Implementing a Digital-to-Analog Converter (DAC) Using Pulse Width Modulation and Low-Pass Filtering on an STM32 Microcontroller

Jacques du Preez† and Georgene de Wet‡
EEE3096S **Group 36**
University of Cape Town
South Africa
†DPRARE001 ‡DWTGEO002

***Abstract*—In this practical, a 10-bit Digital-to-Analog Converter (DAC) using Pulse Width Modulation (PWM) and a low-pass filter was implemented on an STM32 microcontroller. The objective being to generate analog signals by cycling through Lookup Tables (LUTs). These waveforms were observed using an oscilloscope and pushbutton interrupts were implemented to switch between waveforms, growing our practical understanding of embedded systems and signal processing.**

I. Introduction

This project focuses on Digital-to-Analogue converters (DACs). DACs are essential in various applications including audio systems and signal processing. Thus DACs play a crucial role in converting digital signals into analog forms, in this practical this is done via the use of Pulse Width Modulation (PWM) and a low-pass filter.

The main objective of this practical was to implement a DAC. Using the STM32 microcontroller we implementad a DAC by generating analog waveforms through PWM. We utilized Lookup Tables(LUTs) to create sinusoidal, sawtooth, and triangular waveforms and used a DMA to transfer these LUT values to the PWM register. Further, we implemented an interrupt mechanism using the pushbutton on the stm to switch between the generated waveforms.

The source code for this project can be found on GitHub: https://github.com/georgene01/EmbeddedSystems$_{L}ab2_{G}roup36.git$. [1]

II. Methodology

A. Hardware Setup

To implement the Digital-to-Analog Converter (DAC) using Pulse Width Modulation (PWM), the following hardware components were used:

- **STM32 Development Board**: The core processing unit for generating the PWM signals and controlling the DAC output.
- **Breadboard**: Used for connecting the low-pass filter circuit.
- **Oscilloscope**: To observe the output waveforms from the DAC for analysis.
- **Signal Generator**: Used for testing the low-pass filter.
- **Low-Pass Filter Components**: A single Resistor and capacitor were used to build the low-pass filter, which aimed to smooth the PWM outputs' analog signal.

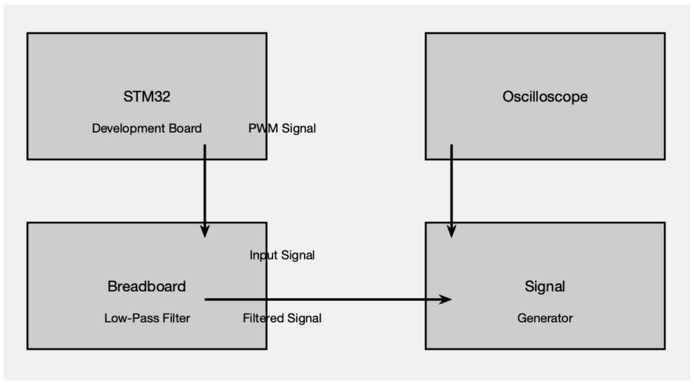


Fig. 1: Hardware Setup for DAC Implementation Using STM32 Microcontroller [2]

The STM32 development board was configured to output PWM signals on pin PB0, which was then fed into the low-pass filter circuit on the breadboard. The filtered signal was monitored using an oscilloscope to verify the waveform generation.

B. Software Implementation

1) LUT Generation: Three Lookup Tables were generated to represent the following analog waveforms: sinusoidal, sawtooth, and triangular. Each LUT contained 128 values, starting at 0 up to 1023, these corresponding to the 10-bit resolution of the DAC. These LUTs were created directly in the `main.c` file.

