Wesley Kerr
Hayden Earl

Lab 2 Report - Audio Sine Wave

Procedure

For this lab, a circuit was built using an audio amplifier, a 0.047 µF capacitor, a 10Ω resistor, a 50KΩ potentiometer, and a speaker. The goal of this lab was to produce a 440 Hz sine wave through the speaker, to produce an *A* note. The circuit was built using the documentation included with the audio amplifier, according to **Figure 1**. After assembling, code was written to produce a sine wave through the speaker using the on-board DAC. The sine wave was approximated from a lookup table.

The clock was configured to the 16MHz HSI clock. Timer 4 was configured to the DAC instead of the SysTick timer, since FreeRTOS uses SysTick. The timer was put in edge-aligned, up-counting, master mode. The prescaler was set to 7: we needed it to be at least 2 to give us 2 MHz timer ticks (16 MHz/(7+1)=2 MHz). With 64 entries in the lookup table, to produce a 440.14 Hz sine wave we needed the interrupt rate to be 28.169 kHz (64*440.14=28.169 kHz. For this, we set the auto-reload register to 70, since 2 MHz/(70+1)=28.169 kHz. The capture-control register was set to 35 to get a 50% duty cycle, since we want the sine wave to be symmetric.

After setting these settings in the timer, we were able to output the approximated sine wave to the speaker.

Results

Writing configuration functions was the most difficult part of the lab. There are lots of different registers to change values in, especially for Timer 4. Prior to class on Monday, we went through the textbook from Microcontrollers. It was very helpful for enabling and configuring Timer 4 and DAC 1. We had some trouble with the interrupts at first, and realized we needed to move the NVIC_Enable line towards the end of our configuration function.

The other issue we encountered was the power supply. At first, we used an external source but received a lot of noise and clipping. After switching to the on-board 5V supply, we received a much better sound.

