

690 Project 2: SpeakerBuzz

Luke Weaver

November 1, 2017

Abstract

The purpose of this project is to create a Speaker lab for incoming embedded systems students. Students will play a tone from the speaker and control the pitch of the tone with buttons on the LaunchPad. The next task is to display the current frequency on UART. Due to this only being the students second lab and this covers concepts they are not yet familiar with(such as PWM, UART, buttons, and BoosterPacks), this will be a two week lab and sample code will be provided in the write up which can be found in the Abstract. The BOOSTXL-AUDIO BoosterPack is housed on the Tiva TM4C1294 Evaluation Board, which is used as the interface between the BoosterPack and the computer. I am using Code Composer V6.1.3 to communicate and build the project on the Tiva board. The project is set up to run a single PWM task in order to send data to the audio amp.

Contents

List of Figures	1
1 APPENDIX	2
2 Principles of Operation	6
3 Data Structure Descriptions	7
4 Function Descriptions	8
4.1 Task_PWM(void)	8
4.2 void TimerConfigure(uint32_t ui32Base, uint32_t ui32Config)	8
4.3 void TimerMatchSet(uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)	8
4.4 void TimerLoadSet(uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)	8
4.5 void TimerEnable(uint32_t ui32Base, uint32_t ui32Timer)	8
4.6 Main_Sharp1_Test.c(void)	8
4.7 PsuedoCode	8
5 Parameters	10
5.1 uint32_t Pulse Width	10
5.2 uint32_t frequency	10
5.3 uint32_t USR_SW1Data,USR_SW2Data	10
5.4 uint32_t SW1_Pressed, SW2_Pressed	10
5.5 unsigned long period	10
6 Testing	11
7 Lessons Learned and Revision History	12
8 Program Listing	13
8.1 Main_NoSharp1_Test.c	13
8.2 Task_MPU.c	15

List of Figures

1	Top-Level Block Diagram	6
2	UART Example Output	11

List of Tables

none

1 APPENDIX

EECS 388 Laboratory Exercise #2 SpeakerBuzz

1 Introduction

This laboratory implements a new task to generate a tone using the speaker on the BOOSTXL-AUDIO BoosterPack and to change the pitch with by using buttons on the TIVA LaunchPad. The first week's task is to generate a tone and the second week will be to get user input to change the pitch.

To begin this lab, make a copy of the Program_Blinky project in your workspace.

1. Select the Program_Blinky project and right-click.
2. Select "Copy"
3. In the workspace pane, right-click and select "Paste"
4. When asked for a project name, use the project name "Program_SpeakerBuzz"
5. Within the new project, create a task called "Task_PWM.c"

1.2 Tone Generation Task

Create and call an instance of this task in the main function. Go back to your Task_PWM.c file and enabling the following pins and ports.

1.2.1 Set-up AMP

You must initialize GPION and write a high value to Pin2 to enable the AMP to drive the speaker.

```
//Enable GPIO Port N.  
// SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);  
// Configure GPIO_N to drive the Speaker.  
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_2);  
//set PortN Pin2 as an output  
GPIOPinTypeOutput();  
//write a high value here
```

1.2.2 Creating an infinite while loop and defining frequency

1. Define a uint_t named halfFrequency.
2. Create a while loop with a delay using vTaskDelay(). VTaskDelay to use parameters configTICK_RATE_HZ and frequency. The VTaskDelay parameter is evaluated as an integer and computed as tics. The amount of tics per second is determined by the configTick_Rate_HZ.
3. Use VTaskDelay, set the period to 0.004 seconds.

1.2.3 Set-up Timers

```
//Enable GPIO Port N.
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
// Configure GPIO_N to drive the Speaker.
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_2);
//Turn on the amp. At this point you should hear an audible click coming from the
speaker on startup.
GPIOPinWrite( GPIO_PORTN_BASE, GPIO_PIN_2,1);
//Sets up pin to use alternative function. The General Purpose timer. Timer B Base 3 has
3 different modes: Capture, Compare, and PWM
GPIOPinConfigure(GPIO_PM3_T3CCP1);
GPIOPinTypeTimer(GPIO_PORTM_BASE, GPIO_PIN_3); //Sets as a timer pin.
//Specifies mode as PWM.
TimerConfigure(TIMER3_BASE,
TIMER_CFG_SPLIT_PAIR|TIMER_CFG_B_PWM);
//Initialized Pulse Width value
TimerMatchSet(TIMER3_BASE, TIMER_B, Pulse_Width);
//Set period
TimerLoadSet(TIMER3_BASE, TIMER_B, period);
//enables timer
TimerEnable(TIMER3_BASE, TIMER_B);
```

1.2.4 Generating a tone

1. Inside your while loop alternate the pulse width from a high to low value every half period.

1.2.3 Laboratory Measurements

Your GTA will ask you to display your frequency using “Task_PWM” using UART.

