**Lab 3 Audio UART – ECE 5780**

Nate Sheffield

A02268057

Nathan Critchfield

A02283426

**Objective**

The purpose of this lab is to implement a program using FreeRTOS to produce 8 different notes from 220 Hz to 440 Hz when an LED is activated by a button push on our STM32 Nucleo Board. These will be produced as sine waves that are 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 2 code to be able to produce 8 sine wave frequencies (notes) from 220 Hz to 440 Hz. These sine wave frequencies are outputed through our audio amplifier circuit when the LED is activated with a button press. In order to switch which note is produced our labs accepts UART serial input from a PuTTY terminal. If a character from ‘a’-‘h’ is sent from the terminal it will change what note is currently being played through the speaker. These notes that are played represent the notes in one octave of a musical scale: low a, b, c, d, e, f, g, and high a. As per the lab requirements our code does not contain global variables and instead uses two different queues to send data between tasks and interrupts. We have a queue to send data between two tasks and then a different queue to send data between a task and an interrupt. This allows our interrupts to be aware if the LED is active or not and whether or not the TIM4 should be outputing the sine wave frequency to the audio amplifier through the DAC.

While writing this lab we had a lot of issues getting our code to function properly. When we first implemented our UART we had several issues with getting all of the necessary intialization steps implemented with the right values and in the right order. We had to refer back to how we initialized our lab in the microcontrollers class and then modify it to implement interrupts and use a different clock. At first we were unable to receive any input from the UART and we could not figure out what was wrong with our initialization. After a lot of looking through the STM32 datasheet we were able to figure out that we had one bit wrong on our clock selection and that our gpio pin was not getting put in the correct mode which was also only off by one bit. After this we were finally able to accept serial input from a PuTTY terminal.

We additionally had issues with getting our queues implemented correctly to eliminate the global variables in our codes. We implemented the correct queue receive and send functions in the proper locations dependant on if it was in an interrupt or a task. However, we were trying to use our queue in the timer interrupt before we had created our queues. After discovering this our queues were able to work but we ran into another issue. After this we were hitting the config assert function which meant our priorities of our tasks and interrupts were off as well as were using too much memory in the creation of our tasks and interrupts. After adjusting our values to make them smaller we were finally able to get our LED, Button, Timer and UART working again.

A circuit board with wires connected to it

Description automatically generated

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

**A screenshot of a computer code

Description automatically generated**

**Figure 2. Code Snippet of the ARR values that produce the different note frequencies**

**Conclusion**

In conclusion, we learned a lot from the issues of this lab. Though it was frustrating to run into issue after issue it helped us understand more about the structure of queues and how the UART works using the STM32 and FreeRTOS. We learned that the order of how things are initialized matters and that trying to use a queue before it is created causes your code to get stuck in an infinite loop. We also learned that one or two bits can completely change the initialization of a module and cause it to not work properly. Likewise, it is important to understand the scope of your variables and to make sure that things are defined in the right location and in the right way so their data does not get overwritten or deallocated. Overall, this was a challenging lab for us but in the end, we were able to accomplish all 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. #include "queue.h"

8.

9. #include "init.h"

10.

11. int main(void) {

12. //Initialize System

13. SystemInit();

14. clock\_Config();

15. but\_led\_queue = xQueueCreate(2,sizeof(uint8\_t));

16. but\_tim\_queue = xQueueCreate(2,sizeof(uint8\_t));

17. gpio\_Config();

18. UART\_config();

19. timer\_Config();

20. DAC\_Config();

21.

22. //Set the priority for the interrupts

23. NVIC\_SetPriority(USART2\_IRQn,0x07);

24. NVIC\_SetPriority(TIM4\_IRQn,0x07);

25.

26. //Task for LED

27. if(xTaskCreate(LED\_task, "LED", 32, NULL, 2, NULL) != pdPASS){

28. while(1);

29. }

30. //Task for Button

31. if(xTaskCreate(Button\_task, "Button", 32, NULL, 2, NULL) != pdPASS){

32. while(1);

33. }

34.

35. //Start Task Scheduler

36. vTaskStartScheduler();

37. while(1);

38. }

39.

40. //Function to toggle led\_state

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

42. static uint8\_t buffer[1];

43. buffer[0] = 0;

44. while(1){

45. if(uxQueueMessagesWaiting(but\_led\_queue) > 0){

46. if(xQueueReceive(but\_led\_queue,buffer,50)== pdTRUE){}

47. }

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

49. if(buffer[0] == 1){

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

51. }

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

53. else if(buffer[0] == 0){

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

55. }

56. }

57. }

58.

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

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

61. static uint8\_t buffer[1];

62. buffer[0] = 0;

63. uint32\_t button\_in;

64. while(1){

65. //Read in the value of the button

66. button\_in = GPIOC->IDR;

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

68.

