DSP-based Oscillator Synchronization for Virtual Phased Array Development

Kurt Snieckus

March 5, 2012

Introduction

Beamforming is a directional signal propagation technique in which multiple elements are used to transmit or receive the same signal with different phases and amplitudes to allow for directional RF prorogation. The oscillators driving the elements used in beamforming systems always need to be synchronized to within a small fraction of the carrier period to function correctly. In the development of new "Virtual Phased Array" systems, where there is no master oscillator source to synchronize the transmitters or receivers, it is necessary to have a system to synchronize the oscillators on an extremely precise level in a distributed way.[1] This report details the beginnings of the implementation of a proof of concept oscillator synchronization system based around low-cost DSP hardware running at audio frequencies.

Background

The system developed in this report consists of an algorithm running on DSP hardware that receives bursts of sinusoidal oscillation, and produces a continuous oscillation synchronized to the bursts. This will allow for a system of many nodes to be sending bursts of oscillation to each other while synchronizing themselves with the bursts from other nodes resulting in each node being accurately synchronized with all the other nodes. Several signal processing techniques need to be combined to create this system. The incoming burst needs to be detected, the phase offset needs to be calculated and a local oscillator needs to be generated. Future systems will also implement a Kalman filter to keep the oscillator synchronized between bursts. This section will explain the theory behind the implementation of these components.

Signal Detection

To detect the incoming burst of sinusoidal oscillation, the input signal is continuously cross correlated with a generated sinusoidal signal of the approximate same frequency. The following equation describes cross correlation for a buffer size of N, where $x[n]$ is the incoming signal buffer and $s[n]$ is the generated signal buffer, and $y[n]$ is the result.[2]

$$y[n] = \sum_{n=-N/2}^{N/2} x[n]s[n+m]$$

This works to detect the incoming signal even for the expected slight variations in generated signal frequency and phase vs. incoming signal frequency and phase.

Phase Determination

The most obvious way to determine the phase of an incoming signal is through down-mixing with a local oscillator. Here ω_i is the frequency of the internal oscillator, ω_e is the frequency of the input signal, ϕ_e is the phase offset of the external signal relative to the internal oscillator, and A is the incoming signal amplitude.

$$\sin(\omega_i t) A \sin(\omega_e t + \phi_e) = \frac{1}{2}(A\cos((\omega_e - \omega_i)t + \phi_e) - A\cos((\omega_e + \omega_i)t + \phi_e))$$

Next we apply a low pass filter to remove the term with the $(\omega_e + \omega_i)$ leaving us with.

$$LPF\left\{\frac{1}{2}(A\cos((\omega_e - \omega_i)t + \phi_e) - A\cos((\omega_e + \omega_i)t + \phi_e))\right\}$$

$$= \frac{1}{2}(A\cos\left((\omega_e - \omega_i)\,t + \phi_e\right)$$

This will give a very low frequency sine wave that contains the phase offset ϕ_e. Next we assume that $\omega_e \approx \omega_i$ which gives the following equation.

$$\frac{1}{2}(A\cos\left((\omega_e - \omega_i)\,t + \phi_e\right) = \frac{1}{2}A\cos\phi_e$$

Now we if we divide out the gain A and $\frac{1}{2}$ term we can take the inverse cosine to get the resulting phase offset we desire.

$$\arccos\cos\phi_e = \phi_e$$

This method is not desirable however, because of the dependency on knowing the gain of the incoming signal. The incoming signal gain is not necessarily constant because of potentially varying channel conditions and an automatic gain control does not estimate the gain accurately enough. We need an alternate method to determine the phase offset of the incoming signal.

