ECSE 444 Lab 3 Report

DAC, Timers, Interupts, DMA and Analog Interfacing

Group 11: Byron Chen, 260892558, Kevin Li, 260924898

***Abstract*—The main purpose for this lab is to using DAC of STM32 to output different music notes through a speaker. In this lab, three different methods and three different kind of waves, i.e. triangle wave, saw wave and sine wave, are taken to achieve this. After the sound comparing, sine wave is taken to produce the sound. The methods we used to output the music notes are polling, timer interrupts and DMA and the last one is compared to be the best way to achieve our goal. By changing the frequency of the sine wave, different music notes are successfully made by the DAC. Furthermore, a push button is implemented to be able to switch between different notes through interrupts instead of polling.**

***Index Terms*—DAC, Timers, Interrupts, DMA, STM32**

I. Introduction

DIGITAL to Analog Converter (DAC), is a necessary part in processors for producing analog signals. In this lab, we will use DAC on STM32 to produce different music notes by using different methods. In order to find the way of producing the best sound, three different kinds of waves are used through the whole lab. Furthermore, instead of polling, timer interrupts and DMA are also used as the way of outputting signals in order to make the accurate music notes. Apart from this, a push button is also implemented by interrupts for the usage of switching between the different notes.

II. Materials and Methods

This lab can be mainly divided into 2 parts and the second part is worked through 4 steps, they are [1]:

1) Making signals, i.e. triangle, saw and sine, and outputting by polling the variables.
2) Making different music notes by using sine waves with different frequencies trough these steps:
    a) Implementing Push-button Interrupts
    b) Implementing Timer-driven DAC Output
    c) Driving DAC with Timer and DMA
    d) Putting it All Together and Multiple Tone Generation

Before anything has been done, there are two important points to be noticed. First of all, DAC channels and SWV data trace timeline graph should be enabled from CubeMX and CubeIDE respectively, in order to view the timeline of the waves created in this lab. Secondly, The frequency in the debugger configuration should be set the same as that of HCLK for STM32 and the frequencies for the STM in part 1 and 2 are 120MHz and 80MHz respectively.

For part 1, three different arrays that represent the different waves, i.e. triangle, saw and sine, are created before the main program. In this lab, the length of the array is set to be 16, and since the signal is asked to be oscillating with a period of 15ms, a delay time with 0.9375ms is set using HAL_DELAY after each time the element of the array is outputted. In this lab, the precision of the DAC is chosen to be 12-bit, which means that the amplitude should be from 0 to 4095.

The implementation of the triangle wave is shown in Listing 1.

```
for (uint32_t i = 0; i < 16; i++) {
  if (i < 8) {
    triangle_wave[i] = i * 512;
  }else if (i == 8) {
    triangle_wave[i] = 4095;
  }else {
    triangle_wave[i] = 4096 - (i - 8) * 512;
  }
}
```

Listing 1. Creation of the Triangle Wave

Since the maximum amplitude of the triangle wave is 4095, the right middle of the array should be the highest point, i.e. 9$\hat{\text{th}}$ element of the triangle wave, and is set to be 4095. For the elements left half of the triangle wave array, the values are increasing by $\frac{4095+1}{8} = 512$ from 0 and for the right portion of the array, they are decreasing by 512.

The method of generating the saw wave array is similar to the triangle wave array. The only difference is that the triangle wave reaches its maximum at the middle of the period, whereas the saw wave reaches its maximum at the end of its period. As a result, the value of the elements in the saw wave are increasing by $\frac{4095+1}{16} = 256$ from 0. As a result, the maximum amplitude of the saw wave should be $256 \times 15 = 3840$. The implementation of the saw wave is shown in Listing 2.

```
for (uint32_t i = 0; i < 16; i++) {
  saw_wave[i] = i * 256;
}
```

Listing 2. Creation of the Saw Wave

In order to create a sine wave efficiently, arm_sin_f32 in arm_math library is used in this lab. Since the period of a sine wave is 2π in radians, and the length of the array for representing the sine wave is 16, this wave should reach the maximum amplitude at $\frac{\pi}{2}$ and minimum amplitude at $\frac{3\pi}{2}$. This means that, the wave should reach the turning points when outputting the 5^{th} and the 13^{th} elements. Furthermore, since the DAC can only output values from 0 to 4095, the value is calculated as shown in equation 1.

$$sine\_value = 2047.5 \times (1 + sin(radians)) \tag{1}$$

The implementation of the sine wave is shown in Listing 3.

```
for (uint32_t i = 0; i < 16; i++) {
  float32_t radians = 3.14 / 8 * i;
  sine_wave[i] = (uint32_t) (2047.5 * (1 +
arm_sin_f32(radians)));
```

```
    }
```

Listing 3. Creation of the Sine Wave

In order to output the sound from the DAC, PA4 is connected to DAC1_OUT1 and PA5 is connected to DAC1_OUT2 as shown in Figure 1 [2]. According to Figure 2 found in Schematic [3], the board is connected as shown in Figure 3 when outputting Channel 1 and connected as shown in Figure 4 when outputting Channel 2 [1].