Listing 1: LUT Generation in main.c

```
void generate_LUTs(void) {
    for (int i = 0; i < NS; i++) {
        // Sinusoidal LUT
        Sine_LUT[i] = (uint32_t)(512 + 511 * sin(2 * M_PI * i / NS));

        // Sawtooth LUT
        saw_LUT[i] = (uint32_t)(1023 * i / NS);

        // Triangle LUT
        if (i < NS/2) {
            triangle_LUT[i] = (uint32_t)(1023 * (2.0 * i / NS));
        } else {
            triangle_LUT[i] = (uint32_t)(1023 * (2.0 - 2.0 * i / NS));
        }
    }
}
```

2) Timer Configuration: Two timers, TIM2 and TIM3, were configured to handle the PWM generation and LUT cycling:

- **TIM2**: Configured as an Output Compare (OC) timer. This timer was responsible for cycling through the LUT values at a rate determined by the frequency of the desired analog signal.
- **TIM3**: Configured in PWM mode on Channel 3 (corresponding to pin PB0). This timer controlled the duty cycle of the PWM signal based on the LUT values.

The number of timer ticks (`TIM2_Ticks`) required for each LUT update was calculated using the following formula:

$$TIM2\_Ticks = \frac{TIM2CLK}{NS \times F_{SIGNAL}} \tag{1}$$

Listing 2: Calculation of TIM2 Ticks in main.c

```
void calculate_TIM2_Ticks(void) {
    // Calculate the number of clock cycles for each LUT update
    TIM2_Ticks = TIM2CLK / (NS * F_SIGNAL);
}
```

3) DMA and Interrupts: Direct Memory Access (DMA) was used to efficiently transfer LUT values to the PWM duty cycle register (`CCR3`). The DMA was configured in Interrupt mode to allow continuous updates of the PWM output without CPU intervention.

Listing 3: DMA Initialization in main.c

```
void start_DMA_IT(void) {
    // Start DMA in Interrupt mode, using the Sine LUT as the initial source
    HAL_DMA_Start_IT(&hdma_tim3_ch3, (uint32_t)Sine_LUT, (uint32_t)&htim3.Instance->CCR3, NS);

    // Enable DMA transfer
    __HAL_TIM_ENABLE_DMA(&htim3, TIM_DMA_CC3);
}
```

The pushbutton on PA0 was used to cycle through the waveforms. An interrupt was configured to switch the waveform upon a button press, with debouncing implemented using the `HAL_GetTick` function.

Listing 4: Pushbutton Interrupt Handler in main.c

```
void EXTI0_1_IRQHandler(void)
{
    static uint32_t last_press_time = 0;
    uint32_t current_time = HAL_GetTick();

    if (GPIO_Pin == GPIO_PIN_0) {  // PA0 pushbutton
        // Implement debouncing
        if (current_time - last_press_time > 200) {  // 200 ms debounce time
            static uint8_t waveform_select = 0;
            waveform_select = (waveform_select + 1) % 3;

            // Stop the DMA transfer
            HAL_DMA_Abort_IT(&hdma_tim3_ch3);

            // Switch to the next waveform LUT
            switch (waveform_select) {
                case 0:
                    HAL_DMA_Start_IT(&hdma_tim3_ch3, (uint32_t)Sine_LUT, (uint32_t)&htim3.Instance->CCR3, NS);
                    break;
                case 1:
                    HAL_DMA_Start_IT(&hdma_tim3_ch3, (uint32_t)Sawtooth_LUT, (uint32_t)&htim3.Instance->CCR3, NS);
                    break;
                case 2:
                    HAL_DMA_Start_IT(&hdma_tim3_ch3, (uint32_t)Triangle_LUT, (uint32_t)&htim3.Instance->CCR3, NS);
                    break;
            }

            // Re-enable DMA transfer
            __HAL_TIM_ENABLE_DMA(&htim3, TIM_DMA_CC3);

            last_press_time = current_time;
```

```
        }
    }

    HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
}
```

4) Low-Pass Filter Construction: A single pole low-pass filter was built and designed using a 16k Ohm resistor and 10uF capacitor, thus having a cutoff frequency slightly above the highest desired frequency (1KHz). Thus effectively smoothing out the PWM signal into a continuous analog waveform. The cutoff frequency was calculated in the following way:

$$f_c = \frac{1}{2\pi RC} \tag{2}$$

Where:

- R and C are the resistor and capacitor values used in the filter.