1.3 Buttons

Configure buttons (labeled as USR_SW1 and USR_SW2) on the Tiva board. Use these buttons to change the PWM frequency from 1.2. USR_SW1 will increase the frequency and USR_SW2 will decrease the frequency. Your GTA will assign you values to modify the frequency.

1.3.1 Set-up buttons

To set-up SW_0 and SW_1 you must enable GPIOJ Pin0 and Pin1

//Enable GPIO Port N.

// SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);

// Configure GPIO_N to drive the Speaker.

GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_2);

1.3.2 Change the period only during button down of SW_0 and SW_1

Use flags to determine if the button has been pressed or if it is released.

1.3.3 Laboratory Measurements

Your GTA will ask you to demonstrate the functionality of the buttons of your

“Task_PWM” using UART.

2 Principles of Operation

The BOOSTXL-AUDIO Boosterpack is connected to BoosterPack1 as labeled on the Tiva LaunchPad; this is crucial in order to recreate the lab using the later specified pins. In order to emit a frequency from the Speaker, it is required to enable the speaker's amp. This is done by enabling Port N Pin 2 on the Tiva LaunchPad and writing a high value to the GPIO. The next step enable was to enable the GPIO for the switches and the PWM for the amp. Port M pin 3 is corresponds to the PWM pin on the BoosterPack. I used the digital function of this pin (GPIO_PM3_T3CCP1) to control the PWM using Timers. This function configures Timer3 Base B to be used for this pin. This function has 3 modes: Capture, Compare, and PWM. I configure the Timer to use PWM mode then set the load and match set; for my purpose the load and match set are similar in function to the period and pulse width respectively. Alternating the match set value from high value (99.5%) of the period to a low value (0.5%) of the period at a rate of half the frequency I was able to generate a tone. A top-level block diagram of the design is shown below:

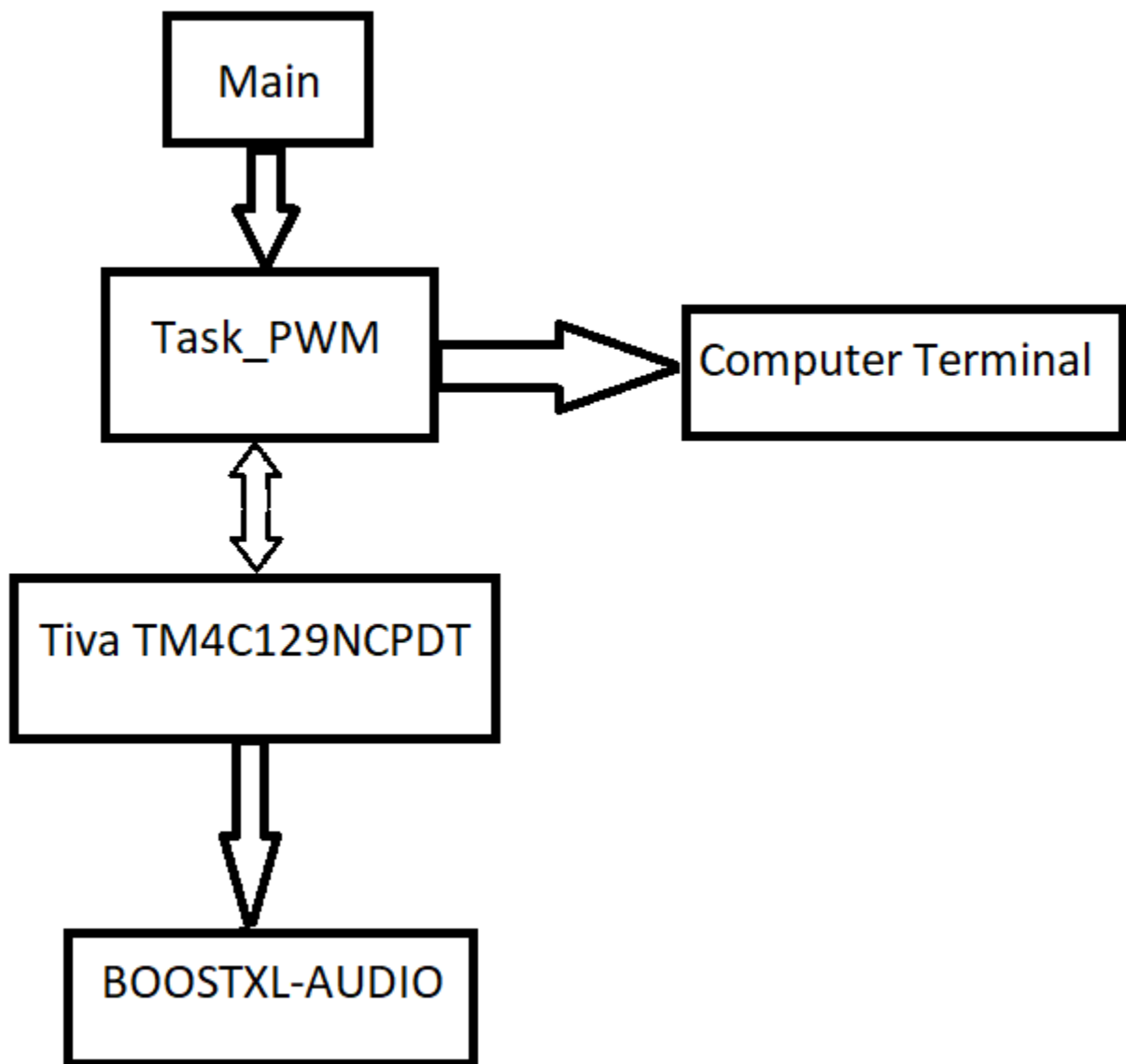


Figure 1: Top-Level Block Diagram

`Task_PWM`, is instantiated within the main using a function `xTaskCreate()`. This is part of the FreeRTOS language and is used to add a task to a list of tasks running; this is used for memory allocation. Upon creation of the PWM task, the PWM is initialized, and the task enters an infinite loop with a starting period of $1/(2000)$ seconds. Inside this loop is where the PWM alternates the match set creating a noise and where the frequency is changed depending on the value of the switches. Pressing Switch1 will increment the value and Switch2 will decrement the value of the frequency; the frequency is displayed on UART once it is changed.

3 Data Structure Descriptions

No major data structures were used.

4 Function Descriptions

4.1 Task_PWM(void)

The PWM task controls all order of operation. Task_PWM initializes the amp, speaker, frequency, timer, and

4.2 void TimerConfigure(uint32_t ui32Base, uint32_t ui32Config)

specifies the mode of the Timer, either as Capture, Compare, or as PWM; in our case we set it up as PWM

4.3 void TimerMatchSet(uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)

This function configures the match value for a timer and is used in PWM mode to determine the duty cycle

4.4 void TimerLoadSet(uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)

This function configures the timer load value.

4.5 void TimerEnable(uint32_t ui32Base, uint32_t ui32Timer)

This function enables operation of the timer module. The timer must be configured before it is enabled

4.6 Main_Sharp1_Test.c(void)

Instantiates Task_PWM and starts the FreeRTOS Task Scheduler.

4.7 PsuedoCode

Lab1 Psuedo Code

```
//function call in main
//xtaskcreate(Task_PWM())

//function inside Task_PWM.c
extern void Task_PWM( void )
{
    PWM_Initialization();
    enable_peripherals(Port M,N, and J)

    public float frequency;
    while ( 1 )
    {
        if(pwmValue = low)
            pwmValue = high;
        else
            pwmValue = low;
        if(OnButton1Down)
        {
            frequency--;
        }
        elseif(OnButton2Down)
        {
            frequency++;
        }
        vTaskDelay( frequency);
    }
}
```

} }

5 Parameters

5.1 uint32_t Pulse Width

This 4 byte integer is used as a parameter for `TimerMatchSet()` and alternates between a large and small

5.2 uint32_t frequency

This integer is used to determine how often the task runs. Although the name of this integer is frequen

5.3 uint32_t USR_SW1Data,USR_SW2Data

These integers read the values of the switches on the LaunchPad.