For part 2 step 1, The push button is implemented by using interrupt instead of polling. This means that an interrupt is generated whenever the button is pushed. In this step, the LED 1 would toggle when pushing the button. To enable LED 1, PA5 is set to be GPIO_Out instead of DAC_OUT2 which means the no sound would be made when the speaker is connected to port D13. The implementation of this step requires overwriting the HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) function. Whenever this function is called, the input is checked with the button pin in order to ensure that it is the push button who is calling the GOIO interrupt. The implementation of this step is shown in Listing 4.

```
    void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
    {
      if ((int) GPIO_Pin == (int) my_button_Pin) {
      // If the GPIO pin trigger the interrupt is
    for the button, toggle the LED pin
        HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
      }
    }
```

Listing 4. Push Button Callback Function

For step 2, since the sampling rate is chosen to be 44.1kHz where as the system clock frequency is chosen to be 80MHz, the counter of the timer is set to be $round(\frac{80Mhz}{44.1khz}) = 1814$. In order to enable the interrupt of the timer, the function HAL_TIM_PeriodElapsedCallback is needed to be overwritten. Whenever this function is called, the input is checked with TIM2 in order to ensure that it is the timer 2 who is calling the peroid elapsed interrupt. The implementation of this step is shown in Listing 5.

```
    void HAL_TIM_PeriodElapsedCallback(
    TIM_HandleTypeDef *htim) {
      if (htim->Instance == TIM2) {
        // If the Timer trigger the interrupt is
    TIM2
        HAL_IncTick();

        sine = sine_wave_1kHz[TIM_wave_index];
        HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1,
    DAC_ALIGN_12B_R, sine);
        TIM_wave_index = (TIM_wave_index + 1) % 44;
      }
    }
```

Listing 5. Timer Callback Function

As shown in Listing 5, the elements of the array storing the sine wave with 1kHz is outputted on by one circularly. By using the equation 2, the length of the sine wave 1kHz is calculated to be 44.

$$Length = \frac{System\ clock\ frequency}{counter * frequency\ of\ the\ signal} \quad (2)$$

Furthermore, since the DAC value should be scaled by 2/3 to ensure the sound made to be continuously, the amplitude of the since waves is calculated to be:

$$4095 \times \frac{2}{3} \times \frac{1}{2} = 1365$$

As a result, the implementation of the sine wave with 1kHz is shown in Listing 6.

```
    for (uint32_t i = 0; i < 44; i++) {
        float32_t radians = 3.14 / 22 * i;
        sine_wave_1kHz[i] = (uint32_t) (1365 * (1 +
    arm_sin_f32(radians)));
    }
```

Listing 6. Creation of the Sine Wave with 1kHz

To explain the methods more clearly, step 3 and 4 are combined in this report. In this lab, the sine waves with four different frequencies, i.e. 0.5kHz, 1kHz, 2kHz and 4kHz, are chosen to make different tones. According to equation 2, the lengths are calculated to be 10, 22, 44 and 88 respectively. To make sure the user could switch the tones by pressing the push button, the sine waves with different frequencies are stored in the global array called sine_waves and their lengths are stored in the global array called lengths. The implementation for the DMA and tone changing is shown in Listing 7.

```
    void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  if ((int) GPIO_Pin == (int) my_button_Pin) {
  // If the GPIO pin trigger the interrupt is for
    the button, toggle the LED pin
    HAL_GPIO_TogglePin(my_LED_GPIO_Port, my_LED_Pin)
    ;

    // Restart the DMA and update the sine_num
    HAL_DAC_Stop_DMA(&hdac1, DAC_CHANNEL_1);
    HAL_DAC_Start_DMA(&hdac1, DAC_CHANNEL_1,
    sine_waves[sine_num], lengths[sine_num],
    DAC_ALIGN_12B_R);
    sine_num = (sine_num + 1) % 4;
  }
}
```

Listing 7. DMA Implementation

Apart form the global arrays, i.e. sine_waves and lengths, a global variable that represents the index of the sine wave is also created, namely sine_num. In order to iterate the global arrays circularly, sine_num is incremented by 1 each time and moding by the length of the global array, i.e. 4. This means that, when the button is pressed, sin_num will be increased by one and whenever the sin_num reaches 5 it will be changed back to 0. As a result, the tones made will be switching in sequence of 0.5kHz, 1kHz, 2kHz and 4kHz circularly.

The implementation of DMA only requires to call the function HAL_DAC_Stop_DMA which stop the previous DMA conversion and HAL_DAC_Start_DMA which output the data in an array of a given length.