The filter was initially tested using a signal generator and oscilloscope to ensure proper attenuation of higher frequencies. Once proper operation was confirmed, the PWM output from `PB0` was fed into the filter, and the resulting analog signal was observed for each waveform.

III. Results

A. Waveform Generation

The generated waveforms were successfully outputted through the DAC via created PWM. The following observations were made using an oscilloscope:

1) Sawtooth Wave Output: The sawtooth waveform was generated as expected. However, some high-frequency noise is apparent in the signal, likely due to the limitations of the low-pass filter's cutoff frequency and roll off due to being a single pole filter as well as the resolution of the PWM.

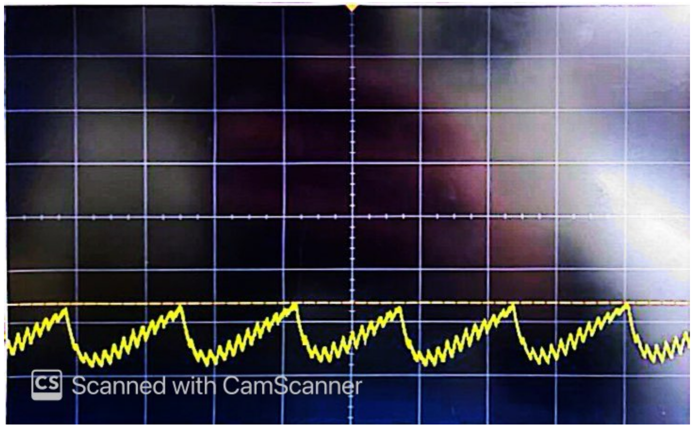


Fig. 2: Sawtooth Wave Output observed on the oscilloscope

2) Sine Wave Output: The sine wave was outputted with a notable smooth curve. Despite the crude nature of the DAC, the low-pass filter was able to produce a reasonably accurate sine wave, though some distortion is visible at the sine waves' higher frequencies.

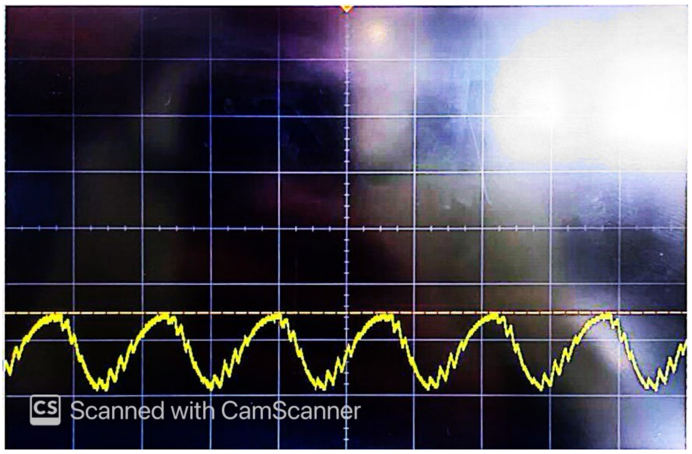


Fig. 3: Sine Wave Output observed on the oscilloscope

3) Triangular Wave Output: The triangular waveform displayed sharp transitions between the rising and falling edges, consistent with the LUT data. As with the other waveforms, there is a slight deviation from the ideal due to filter imperfections.

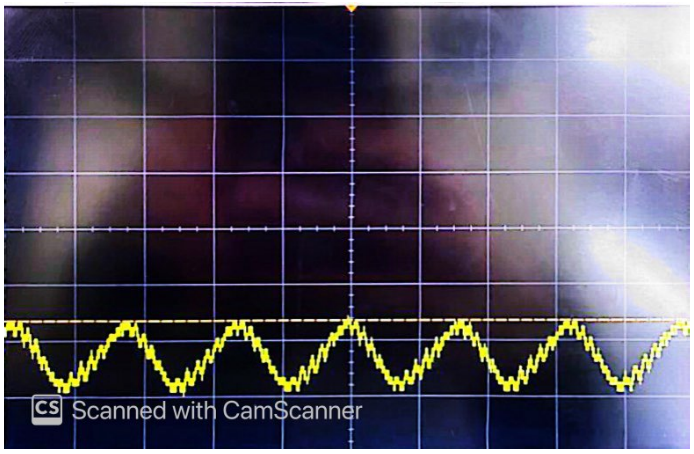


Fig. 4: Triangular Wave Output observed on the oscilloscope

The LUTs used to generate these waveforms were plotted using MATLAB and are shown in Figure 5:

