Final Project: STM32 Signal Generator

Gizem Oznalcaci*, Kaicheng Wu†, Tian Feng‡, Ralph Choucha§, and Maya Ajji¶
Dept. of Electrical and Computer Engineering
McGill University, Montréal, Canada
* Email: gizem.oznalcaci@mail.mcgill.ca
† Email: kaicheng.wu@mail.mcgill.ca
‡ Email: tian.feng@mail.mcgill.ca
§ Email: ralph.choucha@mail.mcgill.ca
¶ Email: maya.ajji@mail.mcgill.ca

***Abstract*—The goal of the project is to design a signal generator with an STM32 board. Rather than clicking on button to select signal type, UART will take the signal type input whereas the frequencies and the amplitudes will be adjusted by the knobs-potentiometers. The signal will be generated by using DSP and the noise will be suppressed through the Kalman filter. The final output will be shown on the oscilloscope through DAC pins. [EXPLAIN RESULTS]**

***Index Terms*—embedded systems, C programming, UART, STM32, DAC, ADC, oscilloscopes, interrupt, DSP, Kalman filter**

I. Introduction

The system mimics the signal generator by allowing users to observe different signal types with various frequency and amplitude levels. To achieve these goals, the program will take input from the user through UART to select the signal type whereas the amplitude and frequency will be selected by the user through 2 ADC channels and 2 potentiometers. If the user wants, he can also combine a new signal type with the current signal type for output. Then, the system will use these inputs, DSP, and Kalman filtering to generate signals that will be displayed on the oscilloscope through the DAC channel. It will also keep sending the current signal's signal type, amplitude, and frequency until the user quits the system or wants to start over.

The frequency range is 1-10 kHz and the amplitude range is 0-2.5V. The wave parameters are controllable through a UART interface and the potentiometers.

II. Implementation

A. Taking Input From The User: UART, ADC & Potentiometer

1) UART: The message and waveform refreshes are driven by the user-button interrupts. After the successful loading of the program, the user will be prompted with a question asking a 5-letter password(default 12345). Then, it will ask the user to select the signal type by selecting a number from 0-3 which represents the signal types sine wave, triangle wave, sawtooth wave, and square wave respectively as shown in Fig. **??**.

After the signal type has been set, the UART will display the signal that should be generated in DAC each time the user button is pressed.

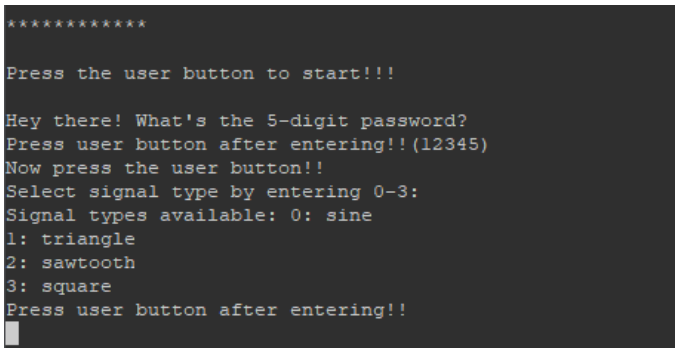

Fig. 1. User input is taken from UART

2) ADC with Potentiometer: To obtain frequency and amplitude values, the program will utilize input coming from potentiometers through the ADCs. In order to measure the correct voltage levels coming from ADC, the internal voltage reference will be polled at the beginning and the system values will be normalized for the conditions of the MCU. **PC4 will control the frequency and PC5 controls the amplitude.** The ADC values had been set to have an amplitude level range from 0V to 3.3V and the frequency to range from 0Hz to 1MHz. These values will be continuously polled when the signal type is set and they will be displayed until the program is terminated.

B. Creating The Wavetable: DSP

The program will be able to output different types of waves that the user can select from. This includes a sine wave, a square wave, a sawtooth wave, and a triangle wave. A separate C file provides functions for each type of wave that take in an array pointer for the buffer array that will hold the values calculated by the function, as well as the number of samples per period that will be taken.

The sine wave is calculated by dividing 2π by the number of samples inside the CMSIS sine function in order to get the step-in values when multiplied by the number of iterations. The sine wave is shifted up by 2048 to center it in the middle of the DAC's range and the remaining 2048 units are multiplied by the sine wave in order to vary the range of the sine signal over the full DAC's resolution.