The alternate method used here is by way of a maximum likelihood estimator. The following equation produces a phase estimate of the unknown signal $x[n]$ of length N having a frequency estimate of $\hat{f}_o$.[3]

$$\hat{\phi} = \arctan\frac{-\sum_{n=0}^{N-1} x[n]\sin 2\pi\hat{f}_0 n}{\sum_{n=0}^{N-1} x[n]\cos 2\pi\hat{f}_0 n}$$

This computes a phase estimate regardless of the gain of the incoming signal $x[n]$ and does so with relatively few operations. The phase offset produced by this computation is relative to the phase of the sampling producing the $x[n]$. Thus, to produce a consistent result, the buffer used by this algorithm must always start at a multiple of sampling frequency divided by signal frequency.

Methods

The whole system was first simulated to verify that the theory used is correct and provide a base system topology for implementation on real hardware. Once the simulation was determined to function properly the system was implemented with real signals on DSP hardware.

The implementation on the DSP used a mixture of sample and batch processing techniques. Along with processes that are inherently sample based processes, like filling the buffer and local oscillator generation, the signal detection algorithm is computed for each sample because it determines when to stop filling the buffer which is a sample based process. The MLE phase estimation is setup as a batch process because it requires a significant amount of resources, and while for the base configuration of buffer lengths and sampling rates used for these tests it could be run for each sample, this is unnecessary and for faster synchronization frequencies and sampling rates it would only work in a batch processing configuration. This also provides a place to add additional computations that only need to be run once for every burst like frequency estimation and Kalman filtering.

A state machine facilitates the batch processing of the MLE phase estimate and keeps the entire signal detection process and resulting phase estimate from being triggered multiple times per burst. The state machine uses a counter to keep track of the state as well as to determine a length of time to wait before starting to run the signal detection algorithm again. The state machine counter is both modified by the sampling processing code and the batch processing code to advance the state, but is primarily controlled in the sample processing function.

The signal detection algorithm was initially designed to cross correlate the incoming signal with a generated sinusoidal signal for each sample period. This, however, proved to be too resource intensive to accomplish for each sample. A cross correlation function based upon the equation giving in the Background section used a maximum of $1.209 * 10^6$ cycles for a buffer size of 100. This is more than 43 times the number of cycles available to each sample at an 8kHz sampling rate in the hardware configuration used here.

While a faster cross correlation algorithm can be implemented with FFTs, this was found to be unnecessary. Only the maximum of the output of the cross correlation is taken and compared to a threshold to determine if the signal is present. This maximum was found via simulation to always occur when the two buffers were lined up with each other (when $m = 0$ in the cross correlation equation given in the Background section). Simulations confirmed that simply taking the dot product of the buffer with the generated signal produced the same result. Because this is done for each sample, it is essentially taking a continuous cross correlation, and results in significantly reduced resource usage. With this scheme only 5950 cycles were needed to compute the dot product which is only 21% of the cycles available for each sample at an 8kHz sampling rate.

The Maximum Likelihood Estimator was chosen as described in the Background section because it is not dependent on the incoming signal gain to cal-

based processing is started again, filling the buffer and looking for the incoming burst. Figure 1 shows the parts and the flow of information in the system. It also denotes the difference between batch and sample based processing.

Simulation

Initially, a MATLAB script was written to simulate the incoming bursts, signal detection, and phase estimation. The script generates a gated sine wave at a frequency very close to 1kHz, to simulate a real system where the clocks are not synchronized. It then iterates over each sample of the generated incoming signal. It fills a small buffer with the "incoming signal" and takes the dot product of the buffer with a sine wave generated at 1kHz. Once the dot product output is greater than a value of 45 and the index of the incoming signal is a multiple of the signal frequency divided by the sample frequency—8 in this case—the MLE phase estimate of the buffer is computed and stored in another buffer. Once the entire sequence has been iterated over, plots of the incoming signal, the dot-product output for each sample and the phase estimates are produced to confirm proper operation.

Hardware Implementation

Once the simulation produced the expected output, the system was implemented on a Spectrum Digital TMS320C6713 DSK. This DSP development board includes audio input and output facilities via an AIC23 Codec that can run at varying sample rates. For this implementation, a sampling rate of 8kHz was used. The Texas Instruments C6713 DSP on the Spectrum Digital DSK board runs at 225mHz. For the 8kHz sampling rate used, 28125 CPU cycles are available to process each sample. For these tests, a 1kHz oscillator frequency was used, but every effort was made to allow easy configuration of any oscillator frequency.