Wesley Kerr
Hayden Earl

Figures

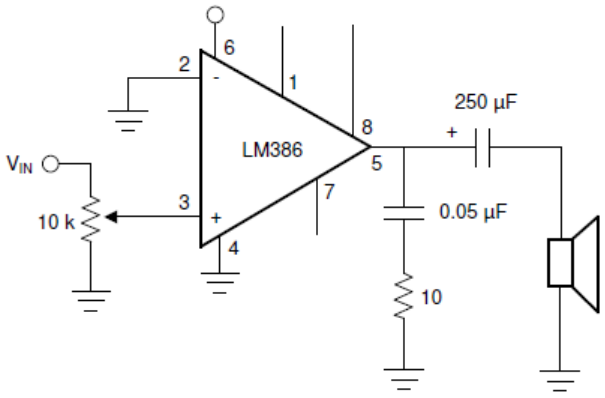

Figure 1. Circuit

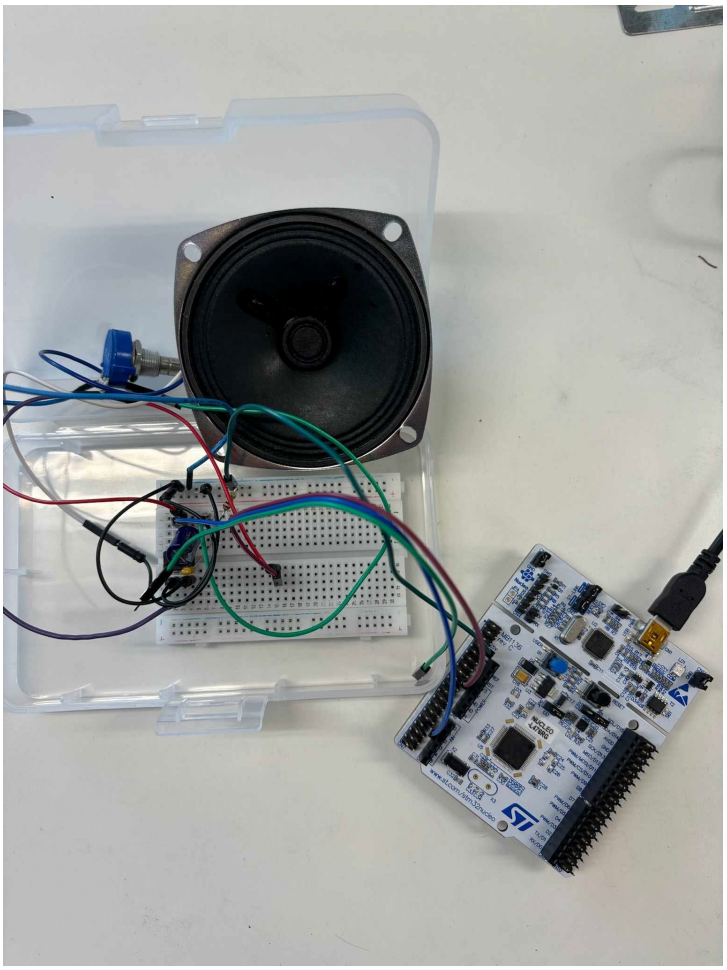

Figure 2. Assembled circuit

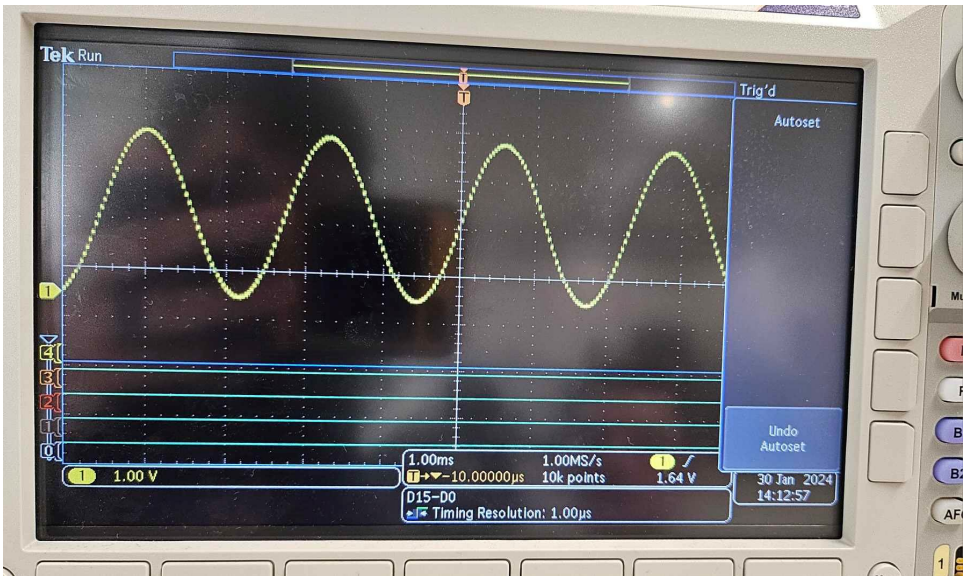

Figure 3. Sine wave on oscilloscope

Conclusion

We learned and re-learned a lot during this lab. A lot was reading through the datasheets and manuals, along with the textbook. After a few hiccups, we were able to get it all working and produce an *A* note (440 Hz).

