Author: Lance Huang

Application Note about Signal Processing

This application note is focused on the signal processing portion of the project. After RF signal is received by the receive-end antenna, it goes through an amplifier, then mixer, and lastly an active low-pass filter. The output of the active low-pass filter will be fed into an ADC.

For our quarter one design, we used the laptop computer's sound card as the ADC. Then we process the ADC digitized signals with matlab. However, in our quarter two design, because we want to do real-time signal processing and build an independent and portable Radar Shield System, we chose to use a Stellaris LM3S8962 microcontroller, which has a built-in ADC and OLED display. So the entire signal processing and result generation can be handled within the microcontroller.

We used an outside source FFT file to perform the Fourier Transform. Unfortunately, the FFT file was written with assembly language by the original author. And the author set up the code such that it will only take 128 digitized time-domain input data at a time (input buffer size = 128) and output 64 frequency-domain spectral data. This limitation constrained our choice of the ADC sampling rate. For example, if we use a sampling rate of 20ksps, the time between taking two samples is 1/20k seconds, and filling an input buffer size of 128 only needs 1/20k x 128 = 0.0064 second = 156.25Hz. So for input signals whose frequency is lower than 15.25Hz, a full period of signal length won't be able to fit in 128 input buffer. And in practice, our test results showed that the higher the sampling rate, the lower the resolution it will be. Based on our need, we decided to a sampling rate of 4ksps, which can detect object up to 10 meters away and with a resolution of 30Hz.

If we have an ideal system which has very few and low noise, the greatest element among the 64 FFT output is the detected object. We then use the index of this greatest element to calculate the corresponding frequency. The formula for calculating the corresponding frequency is, frequency = (index #/128) * (sampling rate/2). However, for many reasons, our system can't be ideal and the FFT result would include a lot of noise, especially in the low frequency range. In this case, the greatest element in the FFT result is not really the detected object but noise.

The best way to eliminate noise is to debug the hardware settings and find out the cause of the noise. We need to make sure we use low-noise RF components (components with a noise figure of 0.8 dB or below are good enough). And we need to make sure all transmission line runs smoothly without any abrupt turning point. If we have difficulty eliminating the cause of noise in hardware settings, we have to deal with it using the software approach.

If we can't root out noise in the hardware settings, the FFT result will inevitably be affected by noise. The best way to undermine the effect of noise is to do subtraction between two consecutive FFT results. Let's look at a simplified model. Our ADC will continuously generate groups of 128 time-domain inputs into the FFT function. For every group of 128 inputs, the FFT will output an array of 64 frequency-domain data. So for example, at time A, the FFT output array A. Then the subsequent FFT output array B is at time B. If the noise of our radar system is caused by some stable mechanism such that the noise itself is stable. For example, if in the FFT output, we can always observe a constantly high magnitude at 150Hz, 180Hz, and 250Hz regardless of the distance of the object we're trying to detect, we can conclude that the high magnitudes at those specific indexes are caused by noise. We can then use Array B to subtract Array A. By doing this, the noise which appears at the same index (frequency) will cancel out each other after the subtraction. And the moving object will less likely be affected because at time A, the moving object corresponds to index A, but at time B, the moving object correspond to index B. So the subtraction of the two arrays won't cancel out the index represents the object. Ideally, after the subtraction, all indexes will have a value close to zero except for the index corresponding to the moving object.

However, using one recent FFT output array to subtract the earlier output array is not enough because noise are not necessarily totally stable in terms of magnitude and frequency. For example, at time A, a noise appears at index 13 with magnitude 37 could jiggle over time and appears at index 14 with magnitude 31. So the best way to do it is to continuously collect a few FFT output arrays first, say 10 arrays (array_0, array_1 … array_9), then we wait for one second. The reason behind this is that we assume during these first 10 arrays we collected, the object is at one position. Then we will wait one second to let the object move on to a new position. Then we start again collecting the same amount of FFT output arrays (say array_10, array_11 … array_19). After collecting these 20 arrays, we let *diff_0 = array_10 – array_0, diff_1 = array_11 – array_1 … diff_9 = array_19 – array_9*. Inside each diff_0, diff_1, … diff_9 array,

we find the max element and record its index #. We then check which index # among these nine “diff_#” array has the most frequent occurrence. The most frequent index # is used to calculate the corresponding frequency.

Just to note that if a better microcontroller with a faster computing power is available, it’s better to collect as many FFT output arrays as possible for subtraction. By having more arrays, this algorithm is less susceptible to random noise.