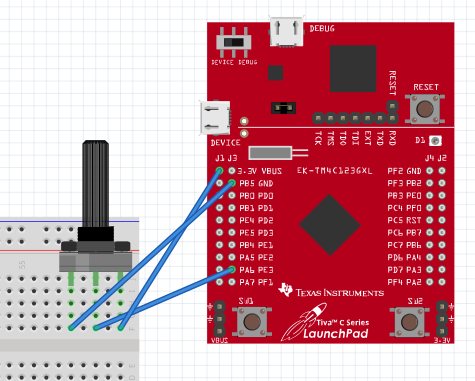
TITLE: TIVAC-TIRTOS-ASSIGNMENT

Midterm by Ron Joshua Recrio

GOAL:

* Get values using ADC on a potentiometer
* Print ADC values to UART
* LED on a PWM
* Switch read task to change duty cycle of PWM

Schematics:



DETAILED IMPLEMENTATION:

The first thing implemented is initializing the clock and all of the hardware components such as LED, switch, pwm, and the timer. Then the ADC and UART are initialized. Once those are made there are multiple ways to implement the following steps. The way chosen was to start with the configuration file and use the GUI to create a timer hardware interrupt. This will allows us to precisely determine which instance we are at in order to do the tasks 10 instances apart. The next thing is to create the tasks. The tasks are, read ADC, UART the adc values, and read the switch. Each of these tasks will be using a semaphore to synchronize. After setting the configuration, each task is made. The tasks are made to be as simple as possible (LED toggle only toggles LED). In the Timer interrupt, we turn on the LED based off of the duty cycle global variable. This variable is changed when the switch is pressed. This will read the ADC value and create a duty cycle value for the PWM. In this case we are simply using the timer as a toggle for our presumed PWM.

CODE:

}//----------------------------------------

// BIOS header files

//----------------------------------------

**#include** <xdc/std.h> //mandatory - have to include first, for BIOS types

**#include** <ti/sysbios/BIOS.h> //mandatory - if you call APIs like BIOS\_start()

**#include** <xdc/runtime/Log.h> //needed for any Log\_info() call

**#include** <xdc/cfg/global.h> //header file for statically defined objects/handles

//------------------------------------------

// TivaWare Header Files

//------------------------------------------

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_types.h"

**#include** "inc/hw\_memmap.h"

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

**#include** "driverlib/gpio.h"

**#include** "inc/hw\_ints.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/timer.h"

**#include** "driverlib/adc.h"

**#include** "driverlib/uart.h"

**#include** "driverlib/pin\_map.h"

**#include** "utils/uartstdio.h"

**#include** "utils/uartstdio.c"

**#include** "driverlib/pwm.h"

// Define

**#define** **PWM\_FREQUENCY** 55

//----------------------------------------

// Prototypes

//----------------------------------------

**void** **hardware\_init**(**void**);

**void** **Timer\_ISR**(**void**);

**void** **initADC**();

**void** **getADC3**(**void**);

**void** **InitConsole**(**void**);

**void** **UARTdisplayADC**(**void**);

//---------------------------------------

// Globals

//---------------------------------------

**volatile** **int16\_t** i16InstanceCount = 0;

**volatile** **int16\_t** dutyCycle = 0;

// 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];

//

// This variable is used to store the output of the ADC Channel 3

//

**uint32\_t** adcValue ;

//---------------------------------------------------------------------------

// main()

//---------------------------------------------------------------------------

**void** **main**(**void**)

{

**hardware\_init**();

**initADC**();

**InitConsole**();

BIOS\_start();

}

//---------------------------------------------------------------------------

// hardware\_init()

//

// inits GPIO pins for toggling the LED

//---------------------------------------------------------------------------

**void** **hardware\_init**(**void**)

{

**uint32\_t** ui32Period, ui32PWMClock, ui32Load;

//Set CPU Clock to 40MHz. 400MHz PLL/2 = 200 DIV 5 = 40MHz

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_XTAL\_16MHZ|SYSCTL\_OSC\_MAIN);

**SysCtlPWMClockSet**(SYSCTL\_PWMDIV\_64);

// Enable PWM and GPIOF

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_PWM1);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOD);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);

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

// Configure PWM to PF1

**GPIOPinTypePWM**(GPIO\_PORTD\_BASE, GPIO\_PIN\_0);

**GPIOPinConfigure**(GPIO\_PD0\_M1PWM0);

// Configure SW1

**GPIODirModeSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_4, GPIO\_DIR\_MODE\_IN);

**GPIOPadConfigSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_4, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