5.4 uint32_t SW1_Pressed, SW2_Pressed

These integers are used as flag conditions to insure that the frequency is only changed the moment a bu

5.5 unsigned long period

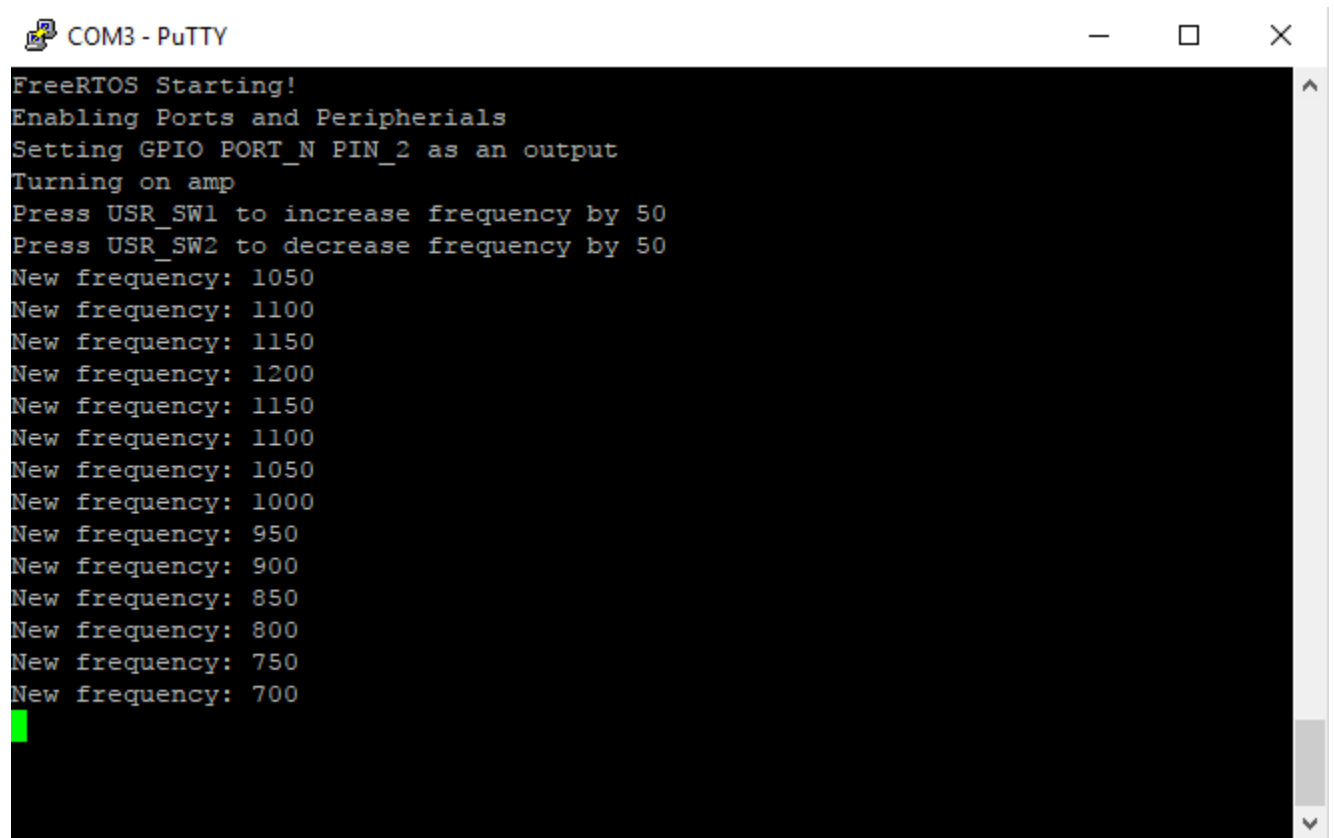
This is the period assigned in the initialization of the Timer inside the function `TimerLoadSet()`.

6 Testing

It was helpful to start with a base project that would build a project for the Tiva board. It included a task named `Task_Blink_LED_PortN_1` which would simply make an LED on the board blink at a specified rate. By leaving this task in our project, I could see if the board was successfully running the project build. I ran into a few problems when importing my `Task_PWM` into the project, but I used the `Task_Blink_LED_PortN_1` as a basis of replication to make the task compatible with the project.