An Agilent 33220A Function Generator was used to produce the gated sine wave output. It was set for a sine wave frequency of 1kHz, a burst length of 20 cycles—160 samples for the DSP—and a burst period of 1 second. The natural variation in the function generator clock and the clock of the codec feeding samples to the DSP caused the desired slight variation in the frequency of a 1kHz sine wave generated by the DSP to the 1kHz sine wave from the function generator. The output of the function generator was supplied to the left and right inputs to the codec on DSK DSP board and an oscilloscope. The output from the codec was also connected to the oscilloscope for comparison to the function generator signal. The system as connected can be see in Figure 2.

The algorithm running on the DSP was setup as in Figure 1. The sample processing elements are run in an interrupt service routine that gets triggered every time a new sample is available from the codec, once every 28125 CPU cycles. The ISR runs certain sample based processes based on the current state of the state machine. The batch processes are run from the main program

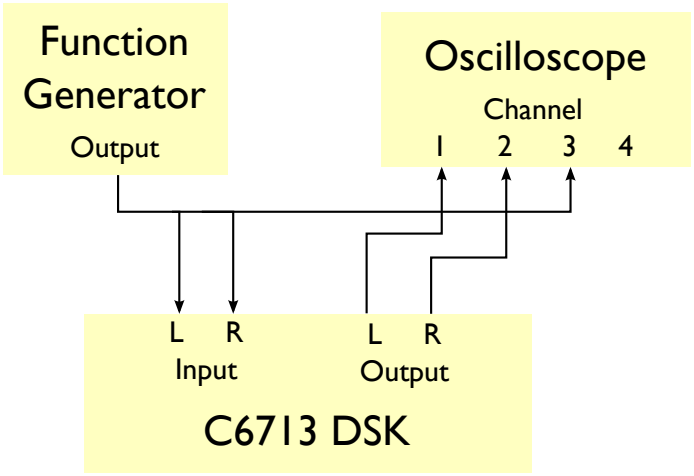

Figure 2: Diagram of the connection of the DSP board to test equipment.

"while(1)" loop, depending on the state of the state machine. The state machine has the following 4 states:

1. The initial state fills the buffer and takes the dot product of the buffer with the generated signal for correlation. If the dot product is greater than a threshold, the state machine goes to state 2.

2. In this state, the signal has been detected, but the buffer still needs to filled until a period boundary is reached for proper phase estimation. Once the period counter reaches 0, the state is incremented.

3. State 3 initiates the MLE phase estimation in the main program loop. Filling of the buffer is stopped so as not to corrupt the MLE calculation operating on the same buffer, and the dot-product calculation is stopped to free up CPU cycles. Once the phase estimate is generated and saved, the state is incremented.

4. This final state restarts filling the buffer, and starts a counter of samples to make sure the buffer is sufficiently void of pulse samples so that another dot-product correlation will not trigger the system again for the pulse that has already been processed. After this counter reaches its limit the state is reset to 1.

Outside of the state machine, the local oscillator generation is run for every ISR call. It uses the same counter that keeps track of the period. The local oscillator with the last phase estimation is output on the left channel of the codec, to be viewed on an oscilloscope. A further constant phase offset is applied to this output to account for latency in the DSP system. This results in the synchronized local oscillator lining up perfectly with the input signal burst.

The right channel output of the codec contains debugging information. It's an output of a non-phase-corrected local oscillator amplitude modulated with the current phase estimation. Signal is only output during states 3 and 4 of the state machine, to confirm when they are triggered.

Results

After working out bugs in the code, the simulation and hardware implementation worked as expected. Some basic robustness tests were performed on the hardware implementation, and the code was profiled to determine what resources are used by the implementation.

