CPE 403 ADV EMB SYS DES F 2016

TITLE: TIRTOS TIVAC Assignment

GOAL:

- Create ADC task to run every 10th instance of HWI
- Create UART diplay task to run every 20th instance of HWI
- Create Switch/Read Task to run every 30th instance of HWI
- Repeat the process above every 30 ms

DELIVERABLES:

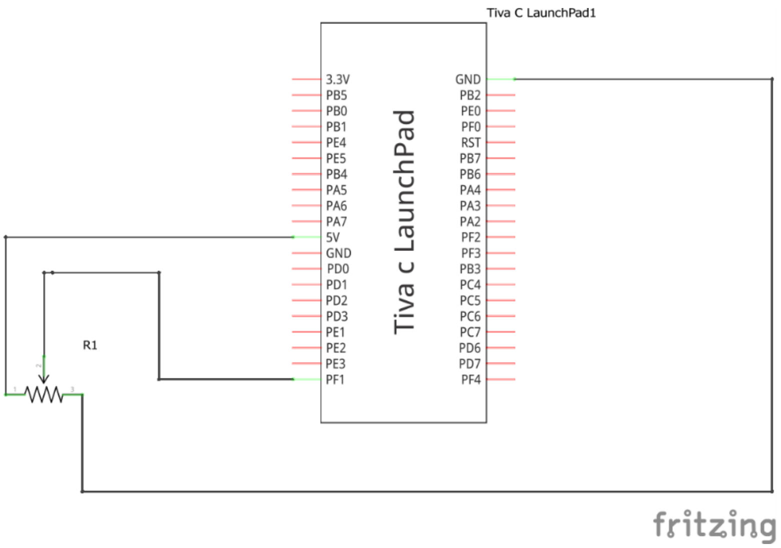
The project will show an LED that is affected by the PWM signal that takes the ADC value generated every 10th instance of the HWI. Also, a terminal will be shown that is connected to the same port as the TIVAC TM4C123GH6PM MCU to show the UART signals being transmitted/received. Every time SW0/SW1 is pressed, the duty cycle will change. Since the period of the PWM is very small, a logic analyzer will be shown which will convey when the switch is pressed to affect the PWM signal of the LED.

COMPONENTS:

