Lab 3: DAC, Timers, Interrupts, DMA and Analog Interfacing

Kaicheng Wu
Department of Electrical and Computer Engineering
McGill University
Montreal, Canada
kaicheng.wu@mail.mcgill.ca
260892789

Tian Feng
Department of Electrical and Computer Engineering
McGill University
Montreal, Canada
tian.feng@mail.mcgill.ca
260913386

***Abstract*—In this lab, we implemented a program that uses the DAC and the buzzer to generate observable audio outputs.**

***Index Terms*—DAC, Timer, Interrupts, DMA, Analog Interfacing, STM32**

I. Introduction

The General Purpose Input/Output (GPIO) pins are interfaces on controllers that could be programmed to perform various functions, such as digital bit input reading, digital pin output, and analog quantity reading/writing.

STM32L4+ processors also have multiple in-built timers, which could be used to generate various signals and interrupts and access the DMA.

For this experiment, we will implement a program that observes interrupts and generates different audio patterns depending on whenever a push is detected.

II. Implementation

The board will automatically play the DMA-single tone of 200Hz when loaded for 5 sec. Then the DAC will play a triangle wave(or sawtooth). If the button is pushed the DAC will output C4, E4, and G4 in a loop. The timer-interrupt-callbacks is commented and the section needs to be enabled(in user section 4) to play a 1 kHz sound.

A. Wave Functions

To start, we implemented three functions to generate different types of waveform signals: triangular wave, sawtooth wave, and sine wave. All three functions take an integer counter representing time and return uint8 values representing the amplitude of the waveform at a particular point in time. Similarly, for all three waves, the unsigned integer 0 is used to represent the minima of the wave whereas the unsigned integer 255 is used to represent the maxima of the wave. All the wave functions use a direct lookup table so that the time complexity is O(1).

The triangular wave generated exhibits a symmetrical rise and fall with a peak amplitude of 255 and a trough amplitude of 0, oscillating at a frequency of 125Hz. The sawtooth wave generated has a linear rise from 0 to 255 followed by an immediate drop to 0, repeating at a frequency of 100Hz. Lastly, the sine wave generated is a smooth oscillation with values derived from a predefined array, exhibiting a sinusoidal shape with an amplitude range between 0 and 255. The sine wave lookup-table generator takes the frequency as a parameter and generates an array of length 2-2000 corresponding to 20- 20 kHz. According to the Nyquist sampling theorem, the sampling rate of 40kHz for the DAC is feasible.

B. Sound Creation

For the low-frequency waves each analog output has an interval of 1 ms, we redirected each signal to a different DAC output channel through `HAL_DAC_SetValue` and used `HAL_Delay` to insert a pause between each cycle. A sample waveform produced can be found in the appendixes. Wiring a buzzer to the breadboard allows us to generate the sound. For sounds of higher frequencies, we chose the TIM interrupts and DMA to make more accurate and fast clock signals.

C. Push Button Interrupt

The push button is connected to an external interrupt line pin to enable interrupt callbacks to change the working frequency of the DAC. First, we wrote a blink() function but we found that any *HAL_Delay()* should not be used in the ISR or the interrupt will get stuck. Then we use the on/off of the LED to indicate the DAC status, where no light means no output, and when the DAC is on, the frequencies loops in C4, E4, and G4 each time the button is pushed. The G4 is then followed by the stop of the DAC.

D. Timer Interrupts the DAC

We first tried to use TIM6, the simplest timer; however, only those 32-bit counters seemed to work. We finally chose TIM2 with Prescaler set to zero and Period Reload register set to 2999. Such settings will result in a 40 kHz DAC clock providing an input clock of 120 MHz by calculating 120M/((1+0)(1+2999)) Hz. Timers are started in the interrupt mode and when the counter period overflows the timer will send an IRQ. Now we can use the *set_Value* functions in a higher frequency called by the timer elapse callback function.

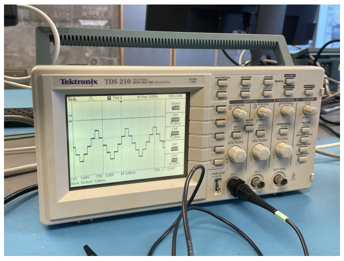

Fig. 1. Triangle wave shape for part1

E. DAC with Direct Memory Access (DMA)

Then we restart the DAC in DMA mode wrapped in the button callback function to play multiple tones in a loop. The DMA will send data directly to the DAC repeatedly at the frequency of the TIM2 so the CPU is free for other tasks.

F. Clipping Control

We also implemented a clipping control function for the wave arrays to limit the current if any clipping happens. The drawback is that the smaller sounds of the 8-bit format will have an even lower dynamic range with a clipping control.

III. Constraints and Limitations

Due to time constraints, some parts of our program were not properly organized before the demo, the multiple-tone test works well after we remove all *HAL_Delay()* functions in the button interrupt callback. Also, the lack of proper version control negatively impacted our project as STM32CubeIDE's code generation introduced some unwanted changes. For the next lab and the final project, we will be using git for version control.

IV. Conclusion

To conclude, we implemented a program that uses the DAC and the buzzer to generate observable audio outputs. In this lab, we learned about how to redirect the signal to the DAC output channel and how to use timers and interrupts with STM32.

V. Appendix

References

[1] RM0432 Reference manual STM32L4+ Series advanced Arm®-based 32-bit MCUs.(n.d.).
Retrieved from https://www.st.com/resource/en/reference_manual/rm0432-stm32l4-series-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[2] STM32L4S5xx STM32L4S7xx STM32L4S9xx Ultra-low-power Arm Cortex-M4 32-bit MCU+FPU, 150DMIPS, up to 2MB Flash, 640KB SRAM, LCD-TFT & MIPI DSI, AES+HASH. (n.d.).
Retrieved from https://www.farnell.com/datasheets/2602792.pdf