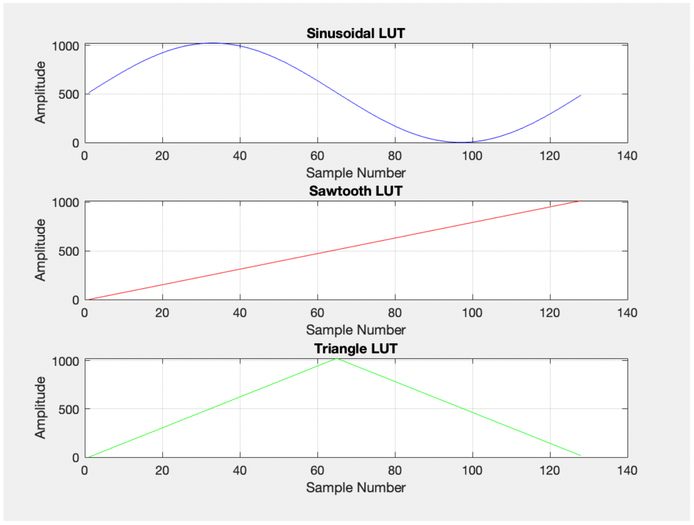


Fig. 5: Lookup Tables (LUTs) for generating the waveforms: Sinusoidal (Top), Sawtooth (Middle), and Triangular (Bottom)

B. Low-Pass Filter Response

To validate the performance of the low-pass filter used to smooth the PWM signals, a Bode plot was generated by measuring the frequency response of the filter via MatLab. The plot shows the theoretical and measured magnitude response of the filter, as shown in 6:

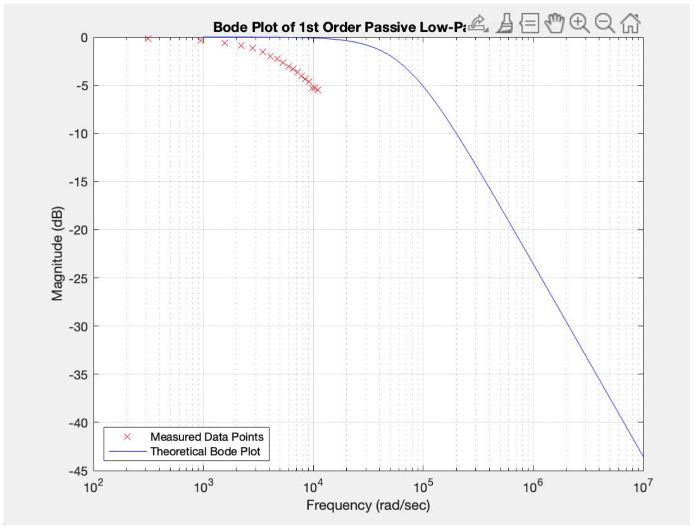


Fig. 6: Bode Plot of the Low-Pass Filter's Frequency Response

The Bode plot indicates that the low-pass filter effectively attenuates frequencies beyond its designed cutoff frequency, which is essential for producing cleaner analog signals from the PWM input, however the steepness of the rolle-off could use improvement, perhaps via the use of filter cascading.

IV. Conclusion

This practical successfully demonstrated the use of Pulse Width Modulation (PWM) in implementing a simple Digital-to-Analog Converter on the STM32 microcontroller.

Despite the limitations in resolution and filtering, the generated waveforms closely resembled the desired signals, with sine, sawtooth, and triangular waves all being effectively produced and observed on an oscilloscope, showing their distinctive forms and characteristics. The practical also highlighted the importance of low-pass filtering in smoothing PWM outputs and the trade-offs involved in achieving higher signal quality.