TIVAC TM4C123GH6PM MCU

- Logic Analyzer
- Jumper Wires
- Potentiometer

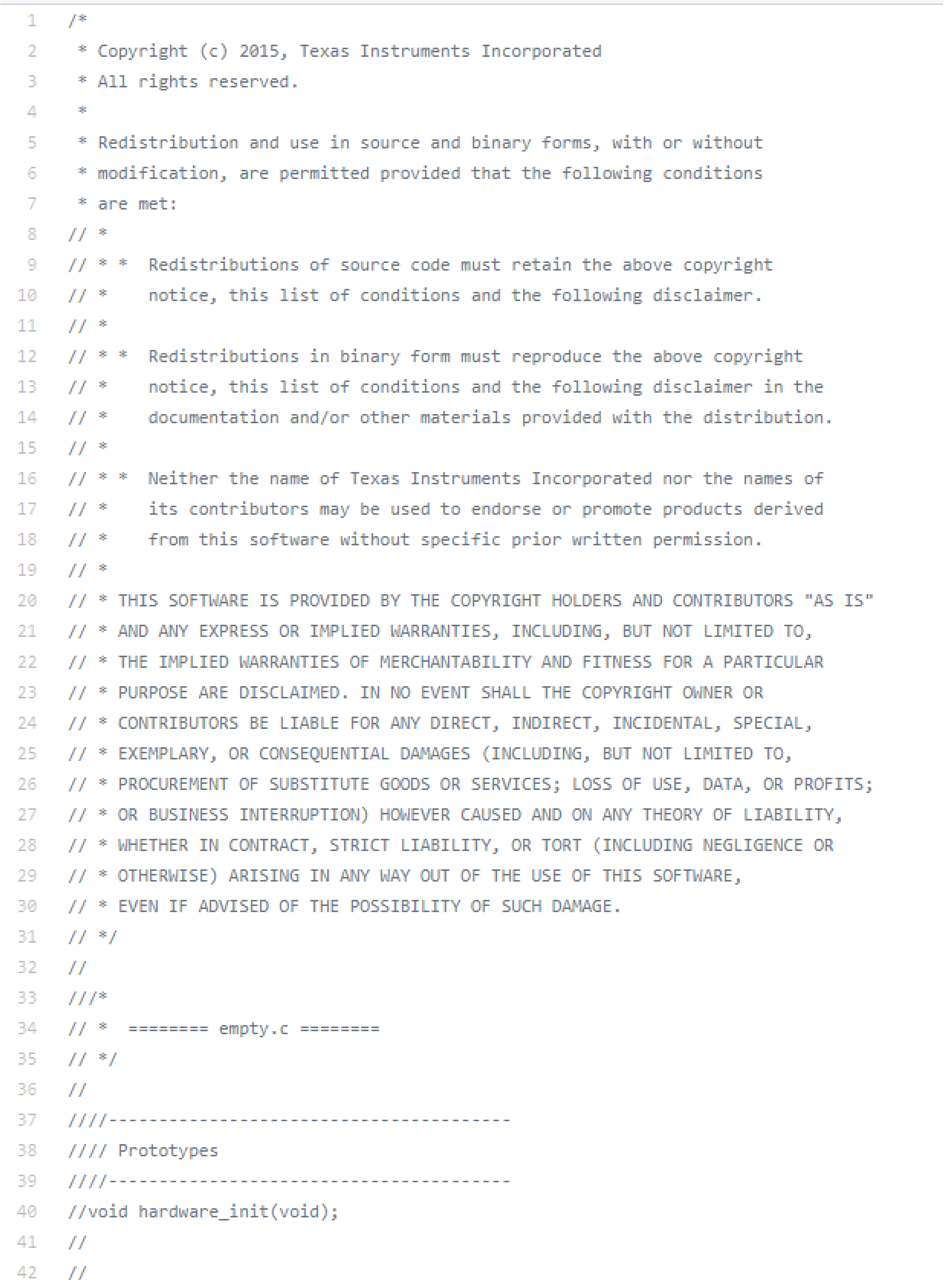
SCHEMATICS:



IIMPLEMENTATION:

UART and GPIO will be initialized as well as the ADC. This is the major initializations made for the assignment. The code below will show these initializations.