The majority of fixes had to do with configuring the right port and pin. I initially tried port M pin 3 as it is the port that corresponds to the primary PWM on the BoosterPack. Unfortunately I could not get it to emit a noise so I resorted to trying the alternative PWM on the BoosterPack, which was enabled by using Port F pin 1. My thought was to use the enable the correct generator and use Port F pin 1's digital function M0PWM1; unfortunately due to hardware constraints this did not work. With Doctor Gary Minden's guidance, I was able to configure the primary PWM on the BoosterPack by using timers. Although I was able to generate a tone it came out robotic and I noticed that functions like writing to UART every half frequency would cause the program to noticeably stall. I realized that this is because I was not allocating enough memory; so I allocated memory parameter in `xTaskCreate` from 128 to 512.

I output the current frequency to UART to see if the buttons were correctly changing the frequency by the correct amount. This allowed me test my buttons logic and to see that the frequency was increasing when the user pressed SW1 and decreased when the user pressed SW2.



```
COM3 - PuTTY
FreeRTOS Starting!
Enabling Ports and Peripherals
Setting GPIO PORT_N PIN_2 as an output
Turning on amp
Press USR_SW1 to increase frequency by 50
Press USR_SW2 to decrease frequency by 50
New frequency: 1050
New frequency: 1100
New frequency: 1150
New frequency: 1200
New frequency: 1150
New frequency: 1100
New frequency: 1050
New frequency: 1000
New frequency: 950
New frequency: 900
New frequency: 850
New frequency: 800
New frequency: 750
New frequency: 700
```

Figure 2: UART Example Output

7 Lessons Learned and Revision History

At first I wanted to make this lab complicated by parsing microphone data to control the speaker. I learned that it is important to remember why you are doing something rather than if you can. Doctor Minden stressed that we are designing this lab for a student beginning Embedded Systems. I realized how overwhelming it can be to be introduced to many new concepts at once. To make things simpler on myself and any future student that might work on this, I abandoned controlling the microphone all together. The amount of functions and concepts introduced in a single lab would not bode well with the students. I spoke with current Embedded Systems students to come to the now implemented project: a speaker controlled by buttons. The concepts used in this project introduce PWM, GPIO, and UART; since these are new concepts I have left critical sample code in the lab write up. I have greater knowledge on BoosterPacks and feel confident going into the next project.

8 Program Listing

8.1 Main_NoSharp1_Test.c

```
/*--Main_Sharp1_Test.c
 *
 * Author: Luke Weaver
 * Organization: KU/EECS/EECS 690
 * Date: October 30, 2017 (B60313)
 *
 * Description: Starts RTOS Scheduler and Creates tasks: Task_PWM
 *
 *
 *
 */

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"

#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>

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

#include "Drivers/Processor_Initialization.h"
#include "Drivers/UARTStdio_Initialization.h"
#include "Drivers/uartstdio.h"

#include "FreeRTOS.h"
#include "task.h"

#include <stdio.h>

extern void Task_PWM( void *pvParameters );
extern void Task_Blink_LED_PortN_1( void *pvParameters );
extern void Task_ReportTime( void *pvParameters );
extern void Task_ReportData( void *pvParameters );

//extern void Task_I2C7_Handler( void *pvParameters );

int main( void ) {
    SysCtlClockFreqSet(SYSCTL_OSC_INT | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_320, 120000000); //set clock rate
    Processor_Initialization();
    UARTStdio_Initialization();
    //
    // Create a task to blink LED
```

```

//
//xTaskCreate( Task_Blink_LED_PortN_1, "Task_Blink_LED_PortN_1", 128, NULL, 1, NULL );
UARTprintf( "FreeRTOS Starting!\n" );
xTaskCreate( Task_PWM, "PWM", 512, NULL, 1, NULL );

//
//    Create a task to report data.
//
//xTaskCreate( Task_ReportData, "ReportData", 512, NULL, 1, NULL );

//
//    Create a task to report SysTickCount
//
//xTaskCreate( Task_ReportTime, "ReportTime", 512, NULL, 1, NULL );

//
//    Create a task to initialize I2C7_Handler.
//
//xTaskCreate( Task_I2C7_Handler, "I2C7_Handler", 1024, NULL, 2, NULL );

//
//    Start FreeRTOS Task Scheduler
//
vTaskStartScheduler();

while ( 1 ) {

}

}