// Initialize

ui32PWMClock = **SysCtlClockGet**() / 64;

ui32Load = (ui32PWMClock / PWM\_FREQUENCY) - 1;

**PWMGenConfigure**(PWM1\_BASE, PWM\_GEN\_0, PWM\_GEN\_MODE\_DOWN);

**PWMGenPeriodSet**(PWM1\_BASE, PWM\_GEN\_0, ui32Load);

**PWMPulseWidthSet**(PWM1\_BASE, PWM\_OUT\_0, dutyCycle);

**PWMOutputState**(PWM1\_BASE, PWM\_OUT\_0\_BIT, true);

**PWMGenEnable**(PWM1\_BASE, PWM\_GEN\_0);

// Timer 2 setup code

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER2); // enable Timer 2 periph clks

**TimerConfigure**(TIMER2\_BASE, TIMER\_CFG\_PERIODIC); // cfg Timer 2 mode - periodic

ui32Period = (**SysCtlClockGet**() / 500); // period = CPU clk div 500 (1ms)

**TimerLoadSet**(TIMER2\_BASE, TIMER\_A, ui32Period); // set Timer 2 period

**TimerIntEnable**(TIMER2\_BASE, TIMER\_TIMA\_TIMEOUT); // enables Timer 2 to interrupt CPU

**TimerEnable**(TIMER2\_BASE, TIMER\_A); // enable Timer 2

}

//---------------------------------------------------------------------------

//---------------------------------------------------------------------------

// Timer ISR - called by BIOS Hwi (see app.cfg)

//

// Posts Swi (or later a Semaphore) to toggle the LED

//---------------------------------------------------------------------------

**void** **Timer\_ISR**(**void**)

{

**TimerIntClear**(TIMER2\_BASE, TIMER\_TIMA\_TIMEOUT); // reset

**if** (**GPIOPinRead**(GPIO\_PORTD\_BASE, GPIO\_PIN\_0))

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, 4);

}

**else**

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, 0);

}

// Task 1

**if** (i16InstanceCount == 10) {

Semaphore\_post(ADC3Sem);

}

// Task 2

**else** **if** (i16InstanceCount == 20) {

Semaphore\_post(UARTSem);

}

// Task 3

**else** **if**(i16InstanceCount == 30) {

Semaphore\_post(SW\_ReadSem);

i16InstanceCount = 0;

}

i16InstanceCount++;

}

**void** **changeDutyCycle**(**void**)

{

**while**(1)

{

Semaphore\_pend(SW\_ReadSem, BIOS\_WAIT\_FOREVER);

// From experimentation the max is ~2200 and min is ~150

// Cut off at 2000 and 200 for clean easy access to extreme values.

// Calculate a new duty cycle given adcValue

**if**(**GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_4) == 0x00)

{

dutyCycle = adcValue;

}

**PWMPulseWidthSet**(PWM1\_BASE, PWM\_OUT\_0, dutyCycle);

}

}

//

// Initializes ADC1 for use

//

**void** **initADC**() {

// The ADC3 peripheral must be enabled for use.

//

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC1);

**SysCtlDelay**(3);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOE);

**SysCtlDelay**(3);

**GPIOPinTypeADC**(GPIO\_PORTE\_BASE, GPIO\_PIN\_3); //Configure ADC pin: PE0

//

// 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**(ADC1\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

//

// Configure step 0 on sequence 3. Sample the ADC CHANNEL 3

// (PE0) 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**(ADC1\_BASE, 3, 0, ADC\_CTL\_CH3 | ADC\_CTL\_IE |

ADC\_CTL\_END);

//

// Since sample sequence 3 is now configured, it must be enabled.

//

**ADCSequenceEnable**(ADC1\_BASE, 3);

//

// Clear the interrupt status flag. This is done to make sure the

// interrupt flag is cleared before we sample.

///

**ADCIntClear**(ADC1\_BASE, 3);

}

//

// Gets value of ADC1 CH 3

//

**void** **getADC3**(**void**) {

**while**(1) {

Semaphore\_pend(ADC3Sem, BIOS\_WAIT\_FOREVER);

//

// Trigger the ADC conversion.

//

**ADCProcessorTrigger**(ADC1\_BASE, 3);

//

// Wait for conversion to be completed.