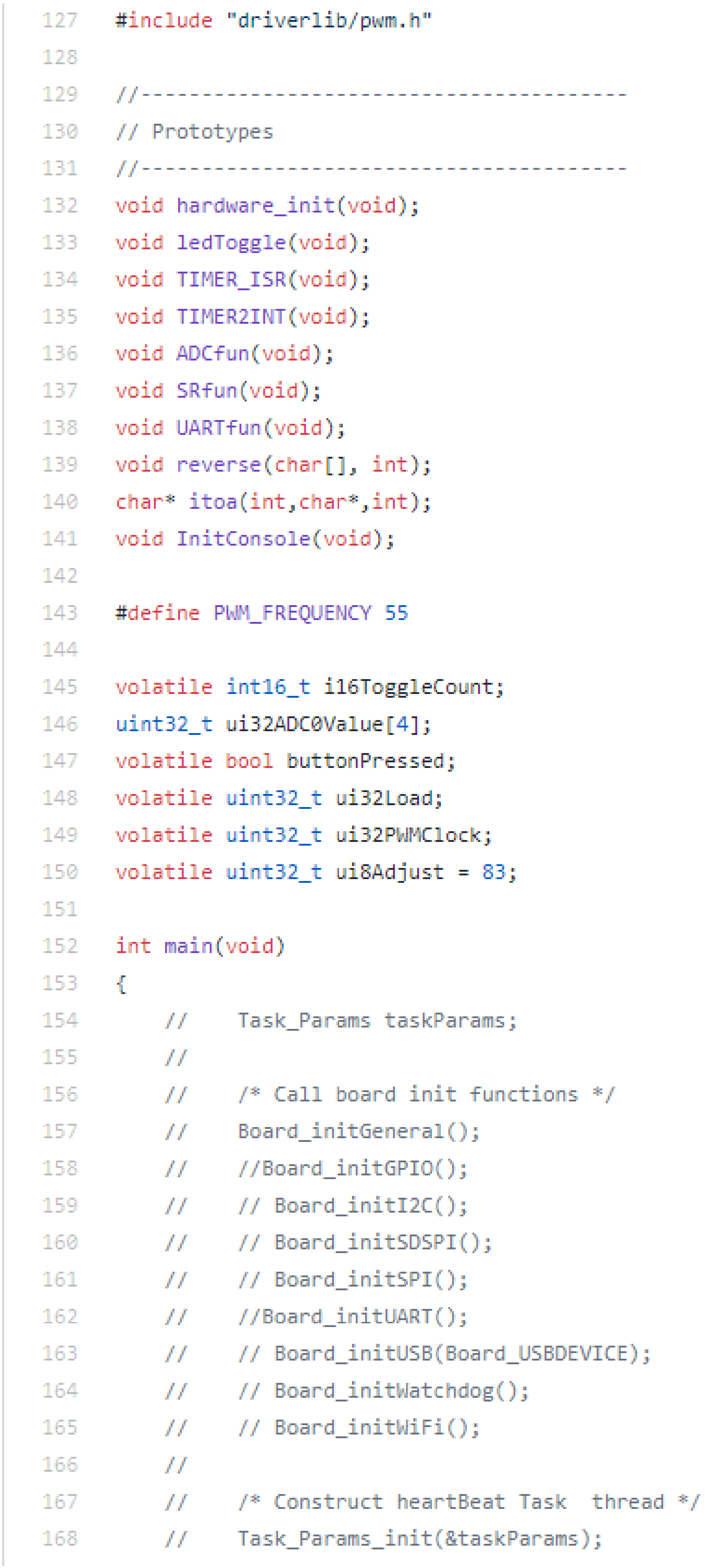
CODE:



```
/*
 * Copyright (c) 2015, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
// *
// * *  Redistributions of source code must retain the above copyright
// *    notice, this list of conditions and the following disclaimer.
// *
// * *  Redistributions in binary form must reproduce the above copyright
// *    notice, this list of conditions and the following disclaimer in the
// *    documentation and/or other materials provided with the distribution.
// *
// * *  Neither the name of Texas Instruments Incorporated nor the names of
// *    its contributors may be used to endorse or promote products derived
// *    from this software without specific prior written permission.
// *
// * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
// * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
// * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
// * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
// * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
// * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
// * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
// */
//
///*
// *  ======== empty.c ========
// */
//
////----------------------------------------
//// Prototypes
////----------------------------------------
//void hardware_init(void);
//
//
```

