

Hydrophone Sampling

25th March, 2007

Colin Bradbury

The TMS320C2812 DSP contains a programmable Analog to Digital Converter (ADC) suitable for sampling multiple input channels automatically. The ADC module consists of a single Converter, two sequencers, 16 sample-and-hold circuits (each connected to an analog input pin), and 16 result registers. The sequencers can be programmed to sample multiple inputs sequentially, a single input repeatedly, or some combination. A sequence is initiated by a Start-Of-Conversion (SOC) signal from an event-generator module outside the ADC. The conversion values are stored consecutively in the result registers.

In a submarine hydrophone application, the objective is to compare the relative timing of a signal arriving at each of several hydrophones. Ideally then, the hydrophones should be sampled simultaneously. With a sequential ADC, however, a simple sequence of samples has a built-in skew between the samples. This skew may have to be compensated for in the signal processing software. The alternative method advocated here is to sample each of the hydrophones twice, first in a forward order and then in a backward order, all combined together in a single sequence. In a three hydrophone system, the sampling order would be 1-2-3-3-2-1, followed by any other analog inputs that need to be sampled. The timing between consecutive samplings is a constant determined by the sequencer hardware. Averaging the results of the sample pairs for each hydrophone yields the signal value at the mid point of the sequence. Consequently, the averaged values are all simultaneous, removing the need for any compensation at the higher level.

At the end of a conversion sequence, an interrupt is generated and the software can then read the result registers, compute the averages, and store the results in memory somewhere. The results for any non-hydrophone inputs are also stored in memory for pickup whenever the appropriate processing routine executes. Unfortunately, this particular ADC gives the results in left justified form, so shifting operations are required before averaging in order to avoid overflow problems. Ideally, the results should be in right justified form. The coding of the interrupt routine is fairly simple:

```
ADCisr () {  
    H1sample = (ADCresult0 >> 4) + (ADCresult5 >> 4);  
    H2sample = (ADCresult1 >> 4) + (ADCresult4 >> 4);  
    H3sample = (ADCresult2 >> 4) + (ADCresult3 >> 4);  
    Store2memory (); }  
}
```

Note that the stored results are 13-bit values instead of the 12 bits from the ADC.

In a real-world application such as an AUV, we are concerned with the overall performance of the system rather than the performance of individual modules. This simple interrupt routine has high performance in itself, but requires memory resources for storing the samples and imposes a load on the downstream processes by having to

recover the samples from memory. The downstream processes are usually a noise filter, low-pass filter, high-pass filter, and then the actual signal-processing routines.

A typical noise filter involves some form of averaging a number of samples. The various methods of averaging are all about equal. Random noise is reduced by the square root of the number of samples involved. Multiple passes of a filter reduce the noise by the square, cube, etc. of the single-pass improvement. Thus two passes of a three-point average is superior (as regards noise reduction) to a single pass of a five-point filter. The downside of such filters is that they also attenuate the signal being processed. The actual filter chosen should provide the best compromise for maximum noise reduction against minimum signal attenuation. To a large extent, this depends on the sampling frequency and the frequency of the signal being processed.

For a sampling frequency of 500 KHz and a signal frequency in the range 20 to 40 KHz, the author prefers the two-pass, 1:2:1 weighted average noise filter. The reasons for this are that it is simple, fast, and wastes none of the range of result values. The first pass algorithm is a three-line computation (per hydrophone):

```
H1average = (H1lastSample * 2) + H1previousSample + H1sample;  
H1previousSample = H1lastSample;  
H1lastSample = H1sample;
```

Since all the samples are only 13 bit values, the computation yields a 15-bit average without the need for any form of division.

The accuracy of the result may be of concern in some systems. The ADC gives values accurate to within half a bit – i.e. the error in each value is in the range $-0.5 < \text{error} < 0.5$ so with eight such errors involved in calculating the average, the worst case total error could be as much as two bits. However, since most of the errors are unrelated to each other, the Central Limit Theorem asserts that the total error will be less than one bit. This error can be effectively eliminated by shifting the average down by one bit position:

```
H1average = H1average >> 1;
```

The second pass algorithm is almost identical to the first:

```
H1result = (H1lastAverage * 2) + H1previousAverage + H1average;  
H1previousAverage = H1lastAverage;  
H1lastAverage = H1Average;
```

Again, since all the averages are only 14-bit values, the computation yields a 16-bit result without the need for any form of division. Putting all the sections together gives a composite interrupt routine as follows:

```
ADCisr () {  
    H1sample = (ADCresult0 >> 4) + (ADCresult5 >> 4);  
    H1average = (H1lastSample << 1) + H1previousSample + H1sample;  
    H1previousSample = H1lastSample;  
    H1lastSample = H1sample;  
    H1average = H1average >> 1;  
    H1result = (H1lastAverage << 1) + H1previousAverage + H1average;  
    H1previousAverage = H1lastAverage;
```

```
H1lastAverage = H1Average;  
// ... similar for other hydrophones  
Store2memory (); }
```

The total processing involved is just four addition and five shift operations for each hydrophone, plus a few fetch and store operations. In addition to the result storage, the memory requirements are just four 16-bit values for each hydrophone. The 'C2812 DSP contains eight 32-bit auxiliary registers which could be used to meet the memory requirements and thus achieve maximum processing speed.

The next question is whether or not the next downstream process should also be folded into the interrupt routine. The arguments in favor of doing this are that it further reduces the memory requirements and improves overall performance. The arguments against are that low-pass and high-pass filters involve several multiplication operations, each of which may be considerably slower than addition of shift operations. The consequence would be that the interrupt routine would take significantly longer to execute and this in turn may force a slower sampling rate in order to avoid interrupts being missed. The bottom line is a tradeoff between sampling frequency and memory requirements. An additional consideration is that the downstream filters may not help the subsequent processing, so they could be completely omitted with no penalty. Simulation results indicate that using a high-pass filter is detrimental to the subsequent signal processing because it tends to change noise into signal waveforms.

This is the end of this White Paper. The use of these techniques without proper acknowledgement of the author in all written works will cause your sub to be cursed and sink to the bottom of the competition arena. Be sure to read the companion White Papers on the hydrophones mathematical model, Pyramidal Frequency Search, and Synchronous Fourier Transforms. Coming soon to a website near you!