ECSE444 lab3

Simon Li
dept. electrical eng
McGill University
Montreal, Canada
xi.yang.li@mail.mcgill.ca

Emily Li
dept. electrical eng
McGill University
Montreal, Canada
yue.l.li@mail.mcgill.ca

I. Introduction

This report elucidates the results and findings derived from a laboratory experiment [1] conducted on the STM32L4S5VI microcontroller unit (MCU), which was specifically designed for the utilization of General Purpose Input/Output (GPIO) and the Digital-to-Analog Converter (DAC) to generate audio outputs. The experiment's primary focus was to practice the implementation of a timer (TIM), interrupts, and Direct Memory Access (DMA). Various waveforms with different frequencies were generated using multiple methods and distinct hardware components. The ensuing sections provide a comprehensive overview of the methodologies employed, the results procured, and the crucial insights extracted from this experiment.

II. DAC Signal Generation

The generation of signals at audible frequencies was accomplished using a Digital-to-Analog Converter (DAC). The DAC allowed the approximation of ideal signals by converting digital values within the program and subsequently feeding them to the DAC. The STM32 board's DAC unit is equipped with two channels, which enables the simultaneous generation of two distinct signals.

Upon activation, the main function of the program initializes the DAC. Two separate methods were employed for inputting the triangle and sawtooth signals into the program:

1) Utilizing a loop and an up/down counter to compute the discrete values at varying signal levels, and writing these values to the DAC.
2) Predefining two arrays containing the different signal levels and iteratively reading and writing these values to the DAC within the main function loop.

The frequencies of the generated signals were controlled using the HAL_Delay() function. This function allowed for the configuration of the duration of each voltage level written to the DAC channels, ensuring a signal period of 15 ms (corresponding to a frequency of 67 Hz). It is essential to consider the DAC's operational precision, which employs 12 bits of right alignment and loads uint16 values to the channels with a maximum value of 4096.

When examining the manually generated waveform on a speaker and oscilloscope, it was observed that the waveform exhibited distinct digital levels rather than a smooth, ideal

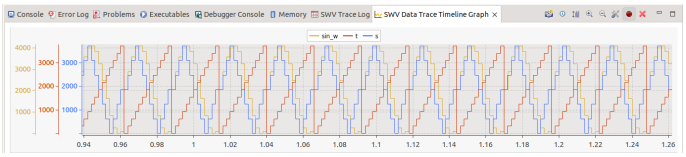

Fig. 1. Signal Waveform

```c
int k=0;
float angle;
uint32_t s;
uint32_t t;
int sin_i = 0;
int status = 0;
uint32_t sin_wave[5][15];
uint32_t sine_table1[15];
uint32_t sine_table2[60];
uint32_t sine_table3[120];
uint32_t sine_table4[30];
uint32_t * wave;
int length;
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
      // Generate a lookup table of sine values
          for (i = 0; i < 15; i++)
          {
              angle = 2*M_PI * i / 15;
              sine_table1[i] = (uint32_t)(2048 + 2047 * arm_sin_f32(angle));
          }
          for (i = 0; i < 60; i++)
                {
                    angle = 2*M_PI * i / 60;
                    sine_table2[i] = (uint32_t)(2048 + 2047 * arm_sin_f32(angle));
                }
          for (i = 0; i < 120; i++)
                          {
                              angle = 2*M_PI * i / 120;
                              sine_table3[i] = (uint32_t)(2048 + 2047 * arm_sin_f32(angle));
                          }
          for (i = 0; i < 30; i++)
                          {
                              angle = 2*M_PI * i / 30;
                              sine_table4[i] = (uint32_t)(2048 + 2047 * arm_sin_f32(angle));
                          }
```

Fig. 2. Waveform Generation

waveform. Additionally, the frequency was not perfectly accurate. As a result, alternative methods for signal generation were explored and tested to improve the quality and accuracy of the generated signals.

III. Advanced Signal Generation using DAC, Timer Interrupt, and CMSIS DSP Library

In this approach, a 2D signal array was constructed using the CMSIS DSP library's arm_sin_f32 function, which generated 15 distinct values per period to accurately represent the signal without significantly affecting the frequency. The frequency of the signal was adjusted by multiplying the different levels by various multiples of π. Four separate sine waves were created

and assigned to the first four rows of the array, whereas the last row was filled with zeros to allow for the deactivation of the signal.

The external interrupt of the push button was configured to control the varying sound wave frequencies output by the DAC. After setting up the NVIC interrupt and enabling EXTI for the push button pin in CubeMX, the desired action to be executed upon pressing the button was incorporated into the HAL_GPIO_EXTI_Callback() function.