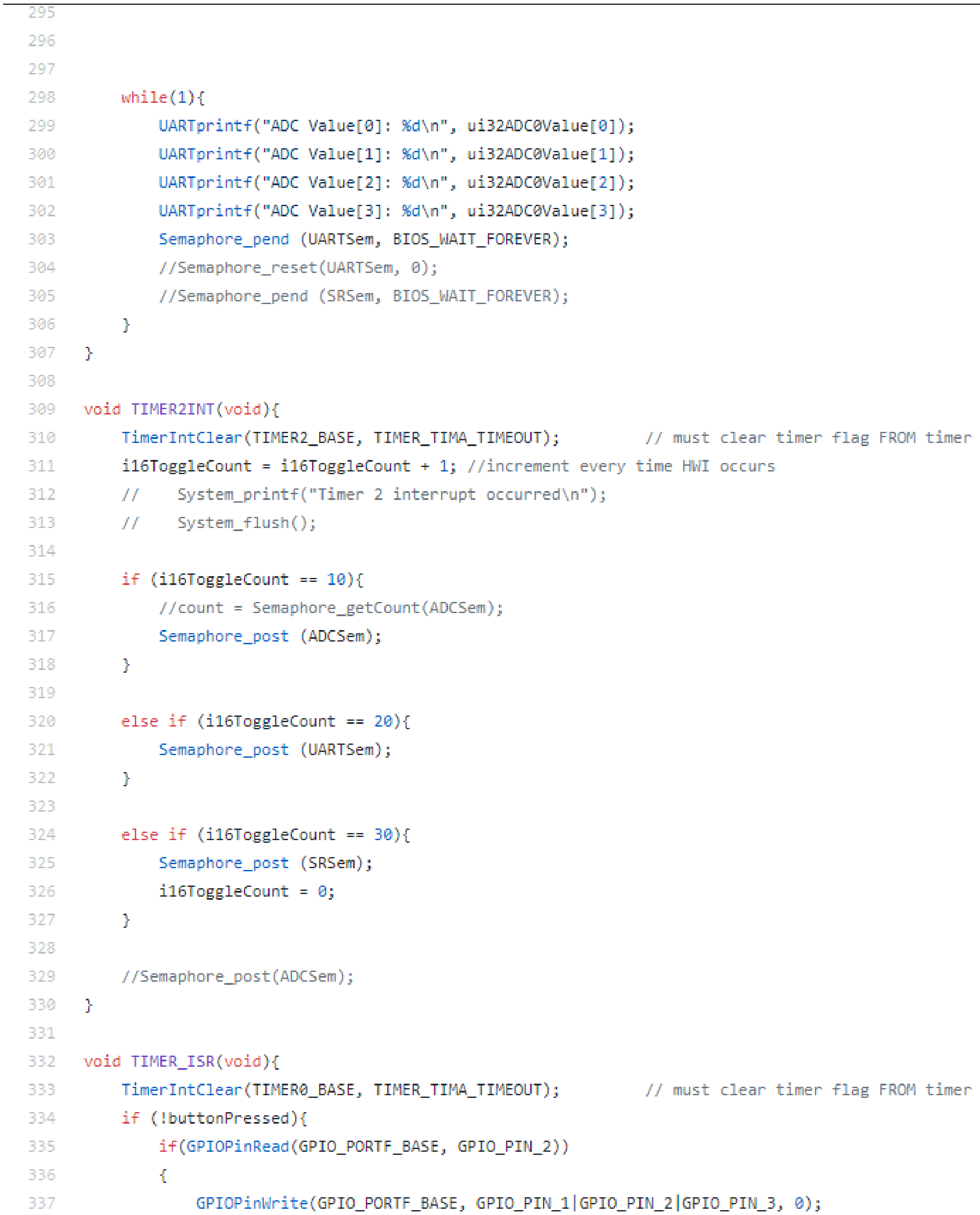
```
//#include <stdbool.h>
//
////----------------------------------------
//// BIOS header files
////----------------------------------------
//#include <xdc/std.h>                        //mandatory - have to include first, for BIOS types
//#include <ti/sysbios/BIOS.h>                //mandatory - 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
//
//
///* XDCtools Header files */
//#include <xdc/runtime/System.h>
//
///* BIOS Header files */
//#include <ti/sysbios/knl/Task.h>
//
///* TI-RTOS Header files */
//#include <ti/drivers/GPIO.h>
//// #include <ti/drivers/I2C.h>
//// #include <ti/drivers/SDSPI.h>
//// #include <ti/drivers/SPI.h>
//#include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>
// #include <ti/drivers/WiFi.h>
//
///* Board Header file */
//#include "Board.h"
//
//#include "driverlib/adc.h"
//#include "inc/hw_memmap.h"
//#include "driverlib/sysctl.h"
//#include "driverlib/timer.h"
//#include "driverlib/interrupt.h"
//
//
//#define TASKSTACKSIZE   512
//
//Task_Struct task0Struct;
//Char task0Stack[TASKSTACKSIZE];
//
///*
```

```c
// *  ======== heartBeatFxn ========
// *  Toggle the Board_LED0. The Task_sleep is determined by arg0 which
// *  is configured for the heartBeat Task instance.
// */
//Void heartBeatFxn(UArg arg0, UArg arg1)
//{
//    while (1) {
//        Task_sleep((UInt)arg0);
//        GPIO_toggle(Board_LED0);
//    }
//}
//
///*
// *  ======== main ========
// */

//----------------------------------------
// 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 "utils/uartstdio.h"
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"
```