A. Future Improvements

The following improvements could be considered:

- **Higher Resolution PWM**: Implementing a higher resolution PWM, through dedicated DAC hardware, which could improve the fidelity of the output waveforms.
- **Filter Cascade**: Adding additional stages of low-pass filters in a cascade configuration could further reduce noise.
- **Optimized Filter Design**: Choosing more optimal resistor and capacitor values, could enhance the filtering performance to better match the desired signal characteristics.

B. Key Considerations

- **Fastest Frequency Generation**: The fastest frequency that can be generated with the STM32 development board is approximately 1 kHz. At higher frequencies, the output signal quality deteriorates due to increased noise and limited resolution of the PWM.
- **Trade-offs**: The main trade-off at higher frequencies is the increased noise due to the low-pass filter limitations and the PWM signal resolution.
- **Signal Integrity Improvements**: To improve signal integrity, additional electronics such as a cascade of low-pass filters could be added after the microcontroller.
- **Output Pin Impedance**: The impedance of the PWM output pin on the STM32 development board is approximately 50 ohms. This plays a role in determining the circuit loading effects.
- **Importance of Low-Pass Filter**: Without adequate filtering, the signal output would be dominated by high-frequency noise present in PWM switching, making the analog signal unusable for most applications.
- **Driving a Pure Sine Wave Inverter**: The principles demonstrated in this practical could be extended to drive a pure sine wave inverter. By refining the waveform generation and filtering processes. Thus a stable sine wave output could be produced, which is necessary for powering sensitive AC loads in inverter applications.

[3]

References

[1] U. o. C. T. Department of Electrical Engineering, “Eee3096s course handout,” Technical Report, January 2024.

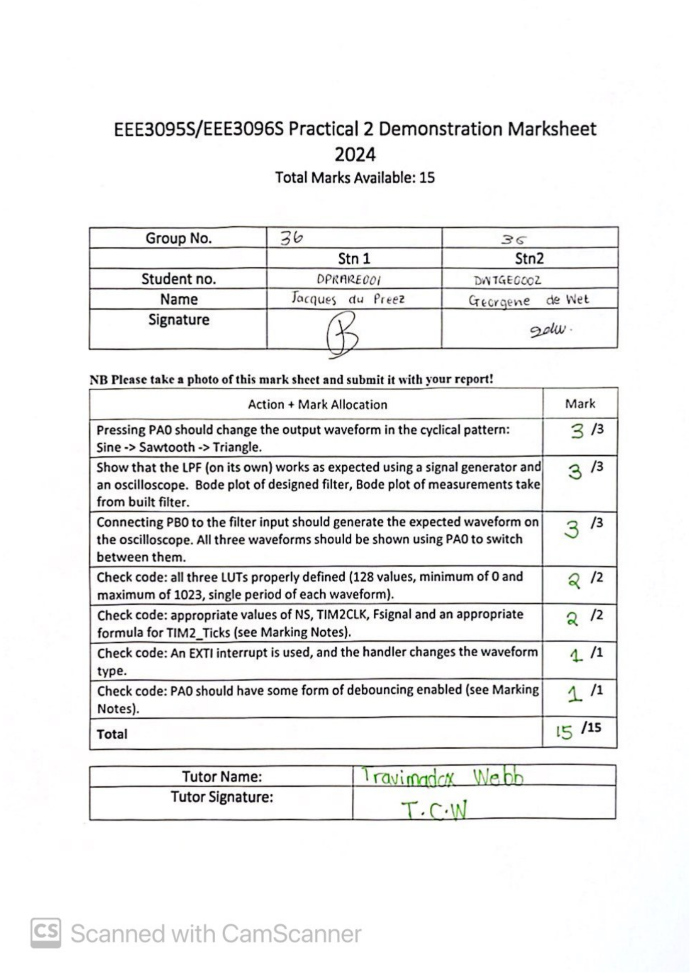
[2] MATLAB, “Matlab r2023a,” The MathWorks, Inc., Technical Report, March 2023. [Online]. Available: https://www.mathworks.com/products/matlab.html

[3] “Grammarly,” https://www.grammarly.com, accessed: 2023-08-27.