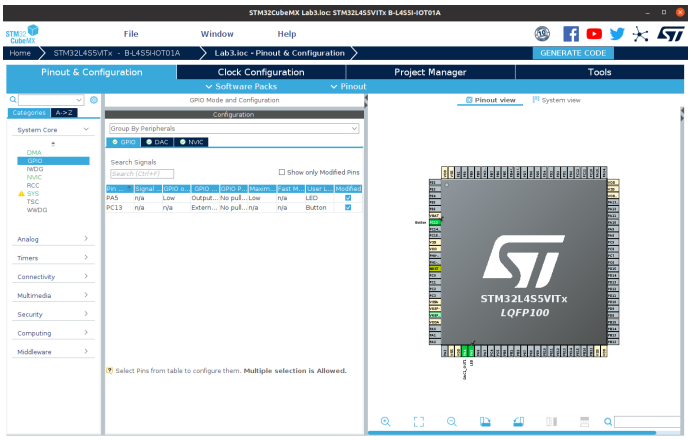

Fig. 3. CubeMX Settings

To attain higher precision in signal frequency, a timer and its corresponding interrupt routine were enabled in CubeMX. The specific action to be performed within the timer interrupt was added to the HAL_TIM_PeriodElapsedCallback() function. The sampling frequency was taken into account when determining the counter period for the timer. Considering the upper limit of human hearing is 20 kHz, a sampling rate of approximately 40 kHz (Nyquist) [2] was selected, corresponding to a counter value of around 3000 with an internal clock of 120 MHz.

Upon pressing the button, the status variable of the button was updated. Incrementing this value facilitated the execution of various tasks based on the variable's value, such as switching between distinct signal arrays to be written to the DAC. An LED indicator was toggled with each button press.

Within the timer interrupt, the values from the selected signal array were iteratively written to the DAC channel. Signal array selection was accomplished by performing a modulo operation on the button status and dividing by 5 to obtain the remainder. These values were then used as the index for the first row of the array containing the diverse signals. The push button facilitated the control of the DAC signal in this manner.

IV. Signal Generation with DAC and DMA

An alternative method for generating signals with the DAC involves using the MCU's Direct Memory Access (DMA). The advantages of employing DMA include faster transfer rates than the CPU, reduced CPU computation, and decreased program latency [3]. After enabling DMA in CubeMX, a

```
415 void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin){
416         if(GPIO_Pin == mybutton_Pin){
417                 HAL_GPIO_TogglePin(myled2_GPIO_Port, myled2_Pin); // Toggle LED
418                 status++;
419
420         }
421 }
422 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim){
423         if (status ==4) {
424                         status = 0;
425                 }
426                 if (status %4 ==0) {
427                         wave = sine_table1;
428                         length  = 15;
429                 } else if (status %4 ==1) {
430                         wave = sine_table2;
431                         length  = 60;
432                 } else if (status %4 ==2) {
433                         wave = sine_table3;
434                         length  = 120;
435                 } else if (status %4 ==3) {
436                         wave = sine_table4;
437                         length  = 30;
438                 }
439         HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, wave[sin_i++]);
440         if(sin_i==15){
441                 sin_i=0;
442         }
443 }
```

Fig. 4. Interrupt Functions

request was added for channel 1 of the DAC, which was set up to read cyclically from an array of uint16 (half-word) values. The DAC could then be initiated in DMA mode by specifying the channel, the memory location of the array, and the array's length. The DAC subsequently read values from the memory location using DMA and looped through the sine wave array without employing a timer interrupt or CPU power. To change the signal generated using DMA with a push button interrupt, the DMA could simply be halted upon pressing the button and restarted with the new signal array's memory location. Signal selection was performed in the same manner as with the timer.

V. Lookup Table Generation for Sine Waves

In this section, we discuss the creation of lookup tables for various sine waves. The objective is to precompute the sine values for different frequencies, which will be utilized later in the program. The code snippet provided defines several variables and arrays, initializes them, and populates the sine lookup tables.

First, the necessary variables and arrays are defined:

sin_i and status are integer variables initialized to 0. sin_wave is a 2D array with 5 rows and 15 columns for storing sine values. sine_table1, sine_table2, sine_table3, and sine_table4 are arrays for storing the precomputed sine values of different frequencies. wave is a pointer to a uint32_t type that will be used to point to the current sine wave array. length is an integer variable that will store the length of the current sine wave array. In the main function, four distinct lookup tables are generated for sine waves with varying frequencies. Each table corresponds to a specific frequency, and the sine values are computed using the arm_sin_f32() function from the CMSIS DSP library. The sine values are calculated based on the angles derived from the fractions of the full circle (2 * pi). The different frequencies are achieved by altering the number of samples per period in each sine table.