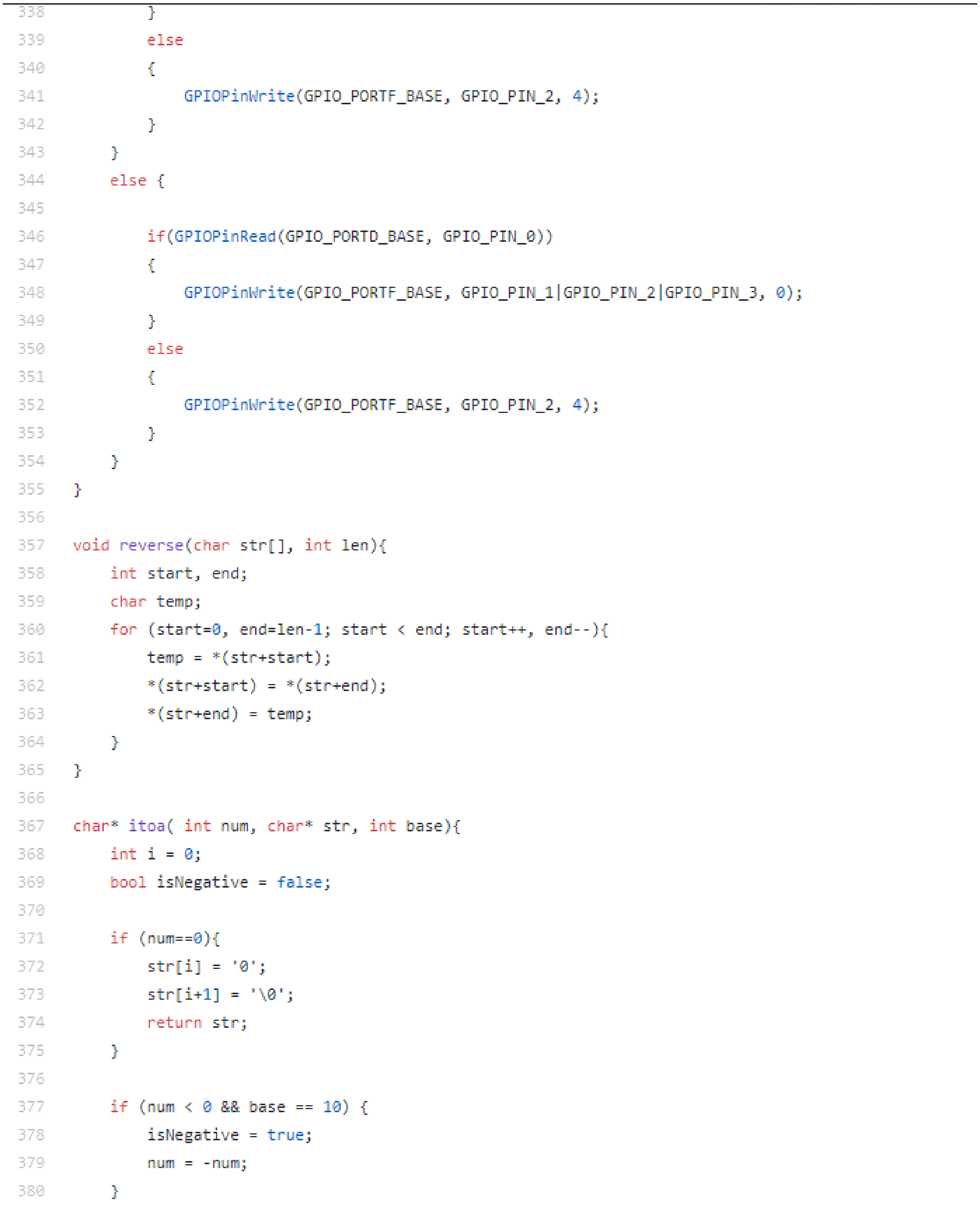


```c
#include "driverlib/pwm.h"

//---------------------------------------
// Prototypes
//---------------------------------------
void hardware_init(void);
void ledToggle(void);
void TIMER_ISR(void);
void TIMER2INT(void);
void ADCfun(void);
void SRfun(void);
void UARTfun(void);
void reverse(char[], int);
char* itoa(int,char*,int);
void InitConsole(void);

#define PWM_FREQUENCY 55

volatile int16_t i16ToggleCount;
uint32_t ui32ADC0Value[4];
volatile bool buttonPressed;
volatile uint32_t ui32Load;
volatile uint32_t ui32PWMClock;
volatile uint32_t ui8Adjust = 83;

int main(void)
{
    //    Task_Params taskParams;
    //
    //    /* Call board init functions */
    //    Board_initGeneral();
    //    //Board_initGPIO();
    //    // Board_initI2C();
    //    // Board_initSDSPI();
    //    // Board_initSPI();
    //    //Board_initUART();
    //    // Board_initUSB(Board_USBDEVICE);
    //    // Board_initWatchdog();
    //    // Board_initWiFi();
    //
    //    /* Construct heartBeat Task  thread */
    //    Task_Params_init(&taskParams);
    //    taskParams.arg0 = 1000;
    //    taskParams.stackSize = TASKSTACKSIZE;
    //    taskParams.stack = &task0Stack;
    //    Task_construct(&task0Struct, (Task_FuncPtr)heartBeatFxn, &taskParams, NULL);
    //
    //    /* Turn on user LED */
    //    GPIO_write(Board_LED0, Board_LED_ON);
    buttonPressed = false;
    hardware_init();
    //
    //    System_printf("Starting the example\nSystem provider is set to SysMin. "
    //            "Halt the target to view any SysMin contents in ROV.\n");
    //    /* SysMin will only print to the console when you call flush or exit */
    //    System_flush();

    /* Start BIOS */
    BIOS_start();

}

//---------------------------------------------------------------------------
// hardware_init()
//
// inits GPIO pins for toggling the LED
//---------------------------------------------------------------------------
void hardware_init(void)
{
    uint32_t ui32Period;

    i16ToggleCount = 0;