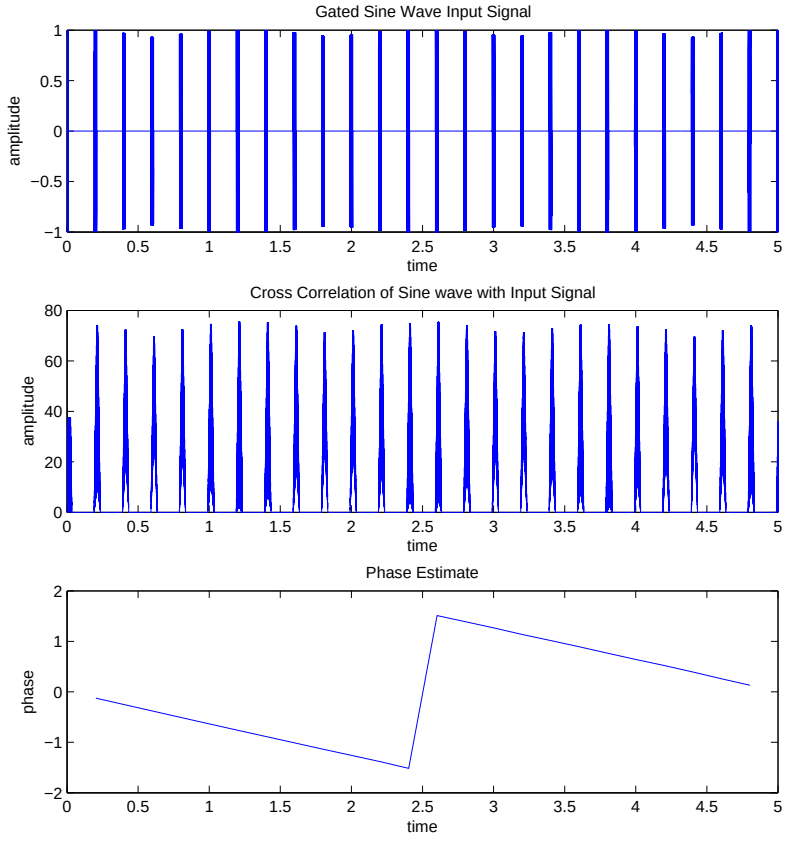

Figure 3: Output of MATLAB simulation

System Simulation

The system was simulated with a MATLAB script roughly approximating what would be implemented in hardware. Figure 3 shows the output of the simulation. The first plot shows the incoming gated sinusoid as generated by the script. The second plot shows the dot-product correlation of the input signal, which has peaks at expected times. The final plot shows the phase estimates for each pulse which are steadily decreasing, as expected for the frequency offset used in the simulation.

Implementation on C6713 DSK

An oscilloscope trace of the output from the DSK running the phase estimation algorithm is shown in Figure 4. The magenta sinusoid on channel 3 is the input to the algorithm, and it can be seen that once the algorithm computes the phase offset, which occurs when the bottom debugging signal starts, a discontinuity occurs in the yellow signal on channel 1 resulting in the two signals being synchronized. In this particular instance, a phase flip has also occurred, bringing the signal from π radians phase difference to 0 radians phase difference. The cyan waveform on channel 2 is the debugging waveform, started when the MLE phase computation is triggered and modulated to the size of the phase estimate generated by that MLE computation. A few samples are output at normal amplitude at the beginning to provide a triggering point for the oscilloscope.