```

8.2 Task_MPU.c

```
/*--Task_PWM.c
 *
 * Author:          Luke Weaver
 * Organization:    KU/EECS/EECS 388
 * Date:           February 22, 2016
 *
 * Description:     Configures PWM using Timers on Tiva TMC41294 Evaluation board
 *
 */

#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>

#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_memmap.h"
#include "inc/hw_uart.h"
#include "inc/hw_timer.h"

#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/pwm.h"
#include "driverlib/timer.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "FreeRTOS.h"
#include "task.h"

uint32_t Pulse_Width;
uint32_t frequency; //in this case 'frequency' is only half of the actual frequency
unsigned long period;
uint32_t USR_SW1Data;
uint32_t USR_SW2Data;
uint32_t SW1_Pressed;
uint32_t SW2_Pressed;
extern void Task_PWM( void *pvParameters ) {

    Pulse_Width = 10;
    period = 2000;
    frequency = 500;
    SW1_Pressed = 0;
    SW2_Pressed = 0;
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER3);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION); // Port N
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM); //port M
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ); //port J
    UARTprintf("Enabling Ports and Peripherals\n");
    GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0); //PORT_J PIN_0
```

```

GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_1); //PORT_J PIN_1
GPIOPadConfigSet( GPIO_PORTJ_BASE,(GPIO_PIN_0 |GPIO_PIN_1) , GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_W
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_2); //PORT_N PIN_2
UARTprintf("Setting GPIO PORT_N PIN_2 as an output\n");
GPIOPinWrite( GPIO_PORTN_BASE, GPIO_PIN_2,1); //amp on
UARTprintf("Turning on amp \n");

GPIOPinConfigure(GPIO_PM3_T3CCP1); //Sets up pin to use alternative function. The General Purpose t
GPIOPinTypeTimer(GPIO_PORTM_BASE, GPIO_PIN_3); //Sets as a timer pin.

TimerConfigure(TIMER3_BASE, TIMER_CFG_SPLIT_PAIR|TIMER_CFG_B_PWM); //Specifies mode as PWM.
TimerMatchSet(TIMER3_BASE, TIMER_B, Pulse_Width); //Initialized Pulse Width valu
TimerLoadSet(TIMER3_BASE, TIMER_B, period); //Set period
TimerEnable(TIMER3_BASE, TIMER_B); //enables timer

UARTprintf("Press USR_SW1 to increase frequency by 50 \n");
UARTprintf("Press USR_SW2 to decrease frequency by 50 \n");

while (1) {
    //alternate between high and low pulse widths
    if(Pulse_Width == 10){
        Pulse_Width = 1990;
    }
    else{
        Pulse_Width = 10;
    }
    USR_SW1Data = GPIOPinRead( GPIO_PORTJ_BASE, (GPIO_PIN_0));
    USR_SW2Data = GPIOPinRead( GPIO_PORTJ_BASE, (GPIO_PIN_1));
    //SW1 Logic
    if(SW1_Pressed == 0 && USR_SW1Data == 0){
        frequency+=25;
        UARTprintf("New frequency: %d \n",(2*frequency));
        SW1_Pressed = 1;
    }
    else if(SW1_Pressed == 1 && USR_SW1Data != 0){
        SW1_Pressed = 0;
    }
    //SW2 Logic
    if(SW2_Pressed == 0 && USR_SW2Data == 0){
        if(frequency<50)
        {
            UARTprintf("Cannot set frequency to 0 \n");
        }
        else
        {
            frequency-=25;
            UARTprintf("New frequency: %d \n", (2*frequency));
        }
        SW2_Pressed = 1;
    }
    else if(SW2_Pressed == 1 && USR_SW2Data != 0){
        SW2_Pressed = 0;
    }
}

```

```
        //set pulse width
        TimerMatchSet(TIMER3_BASE, TIMER_B, Pulse_Width);
        vTaskDelay( ( 1 * configTICK_RATE_HZ ) / frequency);    // 10000/500 = 20 tics. configTICK_RA
    }
}
```