[4] “Stm32cubeide,” https://www.st.com/en/development-tools/stm32cubeide.html, accessed: 2023-08-27.

[5] STMicroelectronics, “Stm32f0xx reference manual,” STMicroelectronics, Technical Report, March 2023. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0091-stm32f0x0-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Appendix



EEE3095S/EEE3096S Practical 2 Demonstration Marksheet
2024
Total Marks Available: 15

Group No.	36	36
	Stn 1	Stn2
Student no.	DPRARE001	DWTGEO002
Name	Jacques du Preez	Georgene de Wet
Signature	[illegible]	[illegible]

NB Please take a photo of this mark sheet and submit it with your report!

Action + Mark Allocation	Mark
Pressing PA0 should change the output waveform in the cyclical pattern: Sine -> Sawtooth -> Triangle.	3 /3
Show that the LPF (on its own) works as expected using a signal generator and an oscilloscope. Bode plot of designed filter, Bode plot of measurements take from built filter.	3 /3
Connecting PB0 to the filter input should generate the expected waveform on the oscilloscope. All three waveforms should be shown using PA0 to switch between them.	3 /3
Check code: all three LUTs properly defined (128 values, minimum of 0 and maximum of 1023, single period of each waveform).	2 /2
Check code: appropriate values of NS, TIM2CLK, Fsignal and an appropriate formula for TIM2_Ticks (see Marking Notes).	2 /2
Check code: An EXTI interrupt is used, and the handler changes the waveform type.	1 /1
Check code: PA0 should have some form of debouncing enabled (see Marking Notes).	1 /1
Total	15 /15

Tutor Name:	Travimadox Webb
Tutor Signature:	T.C.W

Fig. 7: Enter Caption

This section provides the full listing of the `main.c` file used in this practical. The code implements the generation of waveforms using PWM, timer configuration, DMA handling, and interrupt-driven waveform switching. [4]

Listing 5: Full listing of `main.c`

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * ...
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include "stm32f0xx.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
// TODO: Add values for below variables
#define NS 128        // Number of samples in LUT
#define TIM2CLK  8000000  // STM Clock frequency
#define F_SIGNAL 1000 // Frequency of output analog signal
/* USER CODE END PD */
/* ... */