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

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

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

72. button\_in = GPIOC->IDR;

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

74. }

75. if(buffer[0] == 0){

76. buffer[0] = 1;

77. //Send led\_state to queue for LED Task

78. xQueueSendToBack(but\_led\_queue,buffer,50);

79. //Send led\_state to queue for TIM4\_IRQHandler

80. xQueueSendToBack(but\_tim\_queue,buffer,50);

81. }

82. else {

83. buffer[0] = 0;

84. //Send led\_state to queue for LED Task

85. xQueueSendToBack(but\_led\_queue,buffer,50);

86. //Send led\_state to queue for TIM4\_IRQHandler

87. xQueueSendToBack(but\_tim\_queue,buffer,50);

88. }

89. }

90. }

91. }

92.

93. void TIM4\_IRQHandler(void){

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

95. static uint8\_t buffer[1];

96.

97. const uint16\_t sineLookupTable[] = {

98. 305, 335, 365, 394, 422, 449, 474, 498, 521, 541, 559, 574, 587, 597, 604,

99. 609, 610, 609, 604, 597, 587, 574, 559, 541, 521, 498, 474, 449, 422, 394,

100. 365, 335, 305, 275, 245, 216, 188, 161, 136, 112, 89, 69, 51, 36, 23,

101. 13, 6, 1, 0, 1, 6, 13, 23, 36, 51, 69, 89, 112, 136, 161,

102. 188, 216, 245, 275};

103.

104. //if there is a message waiting in the queue from ISR

105. if(uxQueueMessagesWaitingFromISR(but\_tim\_queue) > 0){

106. xQueueReceiveFromISR(but\_tim\_queue,buffer,NULL);

107. }

108. //if the LED is on

109. if (buffer[0] == 1){

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

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

112. sine\_count = 0;

113. }

114. }

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

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

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

118. }

119.

120. void USART2\_IRQHandler(void){

121. uint8\_t uart\_buffer = 0;

122. //xQueueSendToBackFromISR(uart\_tim\_queue,&uart\_buffer,NULL);

123. change\_note(uart\_buffer);

124. }

125.

126. void change\_note(uint8\_t uart\_buffer){

127. uart\_buffer = (uint8\_t)(USART2->RDR);

128. if(uart\_buffer == 'a'){

129. TIM4->ARR = 0xFFFF008E; //2 MHz/(142) = 14.080 kHz interrupt rate; 220 Hz sine wave

130. }

131. else if (uart\_buffer == 'b'){

132. TIM4->ARR = 0xFFFF007E; //126; 246.94 Hz

133. }

134. else if (uart\_buffer == 'c'){

135. TIM4->ARR = 0xFFFF0077; //119; 261.63 Hz

136. }

137. else if (uart\_buffer == 'd'){

138. TIM4->ARR = 0xFFFF006A; //106; 293.66 Hz

139. }

140. else if (uart\_buffer == 'e'){

141. TIM4->ARR = 0xFFFF005E; //94; 329.63 Hz

142. }

143. else if (uart\_buffer == 'f'){

144. TIM4->ARR = 0xFFFF0059; //89; 349.23 Hz

145. }

146. else if (uart\_buffer == 'g'){

147. TIM4->ARR = 0xFFFF004F; //79; 392.00 Hz

148. }

149. else if (uart\_buffer == 'h'){

150. TIM4->ARR = 0xFFFF0046; //71; 440 Hz (High A)

151. }

152. }

**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. #include "queue.h"

8.

9. #include "init.h"

10.

11. void clock\_Config(void){

12. //Change System Clock from MSI to HSI

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

14. while ((RCC->CR & RCC\_CR\_HSIRDY) == 0);

15. RCC->CFGR |= RCC\_CFGR\_SW\_HSI; // make HSI the system clock

16. SystemCoreClockUpdate();

17.

18. //Turn Clock on for GPIOs

19. RCC -> AHB2ENR |= RCC\_AHB2ENR\_GPIOAEN;

20. //RCC -> AHB2ENR |= RCC\_AHB2ENR\_GPIOBEN;

21. RCC -> AHB2ENR |= RCC\_AHB2ENR\_GPIOCEN;

22. }

23.

24. void gpio\_Config(void){

25. //Set PA5 to output mode for LED

26. GPIOA->MODER &= ~GPIO\_MODER\_MODE5\_1;

27. GPIOA->MODER |= GPIO\_MODER\_MODE5\_0;

28. //Turn LED on

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

30. //Set PC13 to input mode for Button

31. GPIOC->MODER &= ~GPIO\_MODER\_MODE13; //0xf3ffffff

32. }

33.

34. void timer\_Config(void){

35. //Turn on Clock for TIM4

36. RCC -> APB1ENR1 |= RCC\_APB1ENR1\_TIM4EN;

37.

38. //Enable interrupts for TIM4

39. NVIC->ISER[0] |= 1 << 30;

40. NVIC\_EnableIRQ(TIM4\_IRQn);

41.

42. TIM4->CR1 &= ~TIM\_CR1\_CMS; // Edge-aligned mode

43. TIM4->CR1 &= ~TIM\_CR1\_DIR; // Up-counting

44.

