPut(UART0_BASE, 'T');
   UARTCharPut(UART0_BASE, '\n');
   UARTCharPut(UART0_BASE, '\r');

        while(1){
                SendVoltage();
                SysCtlDelay(SysCtlClockGet()/3);
        }
}
```