/* Complete code here... */
```

This section includes the waveform plots generated from the Lookup Tables (LUTs) and the corresponding outputs observed on the oscilloscope. [5]

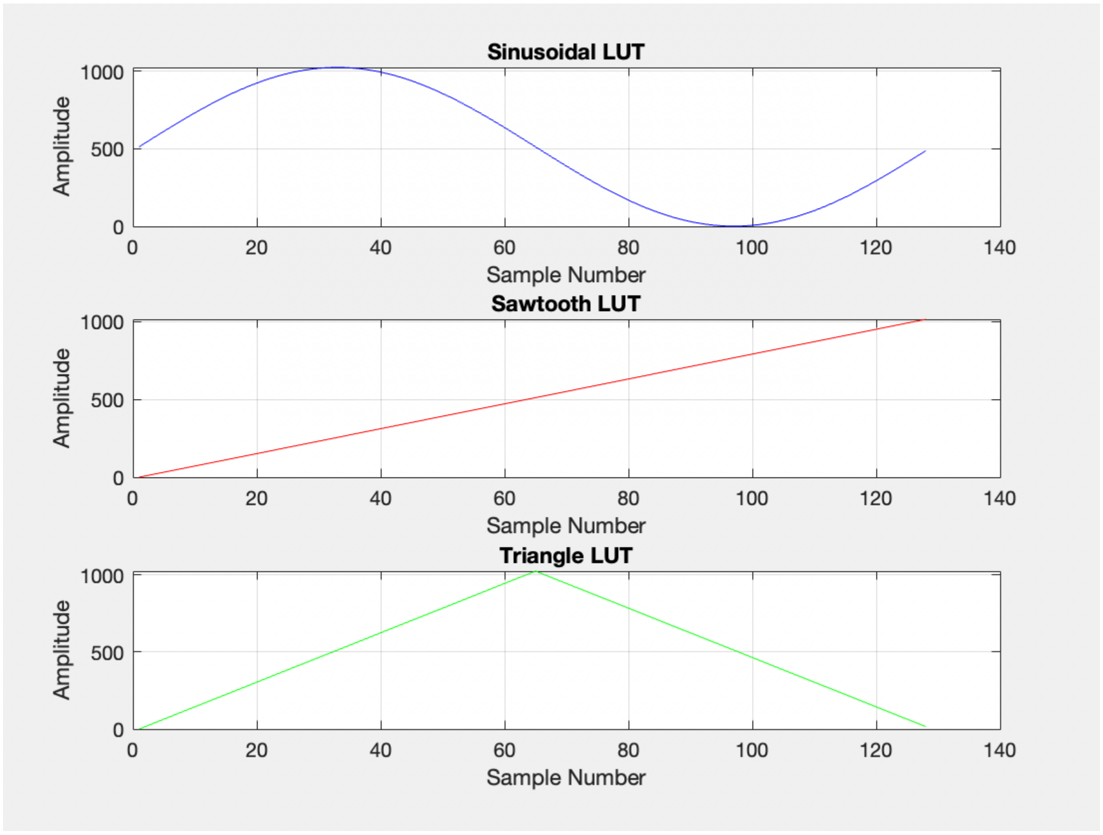


Fig. 8: Lookup Tables (LUTs) for generating the waveforms: Sinusoidal (Top), Sawtooth (Middle), and Triangular (Bottom)

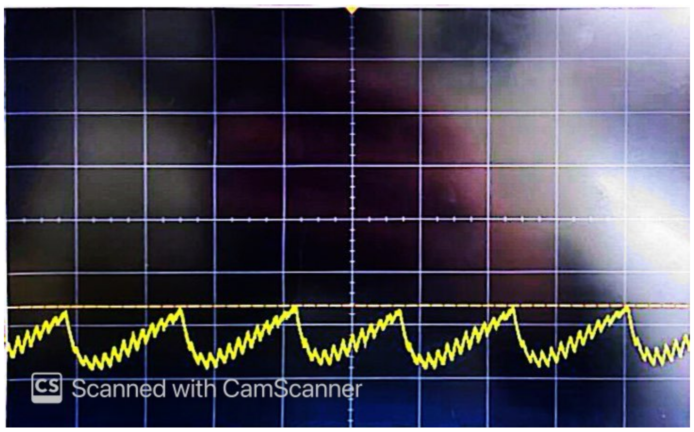


Fig. 9: Sawtooth Wave Output observed on the oscilloscope