45. TIM4->CR2 &= ~TIM\_CR2\_MMS; // Select master mode

46. TIM4->CR2 |= TIM\_CR2\_MMS\_2; // 100 = OC1REF as TRGO

47.

48. TIM4->DIER |= TIM\_DIER\_TIE; // Trigger interrupt enable

49. TIM4->DIER |= TIM\_DIER\_UIE; // Update interrupt enable

50.

51. TIM4->CCMR1 &= ~TIM\_CCMR1\_OC1M;

52. TIM4->CCMR1 |= (TIM\_CCMR1\_OC1M\_1 | TIM\_CCMR1\_OC1M\_2); // 0110 = PWM mode 1

53.

54. TIM4->PSC = 0x7; // 16 MHz / (7+1) = 2 MHz timer ticks

55. TIM4->ARR = 0xFFFF008E; //(70+1) = 14.080 kHz rate; 64 entry = 220 Hz sine wave

56. TIM4->CCR1 = 0x23; // 50% duty cycle (35)

57. TIM4->CCER |= TIM\_CCER\_CC1E;

58.

59. //Enable Control Register 1 for Counting

60. TIM4->CR1 |= TIM\_CR1\_CEN;

61. }

62.

63. void DAC\_Config(void){

64. //Turn on Clock for DAC1

65. RCC -> APB1ENR1 |= RCC\_APB1ENR1\_DAC1EN;

66. //Configure DAC1 GPIO in Analog Mode 0x3

67. GPIOA->MODER |= GPIO\_MODER\_MODE4;

68. //Enable DAC1 Channel 1

69. DAC->CR |= DAC\_CR\_EN1;

70. }

71.

72. void UART\_config(void){

73. //Enable PA2 (TX) and PA3 (RX) to alternate function mode

74. GPIOA->MODER &= ~(0xF << (2\*2));

75. GPIOA->MODER |= (0xA << (2\*2));

76. //Enable alternate function for USART2 for the GPIO pins

77. GPIOA->AFR[0] |= 0x77 << (4\*2); //set pin 2 and 3 to AF7

78. //High Speed mode

79. GPIOA->OSPEEDR |= 0xF<<(2\*2);

80. //Pull up mode for PA3 RX

81. GPIOA->PUPDR &= ~(0xF<<(2\*2));

82. GPIOA->PUPDR |= 0x5<<(2\*2); //Select pull-up

83. //GPIO Output type: 0 = push-pull

84. GPIOA->OTYPER &= ~(0x3<<2);

85.

86. //Enable clk for USART2

87. RCC->APB1ENR1 |= RCC\_APB1ENR1\_USART2EN;

88. //Select system clock for USART2

89. RCC->CCIPR &= ~RCC\_CCIPR\_USART2SEL\_0;

90. RCC->CCIPR |= RCC\_CCIPR\_USART2SEL\_1;

91.

92. //Disable USART2

93. USART2->CR1 &= ~USART\_CR1\_UE;

94. //set data length to 8 bits

95. USART2->CR1 &= ~USART\_CR1\_M;

96. //select 1 stop bit

97. USART2->CR2 &= ~USART\_CR2\_STOP;

98. //Set parity control as no parity

99. USART2->CR1 &= ~USART\_CR1\_PCE;

100. //Oversampling to 16

101. USART2->CR1 &= ~USART\_CR1\_OVER8;

102. //Set up Baud rate for USART to 9600 Baud

103. USART2->BRR = 0x683; //1D4C

104. //USART2 Enable Receiver and transmitter

105. USART2->CR1 |= (USART\_CR1\_TE | USART\_CR1\_RE);

106.

107. //Enable interrupt for USART2

108. NVIC\_EnableIRQ(USART2\_IRQn);

109. //Enables interrupts for USART RX

110. USART2->CR1 |= USART\_CR1\_RXNEIE;

111.

112. //Enable USART2

113. USART2->CR1 |= USART\_CR1\_UE;

114.

115. //Verify that USART2 is ready for transmission

116. while ((USART2->ISR & USART\_ISR\_TEACK) == 0);

117. //Verify that USART2 is ready for reception

118. while ((USART2->ISR & USART\_ISR\_REACK) == 0);

119. }

**Init.h**

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. #include "queue.h"

11.

12. #define BufferSize 8

13.

14. static QueueHandle\_t but\_led\_queue;

15. static QueueHandle\_t but\_tim\_queue;

16. //static uint8\_t buffer[1];

17.

18. void LED\_task(void \*pvParameters);

19. void Button\_task(void \*pvParameters);

20.

21. void clock\_Config(void);

22. void gpio\_Config(void);

23.

24. void timer\_Config(void);

25. void DAC\_Config(void);

26. void TIM4\_IRQHandler(void);

27.

28. void UART\_config(void);

29. void USART\_read(uint8\_t uart\_buffer);

30. void USART2\_IRQHandler(void);

31. void change\_note(uint8\_t uart\_buffer);

32.

33. #endif