01/30/24

Lab 2 Sine Wave – ECE 5780

Nate Sheffield
A02268057
Nathan Critchfield
A02283426

Objective

The purpose of this lab is to implement a program using FreeRTOS to produce a 440 Hz Sine Wave when an LED is activated by a button push on our STM32 Nucleo Board. This sine wave is then put through an audio amplifier circuit and an 8-ohm speaker to produce the sound of the sine wave.

Procedure

Most of this lab was just modifying Lab 1 to also produce a 440 Hz sine wave out of a speaker connected to the MCU through an amplifier as well as light up the LED when the user push button is pressed. We started by loading our code from Lab 1, the blinking LED, into Lab 2. Then we started setting up the Timer 4 interrupt on the MCU. This involved setting up the configuration registers for the timer. After we were certain that the timer interrupt was functioning by setting a breakpoint in the interrupt handler, we started to set up the DAC to produce a 440 Hz sine wave. We used a sine wave lookup table and the timer interrupt to produce the wave. Once the wave was being produced, we built the speaker/amplifier circuit. With the DAC output connected to the circuit we were able to produce an A note through the speaker. The 440 Hz oscilloscope reading is pictured below in Figure 2.

Results

For our lab to produce the desired results we had to modify our existing Lab 1 code, assemble an audio amplifier circuit and connect it all together to an 8 ohm speaker. Starting from our Lab 1 code that toggles LED an LED with a button press we had to enable an interrupt based on Timer 4 and then to send an output of a 440 Hz Sine Wave from a lookup table through the DAC at each interrupt to the audio amplifier circuit. While implementing the modifications to the code we had some issues getting all of the correct initialization set up correctly for Timer 4 and for the DAC. Initially, when we ran the code we were not getting output to DAC so the audio amplifier circuit did nothing. We found that Timer 4 was not triggering the interrupt because we did not have all of setting that we needed initialized. After updating our code we were able to trigger interrupts, however, our amplifier circuit still did not do anything.

After this we figured that we had a wiring issue within our circuit. We returned to look at the example circuit in the LM386 datasheet and compared it to our circuit and we found that two of our wires were not connected on the right line so it was not getting through the circuit to the speaker. After moving these wires we were able to produce the 440 Hz Sine Wave sound. After this we did some minor adjustments because our sine wave was clipping a little bit on the

bottom. After adjusting our lookup table we were able to get a consistent sine wave without clipping.