//

**while**(!**ADCIntStatus**(ADC1\_BASE, 3, false))

{

}

//

// Clear the ADC interrupt flag.

//

**ADCIntClear**(ADC1\_BASE, 3);

//

// Read ADC Value.

//

**ADCSequenceDataGet**(ADC1\_BASE, 3, ADCValues);

adcValue = ADCValues[0];

}

}

**void** **InitConsole**(**void**)

{

//

// Enable GPIO port A which is used for UART0 pins.

//

**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.

//

**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.

//

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

//

// Initialize the UART for console I/O.

//

**UARTStdioConfig**(0, 115200, 16000000);

}

//

// Displays ADC values

//

**void** **UARTdisplayADC**(**void**)

{

**while**(1)

{

Semaphore\_pend(UARTSem, BIOS\_WAIT\_FOREVER);

**UARTprintf**("ADC: %d Duty Cycle: %d%% \n", adcValue, (**int**)(((**float**)dutyCycle / 2500.0)\*100.0));

}

}

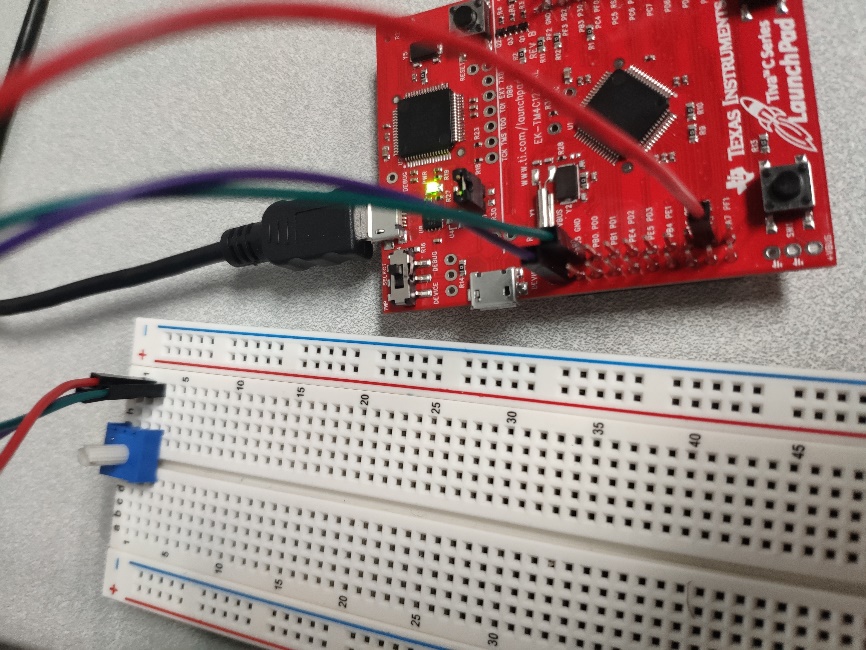
}

Video Link:

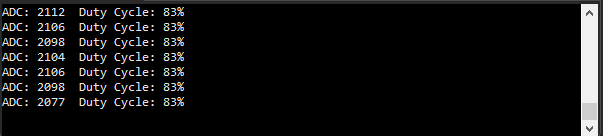
<https://youtu.be/Zjq5c8u9RKY>

Screenshots:

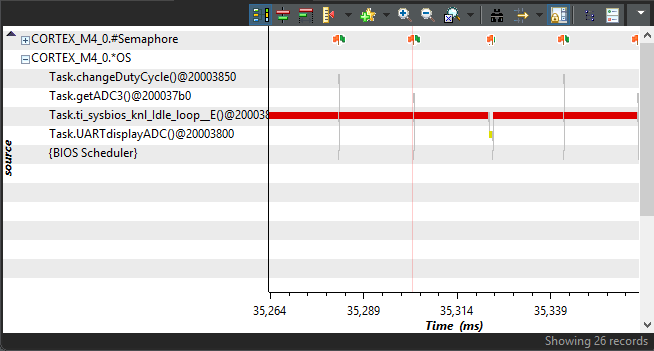
Circuit Implementation:



Output:



RTOS Analyzer:



Conclusions:

The project was successful. The three tasks were implemented accordingly. The only issue may be that the board may be running too slowly and could be increased by decreasing the ui32period. I am still a little confused on how to calculate the specific duty cycles for the PWM. I do not understand how to calculate the specific values to determine the high’s and low’s of the duty cycle.