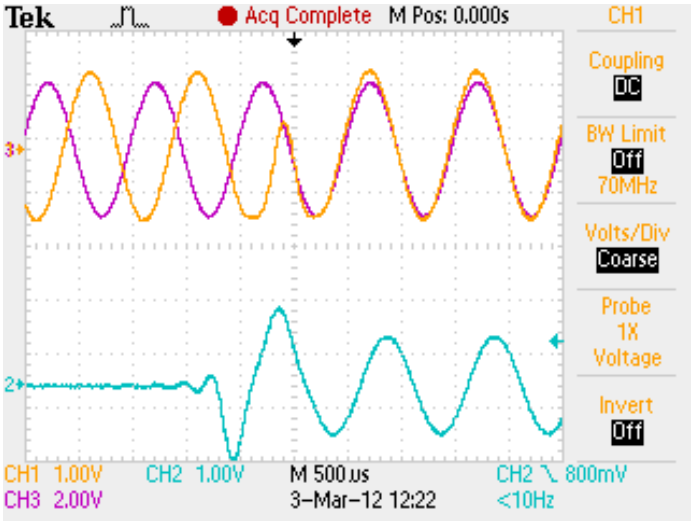

Figure 4: Oscilloscope traces of the DSP output signals, channel 1 and 2, and the function generator input signal, channel 3

Figure 5 shows a plot of the phase estimates generated by the DSP. These were stored in a 200 point buffer before the DSP program was paused and the

values downloaded from DSP memory and plotted with MATLAB. The figure clearly shows that the MLE produces fairly accurate phase estimates every time, and that the phase wraps every $\frac{\pi}{2}$ radians.

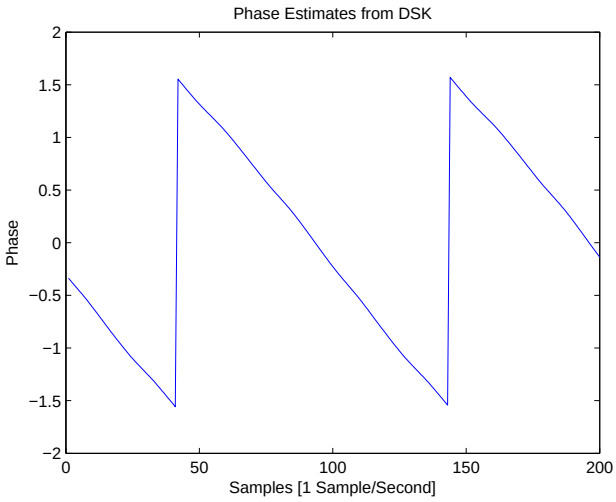

Figure 5: Phase results computed by DSP

Table 1 shows the results of profiling the code run on the DSP with a cycle-accurate simulator. These results indicate that the DSP has a significant amount of resources to spare for all the processes and that the code could easily be run at higher clock speeds. Specifically of concern is the dot_prod() function which needs to be run every cycle when looking for a pulse. In the configuration used for these tests, the dot_prod() function only used 29% of the available CPU cycles for each sample. If the sampling rate were to be increased significantly, the size of the buffer dot_prod() is run on may need to be decreased or the function pipelined, or both.

Function	Calls	Max Cycles
dot_prod()	298	5950
Sample ISR	90302	7040
lo()	180306	460
mle()	298	10873

Table 1: Inclusive maximum cycles from the TI cycle-accurate simulator for signal processing functions with a buffer length of 100 samples.

Conclusions

This system is the first step is development of virtual phased array implementations. This report details the successful implementation of an oscillator phase offset determination scheme, providing the basis for a full oscillator synchronization system that can then be scaled up in frequency for use in RF systems. Further work from here would involve developing a phase unwrapping algorithm, determining frequency offsets from the slope of the generated phase estimates, and then a Kalman filter system to provide a truly synchronized oscillator. At that point two or more DSPs could be setup to send their pulses to one other resulting in their oscillators being fully synchronized.

Bibliography

[1] Robert D. Preuss and D. Richard Brown III. Two-way synchronization for coordinated multicell retrodirective downlink beamforming. *IEEE Transactions on Signal Processing*, 59(11):5415–5427, 2011.

[2] William A. Sethares C. Richard Johnson Jr. and Andrew G. Klein. *Software Reciever Design*. Cambridge University Press, 2011.

[3] Steven M. Kay. *Fundamentals of Statistical Signal Processing : Estimation Theory*. Prentice-Hall, 1993.