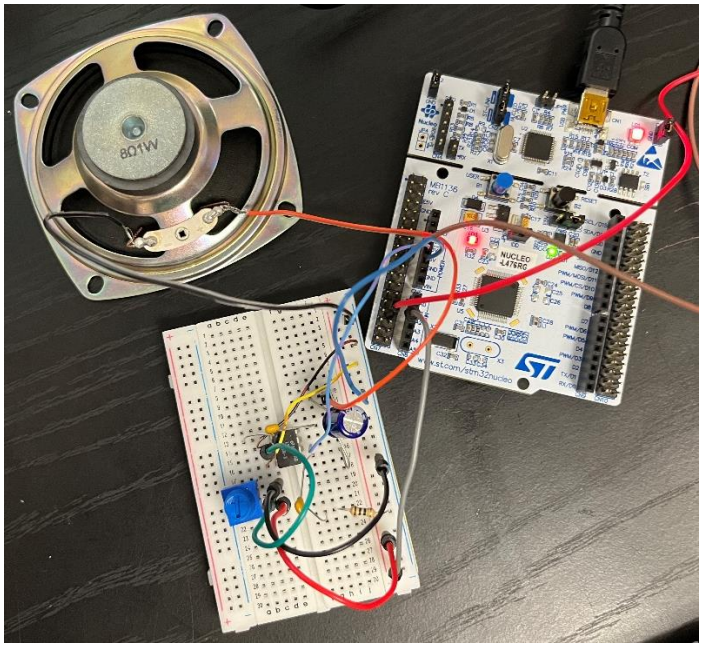

Figure 1. Audio Amplifier Circuit connected to our STM32 Nucleo Board

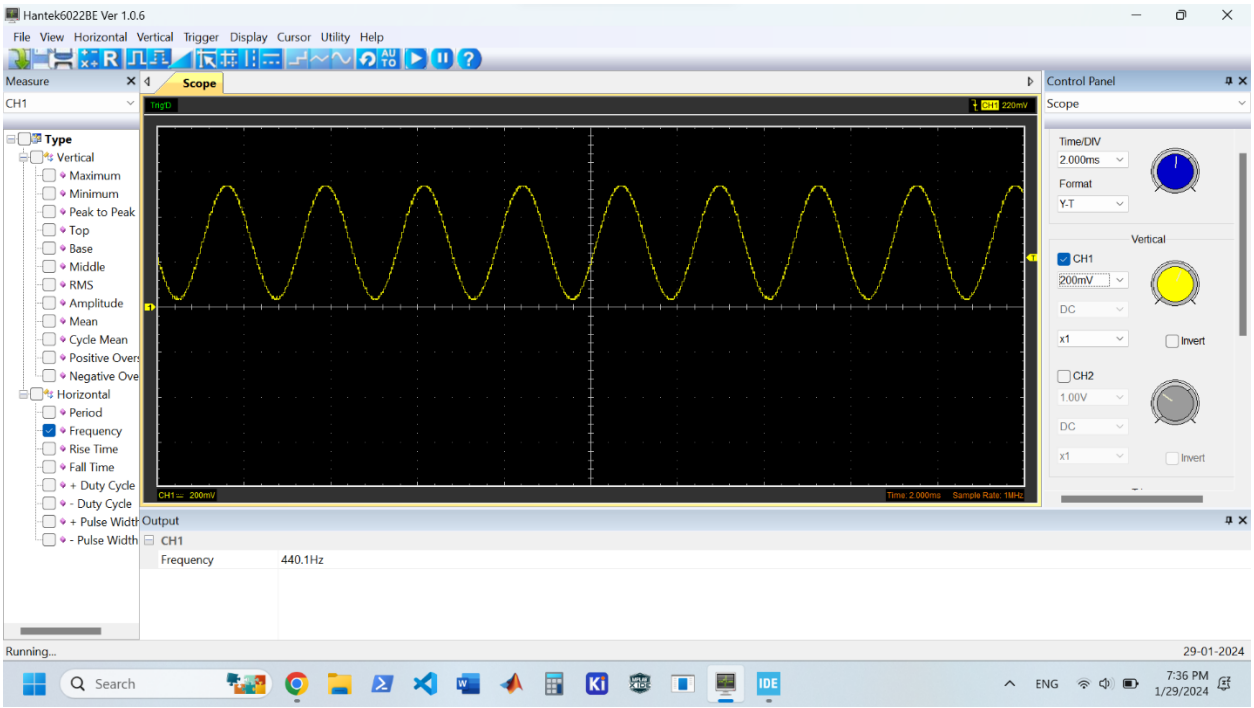

Figure 2. Oscilloscope Screenshot showing our 440 Hz Sine Wave

Conclusion

In conclusion, we were successfully able to modify our lab 1 code and implement an audio amplifier circuit to produce a 440 Hz Sine Wave. We were able to adjust our sine wave so that it did not clip on the rails of the audio amplifier. We also successfully wired the audio amplifier circuit so that we could adjust the intensity of the sound with our potentiometer. Likewise, we implemented Timer 4 so that it would cause an interrupt that would pass in a value from a sine wave lookup table to the DAC which then sent that value through the amplifier circuit to the speaker to produce the sine wave. In the end, we were able to complete all of the lab requirements successfully.

Appendix

Main.c code

```c
#include "FreeRTOS.h"
#include "stm32l476xx.h"
#include "system_stm32l4xx.h"
#include "task.h"
#include "timers.h"
#include "stdint.h"

#include "init.h"

static uint32_t led_state;

int main(void) {
	//Initialize System
	SystemInit();

	clock_Config();
	gpio_Config();
	timer_Config();
	DAC_Config();

	led_state = 0;

	xTaskCreate( //Task for LED
		LED_task, "LED", 16, NULL, 1, NULL);

	xTaskCreate(	//Task for Button
		Button_task, "Button", 16, NULL, 1, NULL);

	//Start Task Scheduler
	vTaskStartScheduler();
	while(1);
}

//Function to toggle led_state
void LED_task(void *pvParameters){
	while(1){
		//If the LED is on turn it off
		if(led_state == 1){
			GPIOA->BSRR |= GPIO_BSRR_BS5;
		}
		//If the LED is off turn it on
		else {
			GPIOA->BSRR |= GPIO_BSRR_BR5;
		}
	}
```

```
}

//Function to read in button state and led_state
void Button_task(void *pvParameters){
	while(1){
		uint32_t button_in;
		//Read in the value of the button
		button_in = GPIOC->IDR;
		button_in &= GPIO_IDR_ID13_Msk;

		//If the button is pressed toggle the LED
		if(button_in == 0){
			while(button_in == 0){
				button_in = GPIOC->IDR;
				button_in &= GPIO_IDR_ID13_Msk;
			}
			if(led_state == 0){
				led_state = 1;
			}
			else {
				led_state = 0;
			}
		}
	}
}

void TIM4_IRQHandler(void){
	static uint32_t sine_count = 0;

	const uint16_t sineLookupTable[] = {
	305, 335, 365, 394, 422, 449, 474, 498, 521, 541, 559, 574, 587, 597, 604,
	609, 610, 609, 604, 597, 587, 574, 559, 541, 521, 498, 474, 449, 422, 394,
	365, 335, 305, 275, 245, 216, 188, 161, 136, 112, 89, 69, 51, 36, 23,
	13, 6, 1, 0, 1, 6, 13, 23, 36, 51, 69, 89, 112, 136, 161,
	188, 216, 245, 275};

	//If the LED is on
	if (led_state == 1){
		sine_count++; //Increment to the next value in the table
		if (sine_count == 64){
			sine_count = 0;
		}
	}
	//Assign DAC to Sine_Wave Table Current Value
	DAC->DHR12R1 = sineLookupTable[sine_count] + 45;

	TIM4->SR &= ~TIM_SR_UIF; //Clears Interrupt Flag
}
```