The following steps are performed for each sine table:

A loop iterates through the required number of samples for the specific sine table (15 for sine_table1, 60 for sine_table2, 120 for sine_table3, and 30 for sine_table4). For each iteration,

the angle is calculated by multiplying 2 * pi with the current sample index divided by the total number of samples. The sine value is computed using the arm_sin_f32() function with the calculated angle as input. The sine value is then normalized to fit within the DAC range (0 to 4095) by adding 2048 and multiplying by 2047. Finally, the normalized sine value is stored in the corresponding sine table at the current index. Upon completion of these steps, the four sine tables (sine_table1, sine_table2, sine_table3, and sine_table4) contain precomputed sine values for different frequencies. These lookup tables can be utilized for efficient sine wave generation in the subsequent sections of the program.

VI. Sine Wave Frequencies and EXTI Callback Explanation

A. Sine Wave Frequencies

The timer sampling rate is set to 44.1 kHz, which is a common sampling rate used in digital audio processing. Given that the frequency of each sine wave is obtained by dividing the timer sampling rate (44.1 kHz) by the length of each wave array, we can calculate the frequencies for each sine wave as follows:

sine_table1: 44.1 kHz / 15 = 2.94 kHz sine_table2: 44.1 kHz / 60 = 735 Hz sine_table3: 44.1 kHz / 120 = 367.5 Hz sine_table4: 44.1 kHz / 30 = 1.47 kHz These frequencies represent the different sine waves that can be generated using the precomputed lookup tables.

B. HAL_GPIO_EXTI_Callback Function

The HAL_GPIO_EXTI_Callback function is an interrupt callback function that is triggered when an external interrupt occurs on the specified GPIO pin. In this case, it is triggered by the push button (mybutton_Pin). The following actions are performed inside the callback function:

The LED (myled2_Pin) is toggled to indicate that the button has been pressed. The status variable is incremented to cycle through the different sine waves. If the status variable reaches 4, it is reset to 0, as there are only four sine waves available. The DAC DMA is stopped to change the sine wave. Depending on the value of status % 4, the wave pointer is assigned to one of the sine tables (sine_table1, sine_table2, sine_table3, or sine_table4), and the length variable is set to the corresponding length of the sine table (15, 60, 120, or 30). The DAC DMA is started again with the new sine wave array, using HAL_DAC_Start_DMA() function with the updated wave pointer and length value. This callback function allows the user to switch between different sine waves by pressing the push button. When the button is pressed, the corresponding sine wave is generated based on the status variable. This provides a convenient way to control the output frequency of the DAC.

VII. Conclusion

In this lab, various methods for generating audio signals using the STM32L4S5VI microcontroller unit (MCU) were explored and evaluated. The Digital-to-Analog Converter (DAC) was employed to convert digital values into analog signals, while different techniques, such as timer interrupts, Direct Memory Access (DMA), and the CMSIS DSP library, were utilized to generate and control the waveform frequencies. Precomputed sine wave lookup tables were created to enhance the efficiency and accuracy of signal generation. Furthermore, the implementation of a push button interrupt allowed users to switch between multiple frequencies seamlessly.

The experiment provided valuable insights into the various hardware components and their practical applications. Moreover, the results demonstrated the advantages of using DMA for faster data transfer rates, reduced CPU computation, and decreased program latency. The findings of this study contribute to the understanding of signal generation methodologies and can be applied in future research and development of audio systems and signal processing applications.

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin){
        if(GPIO_Pin == mybutton_Pin){
                HAL_GPIO_TogglePin(myled2_GPIO_Port, myled2_Pin); // Toggle LED
                status++;
                if (status ==4) {
                        status = 0;
                }
                HAL_DAC_Stop_DMA(&hdac1,DAC_CHANNEL_1);
                if (status %4 ==0) {
                        wave = sine_table1;
                        length  = 15;
                } else if (status %4 ==1) {
                        wave = sine_table2;
                        length  = 60;
                } else if (status %4 ==2) {
                        wave = sine_table3;
                        length  = 120;
                } else if (status %4 ==3) {
                        wave = sine_table4;
                        length  = 30;
                }

                HAL_DAC_Start_DMA(&hdac1,DAC_CHANNEL_1, wave,length,DAC_ALIGN_12B_R);
        }
}
```

Fig. 5. DMA

References

[1] "Lab 3: DAC, Timers, Interupts, DMA and Analog Interfacing" Winter 2023.

[2] Weisstein, Eric W. "Nyquist Frequency." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/NyquistFrequency.html

[3] Jim Jeffers, James Reinders, in Intel Xeon Phi Coprocessor High Performance Programming, 2013. Available: https://www.sciencedirect.com/topics/computer-science/direct-memory-access