    // Board_initUART();

    //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);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

    GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0); //PD0 PWM pin
    GPIOPinConfigure(GPIO_PD0_M1PWM0);
```

```c

    // ADD Tiva-C GPIO setup - enables port, sets pins 1-3 (RGB) pins for output
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0);

    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0 , GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

    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, (ui32ADC0Value[0]/200 * ui32Load)/1000);
    PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
    PWMGenEnable(PWM1_BASE, PWM_GEN_0);

    // Turn on the LED
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 4);

    //initialize ADC
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);

    ADCSequenceEnable(ADC0_BASE, 1);

    // 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() / 1000);                   // period = CPU clk div 1000 (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

    InitConsole();
    UARTprintf("WORKING!");
}



void ADCfun(void){
    while(1){

        ADCIntClear(ADC0_BASE, 1);
        ADCProcessorTrigger(ADC0_BASE, 1);

        while (!ADCIntStatus(ADC0_BASE, 1, false)){}

        ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
        //ui32ADC0Value holds the ADC value...choose what to do with it...
        Semaphore_pend (ADCSem, BIOS_WAIT_FOREVER);
        //Semaphore_reset(ADCSem, 0);
        //Semaphore_pend (UARTSem, BIOS_WAIT_FOREVER);
        //Semaphore_post(UARTSem);
    }
}

void SRfun(void){

    while(1){
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0)==0x00)
        {
            buttonPressed = true;
            PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, (ui32ADC0Value[0]));
        }
        //Semaphore_reset(SRSem, 0);
        //Semaphore_pend (SRSem, BIOS_WAIT_FOREVER);
        Semaphore_pend (SRSem, BIOS_WAIT_FOREVER);

    }
}

void UARTfun(void){
```