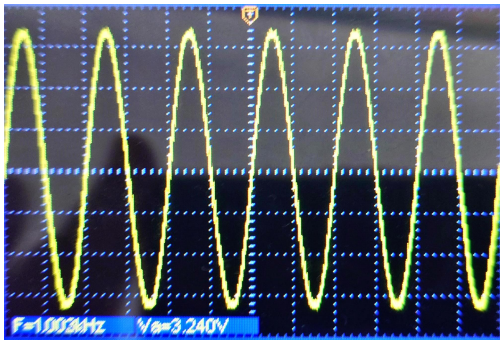

Fig. 2. Sine 3.25Vamp 1kHz

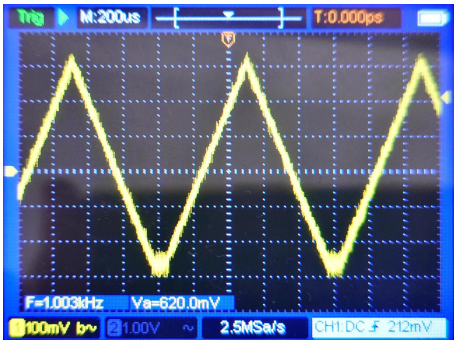

Fig. 5. Triangle 1kHz 600mVamp

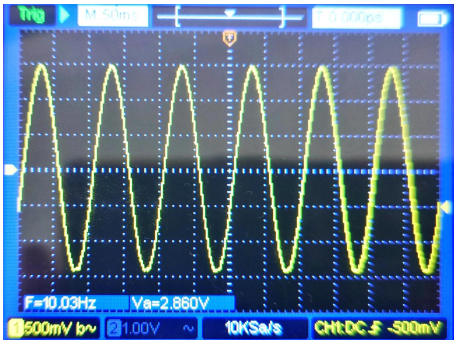

Fig. 3. Sine 10Hz 2.9Vamp

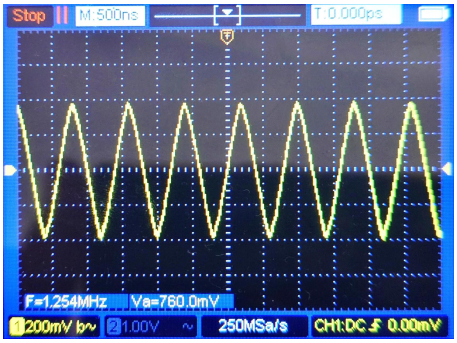

Fig. 6. Triangle 1Mhz

The triangle wave is calculated by first dividing the period into 4 parts. In the first quarter, the wave goes from the center to the top of the DAC's range which is why the values are shifted by 2048 and are made to vary linearly upwards. The next part of the triangle wave is the decreasing part in the second and third quarters. A normalized position is first calculated by doing *(i - temp) / samples* which is then multiplied by 2 times the max resolution of the DAC. This should give the relative decrease for iteration of the for loop and is subtracted from 4096 to calculate the final value. The last quarter is where the wave varies upwards again from 0 which is where we take the values from the first quarter and shift them down by 2048.

The saw wave is calculated by dividing the period into 2 halves. The first half varies from the middle of the range to the top, and the second half has values that vary from 0 to the half. The first half is made to increase linearly from 2048 and the second half takes the values of the first half and shifts them down by 2048.

Finally, the square wave is also divided into 2 halves. The first half of the values are set to the max value of the DAC's range while the second half is set to 0.

C. Noise Suppression: Kalman Filter

In order to filter out unwanted noise and uncertainties from the potentiometer input, we will be using two Kalman filters; one for frequency and one for amplitude.