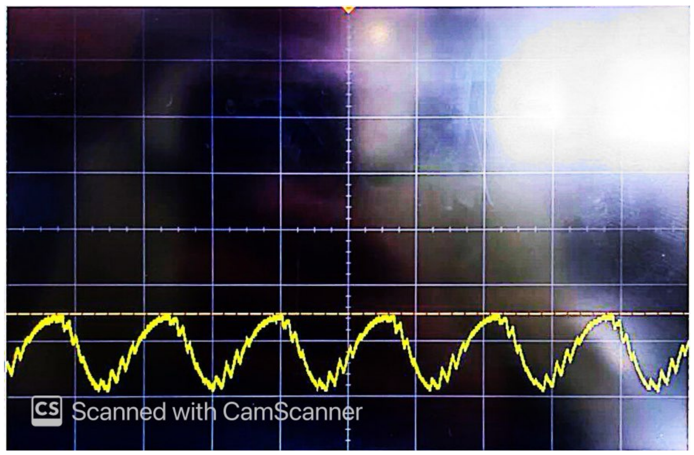


Fig. 10: Sine Wave Output observed on the oscilloscope

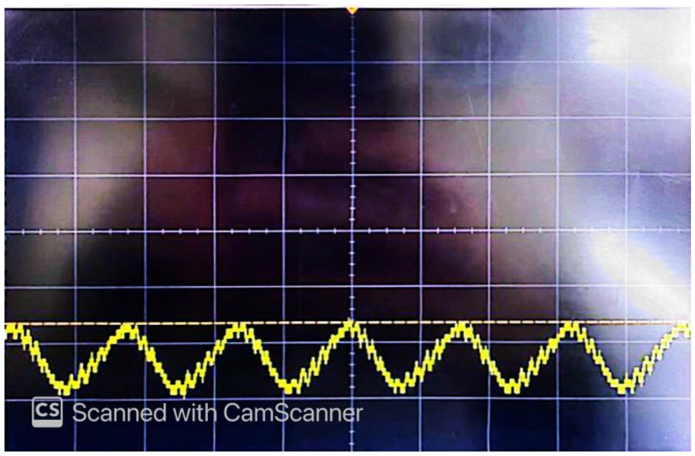


Fig. 11: Triangular Wave Output observed on the oscilloscope

If there are any additional datasets, results, or figures that are relevant to the practical, they can be included in this section.

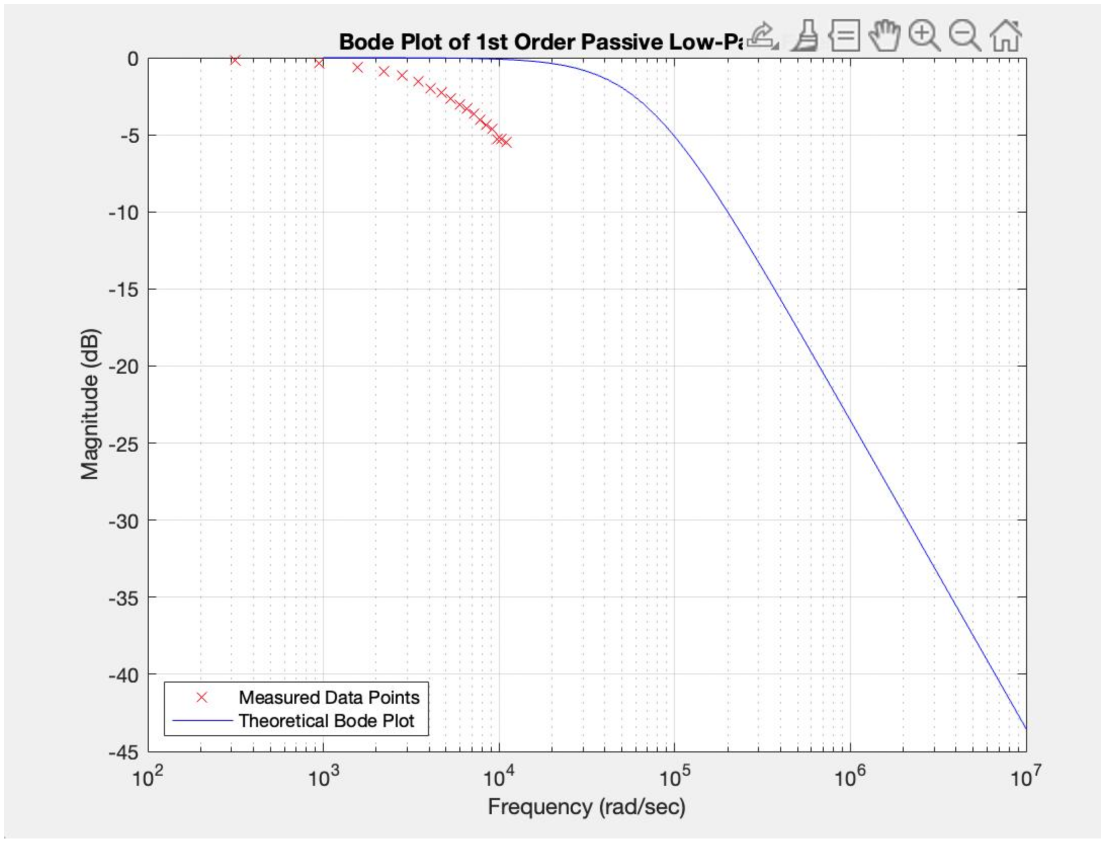


Fig. 12: Bode Plot of the Low-Pass Filter’s Frequency Response