RTDSP Lab 3 Report

# Section 1 – Lab notes questions

##### **Why is the full rectified waveform centred around 0V and not always above 0V as you may have been expecting?**

**Capacitive coupling – capacitor removes DC offset**

##### **Note that the output waveform will only be a full-wave rectified version of the input if the input from the signal generator is below a certain frequency. Why is this? You may wish to explain your answer using frequency spectra diagrams. What kind of output do you see when you put in a sine wave at around 3.8 kHz? Can you explain what is going on?**

**Rectified signal has double frequency of input so input must be half the Nyquist limit to keep output under Nyquist limit.**

# Section 2 – Code operation

##### **Exercise 1**

Figure 1 is a snippet of the interrupt service routine (ISR) that is triggered every time a sample is received. Its function is very simple: read the sample using the mono\_read\_16Bit()function, take its absolute value with fabs()and then write it to the output again using mono\_write\_16Bit().

We use fabs() instead of abs() because the latter returns an int, which means we would only ever get a result of 0 or 1. But you can’t store floats in 16 bit values - ??

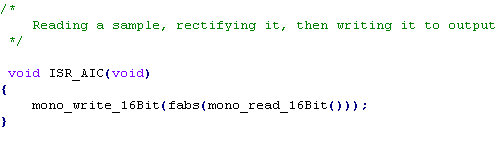
****

Figure - Interrupt Service Routine for Exercise 1

##### **Exercise 2**

The ISR for exercise 2 is more complex; firstly, note that we make use of the sine\_init() and sinegen() functions from the previous lab. You will recall that the values stored in the lookup table are 32-bit floats, but our mono\_write\_16Bit()function takes a 16-bit parameter. Therefore, we need to typecast our rectified value to a short. However, the lookup table value will always be between -1 and 1, so casting to a short would only ever return -1, 1, or 0 – this is because short is a 16-bit integer type, so fractional data is discarded, and there is no half-precision floating point data type in C for us to use instead. In any case, such a data type would still require us to truncate half the bits in our value, still losing information.

Instead, we scale the value up by multiplying by before rectifying and casting, as shown in the first line of the function in figure 2, which ensures that all 16 bits are used, allowing us to keep the maximum amount of information. The reason we don’t use is that it would cause bits to overflow the value, leading to incorrect results in the output.

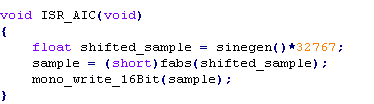


Figure - Interrupt Service Routine for Exercise 2

# Section 3 – Scope Traces

# Appendix – Code Listing