**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**

**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.

A circuit board with wires connected to it

Description automatically generated

**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***

1. #include "FreeRTOS.h"

2. #include "stm32l476xx.h"

3. #include "system\_stm32l4xx.h"

4. #include "task.h"

5. #include "timers.h"

6. #include "stdint.h"

7.

8. #include "init.h"

9.

10. static uint32\_t led\_state;

11.

12. int main(void) {

13. //Initialize System

14. SystemInit();

15.

16. clock\_Config();

17. gpio\_Config();

18. timer\_Config();

19. DAC\_Config();

20.

21. led\_state = 0;

22.

23. xTaskCreate( //Task for LED

24. LED\_task, "LED", 16, NULL, 1, NULL);

25.

26. xTaskCreate( //Task for Button

27. Button\_task, "Button", 16, NULL, 1, NULL);

28.

29. //Start Task Scheduler

30. vTaskStartScheduler();

31. while(1);

32. }

33.

34. //Function to toggle led\_state

35. void LED\_task(void \*pvParameters){

36. while(1){

37. //If the LED is on turn it off

38. if(led\_state == 1){

39. GPIOA->BSRR |= GPIO\_BSRR\_BS5;

40. }

41. //If the LED is off turn it on

42. else {

43. GPIOA->BSRR |= GPIO\_BSRR\_BR5;

44. }

45. }

46. }

47.

48. //Function to read in button state and led\_state

49. void Button\_task(void \*pvParameters){

50. while(1){

51. uint32\_t button\_in;

52. //Read in the value of the button

53. button\_in = GPIOC->IDR;

54. button\_in &= GPIO\_IDR\_ID13\_Msk;

55.

56. //If the button is pressed toggle the LED

57. if(button\_in == 0){

58. while(button\_in == 0){

59. button\_in = GPIOC->IDR;

60. button\_in &= GPIO\_IDR\_ID13\_Msk;

61. }

62. if(led\_state == 0){

63. led\_state = 1;

64. }

65. else {

66. led\_state = 0;

67. }

68. }

69. }

70. }

71.

72. void TIM4\_IRQHandler(void){

73. static uint32\_t sine\_count = 0;

74.

75. const uint16\_t sineLookupTable[] = {

76. 305, 335, 365, 394, 422, 449, 474, 498, 521, 541, 559, 574, 587, 597, 604,

77. 609, 610, 609, 604, 597, 587, 574, 559, 541, 521, 498, 474, 449, 422, 394,

78. 365, 335, 305, 275, 245, 216, 188, 161, 136, 112, 89, 69, 51, 36, 23,

79. 13, 6, 1, 0, 1, 6, 13, 23, 36, 51, 69, 89, 112, 136, 161,

80. 188, 216, 245, 275};

81.

82. //If the LED is on

83. if (led\_state == 1){

84. sine\_count++; //Increment to the next value in the table

85. if (sine\_count == 64){

86. sine\_count = 0;

87. }

88. }

89. //Assign DAC to Sine\_Wave Table Current Value

90. DAC->DHR12R1 = sineLookupTable[sine\_count] + 45;

91.

92. TIM4->SR &= ~TIM\_SR\_UIF; //Clears Interrupt Flag

93. }

***Init.c code***

1. #include "FreeRTOS.h"

2. #include "stm32l476xx.h"

3. #include "system\_stm32l4xx.h"

4. #include "task.h"

5. #include "timers.h"

6. #include "stdint.h"

7.

8. #include "init.h"

9.

10. void clock\_Config(void){

11. //Change System Clock from MSI to HSI

12. RCC->CR |= RCC\_CR\_HSION; // enable HSI (internal 16 MHz clock)

13. 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