Appendix- C Code

```c
#include <stdio.h>
#include "FreeRTOS.h"
#include "task.h"
#include <stdbool.h>
#include "stm32l476xx.h"

void clkConfig(void);
void gpioConfig(void);
void tim4Config(void);
void dacConfig(void); // DAC ch1 PA4
void vLED_Control(void *pvParameters); // led is PA5
void vButton_Control(void *pvParameters); // button is PC13, active low


void TIM4_IRQHandler(void);


bool BTN_ST = false; // global button state
unsigned int lookup_index = 0; // index for lookup table

const uint16_t sineLookupTable[] = {
0x200, 0x232, 0x264, 0x295, 0x2c4, 0x2f1, 0x31c, 0x345,
0x36a, 0x38c, 0x3aa, 0x3c4, 0x3d9, 0x3ea, 0x3f6, 0x3fe,
0x400, 0x3fe, 0x3f6, 0x3ea, 0x3d9, 0x3c4, 0x3aa, 0x38c,
0x36a, 0x345, 0x31c, 0x2f1, 0x2c4, 0x295, 0x264, 0x232,
0x200, 0x1ce, 0x19c, 0x16b, 0x13c, 0x10f, 0xe4, 0xbb,
0x96, 0x74, 0x56, 0x3c, 0x27, 0x16, 0x0a, 0x02,
0x00, 0x02, 0x0a, 0x16, 0x27, 0x3c, 0x56, 0x74,
0x96, 0xbb, 0xe4, 0x10f, 0x13c, 0x16b, 0x19c, 0x1ce};

/*
const uint16_t sineLookupTable[] = {
2048, 2248, 2447, 2642, 2831, 3013, 3185, 3346,
3495, 3630, 3750, 3853, 3939, 4007, 4056, 4085,
4095, 4085, 4056, 4007, 3939, 3853, 3750, 3630,
3495, 3346, 3185, 3013, 2831, 2642, 2447, 2248,
2048, 1847, 1648, 1453, 1264, 1082, 910, 749,
600, 465, 345, 242, 156, 88, 39, 10,
0, 10, 39, 88, 156, 242, 345, 465,
600, 749, 910, 1082, 1264, 1453, 1648, 1847};
*/
// this is table in hex for debugging purposes
/*
const uint16_t sineLookupTable[] = {
0x800, 0x8c8, 0x98f, 0xa52, 0xb0f, 0xbc5, 0xc71, 0xd12,
0xda7, 0xe2e, 0xea6, 0xf0d, 0xf63, 0xfa7, 0xfd8, 0xff5,
0xfff, 0xff5, 0xfd8, 0xfa7, 0xf63, 0xf0d, 0xea6, 0xe2e,
0xda7, 0xd12, 0xc71, 0xbc5, 0xb0f, 0xa52, 0x98f, 0x8c8,
0x800, 0x737, 0x670, 0x5ad, 0x4f0, 0x43a, 0x38e, 0x2ed,
0x258, 0x1d1, 0x159, 0xf2, 0x9c, 0x58, 0x27, 0x0a,
0x00, 0x0a, 0x27, 0x58, 0x9c, 0xf2, 0x159, 0x1d1,
0x258, 0x2ed, 0x38e, 0x43a, 0x4f0, 0x5ad, 0x670, 0x737};
*/

void clkConfig(void){
    RCC->CR|=RCC_CR_HSION;
    while((RCC->CR&RCC_CR_HSIRDY)==0);
    RCC->CFGR|=RCC_CFGR_SW_HSI;
    SystemCoreClockUpdate(); //updates FreeRTOS clock
}


void gpioConfig(void){
        //output for LED
    RCC->AHB2ENR|=RCC_AHB2ENR_GPIOAEN;
    GPIOA->MODER&=~GPIO_MODER_MODE5;//led
```

```
67      GPIOA->MODER|=GPIO_MODER_MODE5_0;
68          //input for BTN
69      RCC->AHB2ENR|=RCC_AHB2ENR_GPIOCEN;
70      GPIOC->MODER&=~GPIO_MODER_MODE13;
71  }
72
73
74
75  void tim4Config(void){
76      RCC->APB1ENR1 |= RCC_APB1ENR1_TIM4EN;        // Enable TIM4 clock
77      TIM4->CR1&=~TIM_CR1_CEN;
78      TIM4->CR1 &= ~TIM_CR1_CMS;    // Edge-aligned mode
79      TIM4->CR1 &= ~TIM_CR1_DIR;    // Up-counting
80
81      TIM4->CR2 &= ~TIM_CR2_MMS;    // Select master mode
82      TIM4->CR2 |= TIM_CR2_MMS_2;    // 100 = OC1REF as TRGO
83
84      TIM4->DIER |= TIM_DIER_TIE;    // Trigger interrupt enable
85      TIM4->DIER |= TIM_DIER_UIE;    // Update interrupt enable
86
87      TIM4->CCMR1 &= ~TIM_CCMR1_OC1M;
88      TIM4->CCMR1 |= (TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2);  // 0110 = PWM mode 1
89
90      TIM4->PSC = 7;             // 16 MHz / (7+1) = 2 MHz timer ticks
91      TIM4->ARR = 70;         // 2 MHz / (70+1) = 28.169 kHz interrupt rate; 64 entry
        look-up table = 440.14 Hz sine wave
92      TIM4->CCR1 = 35;         // 50% duty cycle
93      TIM4->CCER |= TIM_CCER_CC1E;
94      TIM4->CR1 |= TIM_CR1_CEN;    // Enable timer
95          NVIC_EnableIRQ(TIM4_IRQn); // this makes the interupt actually get called
96  }
97
98
99
100
101
102 /*
103 void tim4Config(void){
104     //enable clock
105     RCC->APB1ENR1|=RCC_APB1ENR1_TIM4EN;
106     //configure control registers
107     TIM4->CR1&=~TIM_CR1_DIR; //controls direction of timer (up/down)
108     TIM4->CR1&=~TIM_CR1_CMS;
109     TIM4->CR2&=TIM_CR2_MMS;
110     TIM4->CR2|=TIM_CR2_MMS_2;
111
112     //configure dma/interrupt control
113     TIM4->DIER|=TIM_DIER_TIE;
114     TIM4->DIER|=TIM_DIER_UIE;
115     NVIC_EnableIRQ(TIM4_IRQn);
116     //NVIC_SetPriority(TIM4_IRQn,0);
117     //output compare mode
118
119     //configure prescalar
120     //440*64=28160
121     // divide by 8