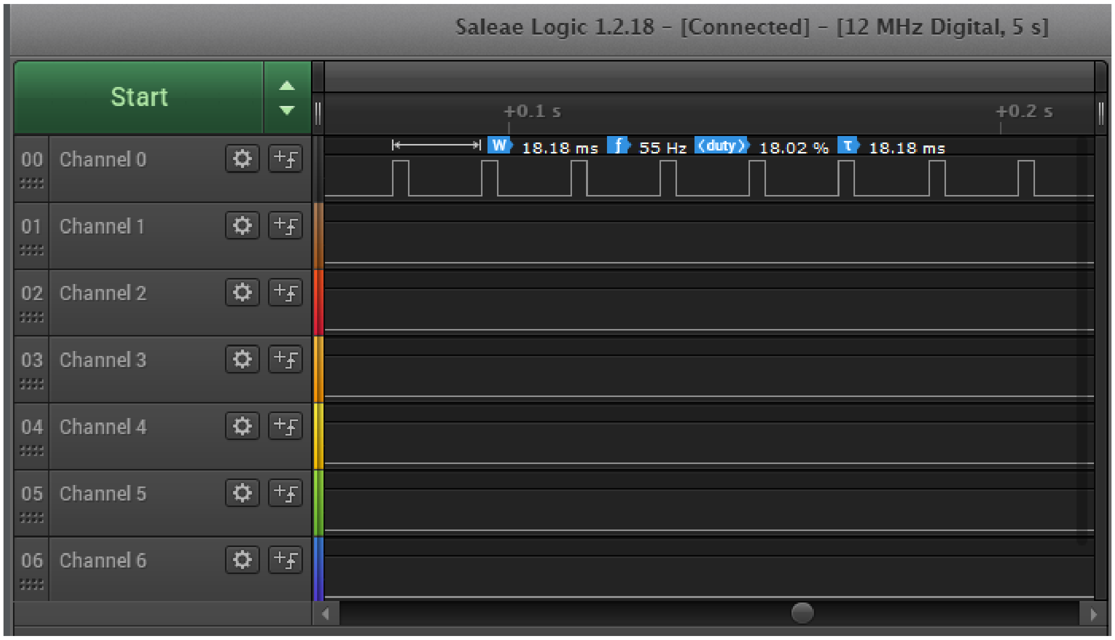
```c



    while(1){
        UARTprintf("ADC Value[0]: %d\n", ui32ADC0Value[0]);
        UARTprintf("ADC Value[1]: %d\n", ui32ADC0Value[1]);
        UARTprintf("ADC Value[2]: %d\n", ui32ADC0Value[2]);
        UARTprintf("ADC Value[3]: %d\n", ui32ADC0Value[3]);
        Semaphore_pend (UARTSem, BIOS_WAIT_FOREVER);
        //Semaphore_reset(UARTSem, 0);
        //Semaphore_pend (SRSem, BIOS_WAIT_FOREVER);
    }
}

void TIMER2INT(void){
    TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);         // must clear timer flag FROM timer
    i16ToggleCount = i16ToggleCount + 1; //increment every time HWI occurs
    //    System_printf("Timer 2 interrupt occurred\n");
    //    System_flush();

    if (i16ToggleCount == 10){
        //count = Semaphore_getCount(ADCSem);
        Semaphore_post (ADCSem);
    }

    else if (i16ToggleCount == 20){
        Semaphore_post (UARTSem);
    }

    else if (i16ToggleCount == 30){
        Semaphore_post (SRSem);
        i16ToggleCount = 0;
    }

    //Semaphore_post(ADCSem);
}

void TIMER_ISR(void){
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);         // must clear timer flag FROM timer
    if (!buttonPressed){
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
```



```c
        }
        else
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
        }
    }
    else {

        if(GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_0))
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
        }
        else
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
        }
    }
}

void reverse(char str[], int len){
    int start, end;
    char temp;
    for (start=0, end=len-1; start < end; start++, end--){
        temp = *(str+start);
        *(str+start) = *(str+end);
        *(str+end) = temp;
    }
}

char* itoa( int num, char* str, int base){
    int i = 0;
    bool isNegative = false;

    if (num==0){
        str[i] = '0';
        str[i+1] = '\0';
        return str;
    }

    if (num < 0 && base == 10) {
        isNegative = true;
        num = -num;
    }

    while (num!=0){
        int rem = num % base;
        str[i++] = (rem > 9) ? (rem - 10) + 'A' : rem + '0';
        num = num/base;
    }

    if (isNegative){
        str[i++] = '-';
    }

    str[i] = '\0';
    reverse(str,i);
    return str;
}

void InitConsole(void){
    //Enable GPIO port A which is used for UART0 pins
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    //Configure the pin muxing for UART 0 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.
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    //Initialize the UART for console I/O.
    UARTStdioConfig(0,115200,16000000);
}
```

PWM Pin

PD0=>



Saleae Logic 1.2.18 – [Connected] – [12 MHz Digital, 5 s]

W 18.18 ms f 55 Hz duty 18.02 % T 18.18 ms

Name: Serak Gebremedhin