Real-Time and Embedded Systems

Project 05: Signal Generator

Author: Liam Sullivan
Email: lds7381@rit.edu
Submission Date: 11/17/2022

Analysis and Design

For this project, a signal generator that would be able to produce 4 different types of signals with varying characteristics was created using the STM32 microcontroller. These signals were analog signals that were outputted by the Digital to Analog Converter (DAC) on the STM32. The four types of signals that the STM32 microcontroller needed to generate were Square, Triangle, Sine, and EKG waves. These signals would be produced after a user would input the desired wave through a terminal over UART to the microcontroller. The input from the user would determine the characteristics of the signal that the STM32 would produce. The user would input the desired channel for the DAC to output on, the type of signal, the frequency of the signal, the minimum and maximum voltages of the signal, and the amount of noise to be applied to the signal. The user would enter these values into the terminal and the DAC on the STM32 would output the desired signal. However, some of these values could only be in ranges so as to not go outside the functionality of the DAC. The DAC was able to have a maximum voltage output of 3.3V and the maximum voltage of the desired signal could not exceed this. Also up to only 12 bits of noise was allowed due to the DAC values being a maximum of 12 bits long. When these ranges were followed the STM32 was able to successfully act as a signal generator and produce a desired analog output.

The first step in setting up this project was to get the DAC initialized and set up to produce an analog output. To do this the DAC was enabled to output on channels one or two. Next the output buffer was enabled and the trigger was set to Timer 2 Trigger out event. Timer 2 was set up to work with the DAC. The timer had no pre scalars and had its trigger event set to Update event. When the DAC worked with the Timer, this meant that when timer two had reached its ARR value the DAC would output the next value in its output buffer. However to get the information from this buffer, Direct Memory Access (DMA) had to be set up to pull a Word of information out of the buffer (memory) and directly put it into the DAC peripheral. This would happen every time the Timer 2 would trigger (reach ARR). This fully set up the DAC and allowed it to be fed a buffer of DAC voltage output values in a range of 0 - 4095 which corresponded to 0V - 3.3V to be set to the DAC for signal generation.

Using Timer 2 also allowed for the change of the frequency easily. The frequency could be set by changing the timer 2's ARR register's value to one that would match the desired frequency of the user for the desired waveform. The equation for calculating the ARR register's value for the desired frequency can be found below in Equation 1.

TIM2->ARR = (BASE_CLOCK_SPEED/frequency)/WAVE_POINTS (1)

BASE_CLOCK_SPEED = 80MHz

WAVE_POINTS = 256

Using the equation above any desired frequency for a signal to be outputted for the STM32 under 100KHz would be feasible. The STM32 cannot output past the maximum frequency of 100KHz.

To get the output buffer for the DAC and DMA, four separate functions to generate the points in regards to the signal type, min and max voltages, and noise bits were created. These functions would generate an output buffer for Rectangle, Triangle, Sine, and EKG waves. All of these functions created an output buffer that had values between the desired voltage ranges and had a total of 256 total DAC output buffer points.

To create the output buffer that would produce a Rectangle wave, the first half of points was set to the maximum voltage value and the second half set to the minimum voltage value. This generated an output buffer that would produce a wave that was a Rectangle wave that follows the maximum and minimum voltage ranges.

Next, to create the triangle wave for the output buffer half of the points had to be sloping up to the maximum voltage then the second half sloping down to the minimum voltage. These slopes had to be linear to give the look of a triangle. To do this an increment/decrement was found to create these slopes between the given voltage values. The equation to find this increment/decrement can be seen below in Equation 3.

increment = (Max Voltage - Min Voltage) / ((WAVE_POINTS - 1)/2) (3)

From this increment the linear slope of the triangle wave was able to be calculated for both increasing and decreasing slopes. By starting using the maximum and minimum values for the voltages in determining the increment, all possible values in the output buffer will be in between the voltage range. This allowed the output buffer to have the correct values for the requested triangle wave.

Another signal that needed to be generated was the Sine wave. This was done by using the ST Resources Waveform Generation documentation that had an equation on how to calculate the necessary points that would be required for the DAC to output a smooth Sine wave. This equation to find all of the values necessary to create a Sine wave can be seen in Equation 4.

wave_points[i] = ((sin((i * ((2.0*PI)/WAVE_POINTS))) + 1.0) * (((Max Voltage)/2.0)) (4)

i = the index of the value in the output buffer

This equation was able to successfully produce 256 separate points along a Sine wave that the DAC can output. Using this equation made it easy to scale the voltage ranges of the desired signal as well. This allowed the DAC to produce the desired output analog signal.