Init.c code

```
#include "FreeRTOS.h"
#include "stm32l476xx.h"
#include "system_stm32l4xx.h"
#include "task.h"
#include "timers.h"
#include "stdint.h"

#include "init.h"

void clock_Config(void){
	//Change System Clock from MSI to HSI
	RCC->CR |= RCC_CR_HSION;         // enable HSI (internal 16 MHz clock)
	while ((RCC->CR & RCC_CR_HSIRDY) == 0);
```

```
14.       RCC->CFGR |= RCC_CFGR_SW_HSI;    // make HSI the system clock
15.       SystemCoreClockUpdate();
16.
17.       //Turn Clock on for GPIOs
18.       RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
19.       //RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
20.       RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
21. }
22.
23. void gpio_Config(void){
24.       //Set PA5 to output mode for LED
25.       GPIOA->MODER &= ~GPIO_MODER_MODE5_1;
26.       GPIOA->MODER |= GPIO_MODER_MODE5_0;
27.       //Turn LED on
28.       GPIOA->BSRR |= GPIO_BSRR_BS5;
29.       //Set PC13 to input mode for Button
30.       GPIOC->MODER &= ~GPIO_MODER_MODE13; //0xf3ffffff
31. }
32.
33. void timer_Config(void){
34.       //Turn on Clock for TIM4
35.       RCC -> APB1ENR1 |= RCC_APB1ENR1_TIM4EN;
36.
37.       //Enable interrupts for TIM4
38.       NVIC->ISER[0] |= 1 << 30;
39.       NVIC_EnableIRQ(TIM4_IRQn);
40.
41.       //Enable DMA/Interrupt Handler
42.       /*
43.       TIM4->DIER |= TIM_DIER_TIE;
44.       TIM4->DIER |= TIM_DIER_UIE;
45.
46.       //Configure Auto-Reload Register
47.       TIM4->ARR = 0xFFFF023A;
48.       */
49.
50.       TIM4->CR1 &= ~TIM_CR1_CMS;    // Edge-aligned mode
51.       TIM4->CR1 &= ~TIM_CR1_DIR;    // Up-counting
52.
53.       TIM4->CR2 &= ~TIM_CR2_MMS;    // Select master mode
54.       TIM4->CR2 |= TIM_CR2_MMS_2;    // 100 = OC1REF as TRGO
55.
56.       TIM4->DIER |= TIM_DIER_TIE;    // Trigger interrupt enable
57.       TIM4->DIER |= TIM_DIER_UIE;    // Update interrupt enable
58.
59.       TIM4->CCMR1 &= ~TIM_CCMR1_OC1M;
60.       TIM4->CCMR1 |= (TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2);  // 0110 = PWM mode 1
61.
62.       TIM4->PSC = 0x7;        // 16 MHz / (7+1) = 2 MHz timer ticks
63.       TIM4->ARR = 0xFFFF0046; // 2 MHz / (70+1) = 28.169 kHz interrupt rate; 64 entry look-up
table = 440.14 Hz sine wave
64.       TIM4->CCR1 = 0x23;           // 50% duty cycle (35)
65.       TIM4->CCER |= TIM_CCER_CC1E;
66.
67.       //Enable Control Register 1 for Counting
68.       TIM4->CR1 |= TIM_CR1_CEN;
69. }
70.
71. void DAC_Config(void){
72.       //Turn on Clock for DAC1
73.       RCC -> APB1ENR1 |= RCC_APB1ENR1_DAC1EN;
74.
75.       //Configure DAC1 GPIO in Analog Mode 0x3
76.       GPIOA->MODER |= GPIO_MODER_MODE4;
77.
```

```
78.       //Enable DAC1 Channel 1
79.       DAC->CR |= DAC_CR_EN1;
80. }
```

Init.h code

```
 1. #ifndef INIT_H
 2. #define INIT_H
 3.
 4. #include "FreeRTOS.h"
 5. #include "stm32l476xx.h"
 6. #include "system_stm32l4xx.h"
 7. #include "task.h"
 8. #include "timers.h"
 9. #include "stdint.h"
10.
11. void LED_task(void *pvParameters);
12. void Button_task(void *pvParameters);
13.
14. void clock_Config(void);
15. void gpio_Config(void);
16. void timer_Config(void);
17. void DAC_Config(void);
18.
19. void TIM4_IRQHandler(void);
20.
21. #endif
```