|  |
| --- |
| The University of York |
| Electronics Department |
| Design & Construction |
| Final Report |
|  |
| **Jonathan Caine**  **Sam Beedell**  **Alexander Brown**  **Liyu Ding** |
| **06/05/2014** |

# Group Summary of Project

The group feels that overall the project was a success, as we managed to meet the vast majority of the level 2 and level 3 requirements outlined in our initial specification. We managed to exceed them in some places, such as with the maximum achievable frequency for sine and square wave generation. In places where we were unable to achieve the desired specification such as with amplitude and frequency modulation, we replaced them with alternative functionality for example frequency shift keying modulation, or added additional functionality such as an arbitrary function generator.

The software team managed to generate both sine and square waves using the DDS on the AD9850 module, which exceeded the 1MHz requirement outlined in the initial specification. This is because the DDS chip itself has an upper limit for the generation of waves at 40MHz, although they become extremely distorted above roughly 5MHz. As a software team we choose to limit the output frequency of the DDS chip to 35MHz to protect the chip itself, and to leave the decision up to the user of the system whether they would like to use the waveforms at high frequencies, even though the distortion becomes very significant the higher the frequency selected.

The software team also successfully managed to create a random noise generator, however it did not use the internal PR sequence generator. Instead while implementing the triangle wave generation using the DAC, we noticed in the data sheet [1] that the DAC has the ability to generate small amplitude noise signals that can be added to another waveform. We based our code around the example code from the peripheral examples by the ST Micro Electronics application team, provided with the CMSIS peripheral library. This resulted in a nice random noise signal that could be easily turned on or off, by enabling or disabling the appropriate channel of one of the DACs.

## Full System Review

After combining the software and hardware sections of the project, the overall product was pretty successful at its intended functionality. It provided sine and square wave generation from approximately 0.03Hz up to 35MHz, with the ability to adjust the frequencies “on the fly”, and triangle wave generation again from approximately 0.03Hz up to about 1MHz. An accurate frequency meter across a range of frequencies from 0.1Hz up to 10MHz and above, although the accuracy is noticeably reduced above 10MHz, and which displays the frequency and duty cycle on the LCD screen. Also a predefined “arbitrary” waveform in the form of a sinc function, a small amplitude noise generator, a pulse generator with the ability to vary the duty cycle and frequency range, and frequency shift key modulation (FSK) for a selected range of carrier frequencies.

Any input signals to the product were prevented from causing damage to either the STM32F4 board or the carrier board, by passing them through a crossing detector circuit, which shifted the amplitude of the signal to be between 0 and 3 volts, and provided an equivalent frequency square wave to input to the carrier board. This meant that theoretically any signal could be able to be used as an input to the product, including sine waves and random or arbitrary waveforms. The product also has the ability to adjust certain parameters of output signals, such as the frequency and duty cycle from the software implementation, and the voltage range and amplitude from the hardware implementation. The carrier board can only output signals in the 0 to 3 volt range, but by passing these through an amplification circuit with a potentiometer to vary the amplitude, followed by a level shifting circuit, the output waveforms can be varied from -12 to +12 volts.

The group feels that the “arbitrary” waveform could have done with more work, so that ideally the user could have downloaded any waveform into the system from a connected computer running MatLab. MatLab was used to generate an array to represent the desired waveform to the precision required by the DAC. If the software team had had extra time, they would have liked to implement a serial over USB protocol, also known as a Communications Device Class (CDC), to be able to use a console window on the connected PC, to send data to the product. This would have allowed the user to define a true arbitrary waveform of their choosing in MatLab, and then “send” it to the board ready to be output.

Also the user interface that was available on the board for the demonstration afternoon, was not very user friendly and appeared buggy. This is mainly due to the lack of hardware de-bouncing provided on the STM32F4 board for the blue user button. This will be expanded on in the “Problems Encountered” section, however the group would really have liked to have had a much more suitable user interface for the product.

Overall the product was a great success, but the group would have liked to have more time available to further enhance the product, and develop it further to a full prototype stage rather than leaving it in mid-development.

## Problems Encountered and still Outstanding

The most noticeable problem that the group encountered was the poor user interface available on the board during the demonstration afternoon. In conjunction with the hardware team, the software team had ordered a 1-pole 12-output rotary switch, which was intended to be used as an alternate method to select the desired functionality of the product. This is because using the blue user button the STM32F4 board resulted in it being very difficult to select the desired functionality, as there was no hardware de-bouncing associated with the input. The software team tried to implement a form of software de-bouncing, by not clearing the interrupt pending flag until the very end of the IRQ handler routine, which means that a rapid second press or prolonged press of the blue user button, cannot generate another interrupt until the IRQ handler routine has completely finished executing first time around. However as the M4 processor on the STM32F4 board executes instructions so quickly, and the IRQ handler routine for the blue user button was so short, it finished executing the routine so quickly that a second or third interrupt was able to be generated immediately after the first one had finished executing, resulting in rather than stepping through the different product functionalities one at a time, 2 or 3 would be jumped through on each button press. The solution to this was to use the rotary switch and write an IRQ Handler for each output of the switch, eg one for each different functionality provided by the product. Therefore when the rotary switch was turned to a desired output, the desired output would always be set correctly, as each IRQ handler routine would only set 1 functionality rather than all of them like using the blue user button required. Unfortunately the company the group ordered the rotary switch from made a mistake, and shipped some accessories for the rotary switch such as the cap and a plastic ring to identify the switch positions, but failed to ship the actual switch. This meant that the switch finally arrived about 30 minutes before the demonstration afternoon was due to start, and although the software team had tried to write the appropriate IRQ handler routines and configure the EXTI lines appropriately, when we tried connecting the switch it didn’t work as expected, and we did not have time to trouble shoot the problems before the demonstration afternoon was due to start. We therefore reverted to the blue user button as a user interface for the demonstration afternoon, as at least it worked, and have since got the rotary switch working as desired.

* Delay() function problems
* IRQ handler problems
* Any overall hardware problems

# Software Team Design/Implementation Report

## Wave Generation Using DDS

## Frequency Meter

## Triangle Wave Generation

## Noise Generation

## Arbitrary Function Generation

## Pulse Generator

## Frequency Shift Keying Modulation

# Hardware Team Design/Implementation Report

# Software Team Appendixes

# Hardware Team Appendixes

# References