III. Results

For part 1, the outputs of the waves, i.e. triangle, saw and sine, are shown in Figure 5 and 6. By listening to the sounds they produced, the saw wave has the largest discontinuity whereas the sine wave has the best sound among these three waves. Furthermore, the LED 1 shown in Figure 7 keeps

blinking when this program is working. We found that the LED 1 is bling with the same frequency as the DAC 1 Channel 1 does, and when the line is connected to Channel 2, i.e. D13, the brightness of the LED 1 becomes smaller. By looking through the Schematic of the board, we found that the ARD.D13 occupies the same port as the LED 1 does which is shown on Figure 2. As a result, it is reasonable for the LED 1 to bilking with the same frequency as the Channel 2 does.

For part 2, LED 1 and push button is tested to be working properly by pushing the button and observing the toggle of the LED 1. Furthermore, when using the timer, the sound is made continuously and pitch is the same as the first sound made by the DMA. Apart from this, the tone is changing properly whenever the push button is pushed and nothing is outputting before pushing the button. Also, to test whether the DAC Channel 2 is disabled or not, the wire was connected to D13 and nothing is outputted from the speaker.

IV. Discussion & Summary

By comparing all kinds of the sound waves, we found the sine waves can make the best and most continuously music tone by using DMA.

V. Work Distribution

Both Byron and Kevin participated all parts through the lab. Byron is mainly responsible for implementing the part 1 and the DMA with different music notes; Kevin is mainly responsible for implementing the push button interrupts and Timer-driven DAC output.

Appendix A
Pictures

Table 145. DAC features

DAC features	DAC1
Dual channel	X
Output buffer	X
I/O connection	DAC1_OUT1 on PA4, DAC1_OUT2 on PA5
Maximum sampling time	1MSPS
Autonomous mode	-

Fig. 1. The Features of DAC [2]

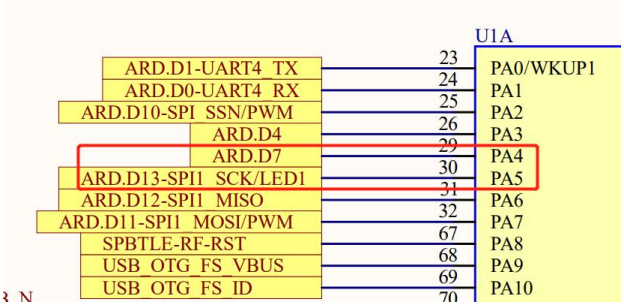

Fig. 2. Connection of I/O [3]

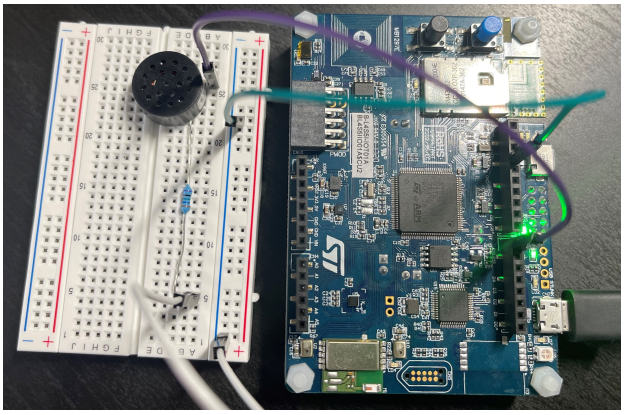

Fig. 3. The Connection of the Board When Outputting DAC Channel 1.

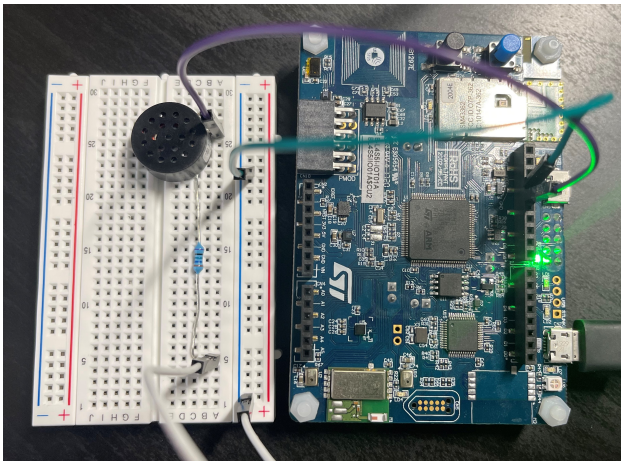

Fig. 4. The Connection of the Board When Outputting DAC Channel 2.

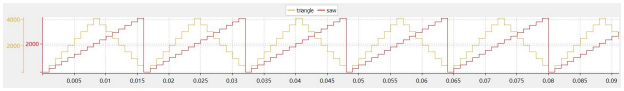

Fig. 5. The Output Waves of Triangle and Saw Waves.

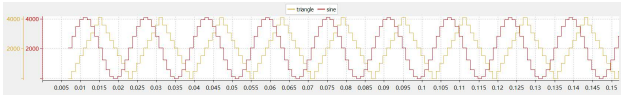

Fig. 6. The Output Waves of Triangle and Sine Waves.

References

[1] Project Description.

[2] STM32L4+ Series advanced Arm®-based 32-bit MCUs. STM.

[3] IOT Node Discovery Kit. STM.