We define a Kalman filter as a C language struct with the following parameters: q (representing the process noise

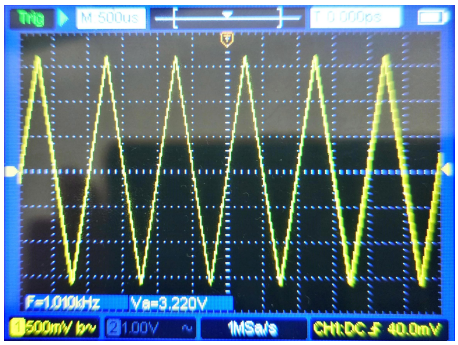

Fig. 4. Triangle 1kHz 3.3Vamp

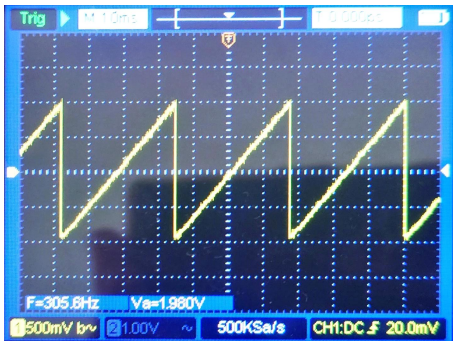

Fig. 7. Sawtooth 300Hz 2.0Vamp

Fig. 8. Square 1kHz 3.3Vamp

variance), p (representing the estimation error covariance), r (representing the measurement noise variance), k (representing the Kalman gain), and x (representing the input value).

With each iteration, we tweak these settings to make the filter work better. So, p gets updated by adding in the process noise ($p+q$), while k changes depending on how much noise there is and how reliable our measurements are ($\frac{p}{p+r}$). Also, the input value x gets adjusted by mixing the current value with how far off our measurement is from our current guess ($x + k \cdot (measurement - x)$). Lastly, we adjust the error estimate p to match our refined guesses ($(1-k) \cdot p$). This loop keeps going, making sure our Kalman filter keeps a good balance between all the noise and uncertainties, giving us solid estimates.

D. Outputting the Signal: DAC with DMA

DAC will be used to create an output according to the input taken through UART and ADCs. The DAC works in 12-bit DMA mode and the generated waveforms using CMSIS will be transferred to the DAC bypassing the CPU with DMA. The output examples are included in the waveform section.

For a higher frequency range(10kHz or more) the DAC will work in 8-bit mode instead of 12-bit to improve high-frequency response. The formatting is implemented in line 404.

E. Auto TIM Clock Speed

If the device is outputting a 10Hz wave and the DAC is working at 10Mhz, then a wavetable will need 1MB which is too heavy for the chip. So the TIM clock for the DAC's DMA channel will auto-adjust each refreshment to cap the sample number at 1000. This part is implemented in the line 153 of the code.

III. Limitations and Further Improvements

Although we have been able to complete the project defined in our proposal, several aspects could have been improved. For example, the current implementation of the application needs to be rebooted for each signal generation operating at a higher frequency due to limited DMA speed. We could improve it by properly deallocating arrays of values that are no longer required.

Another thing is the calibration and calibration data saving. Calibration is recommended since there's some slight difference between each board and the potential meter used. We could use the on-board flash to save the last calibration data for the same measuring circumstances.

The third thing is arbitrary signal generation. Instead of generating the fixed signal, we can let the user specify the signal parameters in each period. This improvement is not hard but due to the time limit, this is not implemented.

The final point is that the built-in DAC has limited speed and the signal will distort at higher frequencies. An individual DAC and Amp circuit could be added to improve high-frequency response and THD.

IV. Conclusion

To conclude, in this lab project we built an STM32-based device that is able to generate sine waves, triangle waves, sawtooth waves, and square waves with frequencies between 1-1MHz and amplitude between 0-3.3 V. This project allowed us to integrate everything we learned from the course ECSE444 into one single functioning application.

V. Milestones, Timelines & Member(s) Responsible

1) Getting input from the user through UART: 04/02/2024 - Gizem

2) Getting input from the potentiometer through ADCs: 04/02/2024 - Gizem

3) Creating an output signal through DAC according to the input taken from UART: 04/04/2024 - Ralph

4) Manipulating the output signal amplitude according to the input taken from ADC: 04/04/2024 - Maya

5) Creating an output signal through DAC according to the input taken from UART and ADC: 04/05/2024 - Tian

6) Using DSP to output more signals: 04/08/2024 - Tian

7) Using Kalman Filter for signal input: 04/08/2024 - Kaicheng

8) Integrating all of the functions together: 04/09/2024 - Tian, Maya, Ralph, Kaicheng

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

[3] ECSE444 Lab4 documents
Retrieved from https://mycourses2.mcgill.ca/