122     TIM4->PSC=7;
123     //configure ARR
124     TIM4->ARR=770;
125     //set duty cycle
126     TIM4->CCR1=30; //or 279? 35usec to update table
127     TIM4->CCMR1&=~TIM_CCMR1_OC1M;
128     TIM4->CCMR1|=(
129     TIM4->CCMR1|=TIM_CCMR_CC1S_EN;
130     TIM4->CCER|=~TIM_CCER_CC1P_EN;
131     TIM4->CCER|=TIM_CCER_CC1E_EN;
```

```
132         //set CCxIE and/or CCxDE bits for interrupt
133         //URS 1?
134 
135         TIM4->MMS|=100;
136         //configure as output
137 
138         //TIM4->CCER|= ;
139         //TIM4->BDTR|= TIM_BDTR_MOE //main output enable
140         //enable timer
141         TIM4->CR1|=TIM_CR1_CEN;
142     }
143     */
144 
145     void dacConfig(void){ //double check this
146         RCC->APB1ENR1|=RCC_APB1ENR1_DAC1EN;
147         //disable to change settings
148         DAC->CR&=~(DAC_CR_EN1); // Disable DAC
149 
150         GPIOA->MODER &= ~(GPIO_MODER_MODE4);//dac
151         GPIOA->MODER |= 0x00000300;
152 
153         DAC->CR |= DAC_CR_TEN1;
154         DAC->CR &= ~(DAC_CR_TSEL1);
155         DAC->CR |= DAC_CR_TSEL1_0;
156         DAC->CR |= DAC_CR_TSEL1_2;
157 
158         // DAC->MCR &=0xFFFFFFF8 // may need to try this instead of the nex three lines
            ======
159         DAC1->MCR &= ~(DAC_MCR_MODE1_0);
160         DAC1->MCR |= DAC_MCR_MODE1_1;
161         DAC1->MCR &= ~(DAC_MCR_MODE1_2);
162         //================================================================================
163 
164         DAC->CR |= DAC_CR_EN1; //enable
165 
166     }
167 
168     void vLED_Control(void *pvParameters){ // led is PA5
169         while (true) {
170                 if(BTN_ST){
171                     //turn led on and send sine wave to speaker
172                     GPIOA->ODR |=0x20;
173                     //DAC_DHR12R1|= ;
174                     DAC1->CR |= DAC_CR_EN1; //enable
175                 }else{
176                     //turn led off
177                     GPIOA->ODR &=0xFFDF;
178                     DAC1->CR &= ~DAC_CR_EN1; //enable
179                 }
180         }
181     }
182 
183     void vButton_Control(void *pvParameters){ // button is PC13, active low
184 
185         while (true) {
186                 bool button = (GPIOC->IDR&0x2000);
187 
188                 if(!button && !BTN_ST){ // Poll button state
189                     BTN_ST=true;
190                 }
191                 else if(!button){
192                     BTN_ST=false;
193                 }
194           while(!button){
195                     button = (GPIOC->IDR&0x2000);
196                 }
```

```
197
198         //vTaskDelay(pdMS_TO_TICKS(10)); // Polling interval
199     }
200 }
201
202
203
204 void TIM4_IRQHandler(void){
205     //DAC->DHR12R1 &= 0x000; // reset the DAC value
206     if((TIM4->SR & TIM_SR_CC1IF)!=0){
207         DAC->DHR12R1 &= 0xFFFFF000; // reset the value
208         DAC->DHR12R1 |= sineLookupTable[lookup_index]; // Load new value from table
209         lookup_index ++; // increment index
210
211         if(lookup_index > 63){ // check if the index is out of range
212             lookup_index = 0; //reset the index for the lookup table
213         }
214         TIM4->SR &= ~TIM_SR_CC1IF;
215     }
216     //check for overflow, generate interrupt
217     if((TIM4->SR&TIM_SR_UIF)!=0){
218         TIM4->SR &= ~TIM_SR_UIF;    //clear interrupt bit
219     }
220
221
222     //clear interrupt bit
223 }
224 int main(void){
225     TaskHandle_t ledHandle=NULL;
226     TaskHandle_t btnHandle=NULL;
227     //TaskHandle_t sinHandle=NULL;
228
229     //setup hardware
230     clkConfig();
231     gpioConfig();
232     tim4Config();
233     dacConfig();
234
235
236     //setup tasks
237     if(xTaskCreate(vLED_Control, "LED_Control", 10,NULL, 1, &ledHandle)==pdFAIL){
238         // need to set value to stack size
239         while(1);
240     }
241     if(xTaskCreate(vButton_Control, "Button_Control", 10,NULL, 1, &btnHandle)==pdFAIL){
242             while(1);
243     }
244
245     //start task scheduler
246     vTaskStartScheduler();
247
248     while(1);
249     //return 0;
250 }
251
```