The last type of signal that was able to be outputted by the DAC was an EKG signal. This EKG signal's DAC values were given. These values were then scaled by the min and max voltage values to fit in the desired voltage range. Since these values did not have to be calculated, the output buffer was just set to them.

To add noise to the waveforms that were being generated, the number of bits of noise was given by the user during input. This number reflected the amount of bits on the value of the output buffer that would be random. If 3 bits of noise were applied, the least significant 3 bits would be XOR'd with all 1's to create randomness on these numbers. This made the outputting waveform be more "staticy" and have more variation instead of being just the straight waveform which would be expected. This allowed the microcontroller to simulate what noise could look like when a signal is possibly being read.

For the architecture of this program, FreeRTOS was not used and only 2 functions were called in the main.c. The first of these functions being the user input function. This function used string.h to split the user input by a delimiter of " ". This allowed for the program to get each individually inputted characteristic of the wave. These characteristics were saved and then sent to the next function. This second function was to set up the Timer 2 to the correct frequency and to then select which waveform will be outputted. From this the function is added to the output buffer depending on the min and max voltages, signal type, and amount of noise. This function then calls the DAC and DMA to start taking values out of the output buffer. The DAC then changes its output depending on the frequency, this makes the desired output signal that the user asks for. The function ends and no new functions or statements are called and the STM will continue to output the signal.

Test Plan

The first test that was performed was on the user input. The user input for this project was a little different than how it has been in previous projects. That is because the program required 6 different inputs for the waveform's specifications. Making sure that the user input was being properly handled and stored was the biggest test. This test was performed on many different implementations to get the user's input. This ultimately led to the use of string.h with the

strok() function to split the user's input. This allowed for the user's input to be read and stored quite easily. Without testing this first, the old user input implementation would still be in use and wrong values could have been gotten which would have led to incorrect waveforms.

The biggest test that was performed was by reading the output waveform from the microcontroller. This test was done with the use of an oscilloscope. The desired outputting waveforms had known characteristics that had to be seen on the oscilloscope. By using the oscilloscope the analog waveforms could be seen in real time and allowed for easy debugging of what could possibly be going wrong with the waveforms. Just testing waveforms on the oscilloscope leads to many errors in the waveforms being seen. Such as when the triangle was first being implemented, it looked like a 'M' not a triangle. I was able to quickly find where the error occurred in the program and fix it to make the waveforms be correct. Testing with the oscilloscope made this project possible and it could not have been performed without it.

Project Results

The results of the project is a program that can produce any of the four desired waveforms that the user could specify. During testing and demonstration of the project, the program was able to produce all types of necessary signals at a frequency of 0.5 - 100,000 Hz while being in the desired voltage ranges and having noise. This showed that the project to create a signal generator with the required specification was correct.

The only way to know that this project was successful was with the use of an oscilloscope. The oscilloscope allowed the waves being outputted by the STM to be seen and analyzed. This was the only verification method for this project. Using the oscilloscope it was able to be seen that the waveforms were being created as expected from the user input. All different types of waves were tested on this oscilloscope and all looked like expected except for the Sine wave. The Sine wave on the oscilloscope had a part where it would flatten out at the bottom of the wave. This was not supposed to happen. It is unsure what caused this to be there but it could have been a small error in the way that Equation 4 was being used. While the waveform was slightly off it still looked and had the characteristics of a Sine wave. From being able to see all these wave and how they were performing the project was deemed a success and the STM32 was able to produce any desired waveform within its specifications.

Lessons Learned

One of the biggest lessons learned was with the use of the oscilloscope. Prior to this project I have used the oscilloscope a decent amount. However, I still didn't know what certain aspects of the oscilloscope were. One of these aspects was the trigger. Prior to this project, I just avoided the trigger because I didn't really know what it was. I now know that the oscilloscope

uses it as a way to read the waveforms more correctly by “triggering” when the waveform crosses that voltage. This also allowed for more accurate reading of the waveform and ultimately easier analyzing.

Another lesson learned would be about the noise bits on the waveform values. For this project I used XOR to make the amount of desired bits random. However, this could be better done using the RNG register instead. Moving forward when needing to produce signals if noise is needed a RNG peripheral will be used in the future. This would be done by shifting out the amount of bits specified from the value then shifting in bits from the RNG register. This would lead to a more accurate depiction of the noise on a waveform going forward.