2008 American Control Conference
Westin Seattle Hotel, Seattle, Washington, USA
June 11-13, 2008

Efficient and Flexible Simulation of Phase Locked Loops, Part II: Post Processing and a Design Example

Daniel Y. Abramovitch

***Abstract*— Although phase-locked loops (PLLs) are arguably the most ubiquitous control loop designed by humans, system theory analysis seems to lag behind the practice of implementation. In particular, full simulation of PLLs is rare. This paper will explain the reasons for this and offer an efficient and flexible simulator for PLLs. Part I of this paper [1] described the simulator design. This part will describe the post processing and show some results.**

I. Introduction

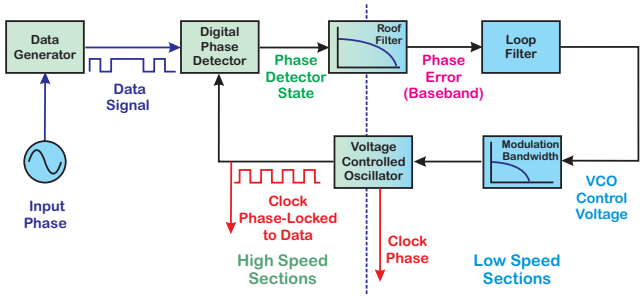

Fig. 1. Simulation block diagram for a classical digital phase locked loop. On the left side of the diagram: data, VCO, and phase detector simulated with component level blocks that are very efficient. On the right side of the diagram: filters and modulation bandwidth are simulated using designs from Matlab that are very flexible and derived from lab measurements.

The simulation of PLLs is made difficult by the stiffness of the feedback loop. The data signals typically have several orders of magnitude higher frequencies than the phase modulation. Part I of this paper [1] described the simulator design. This part will describe the post processing and show some results.

II. Design Example

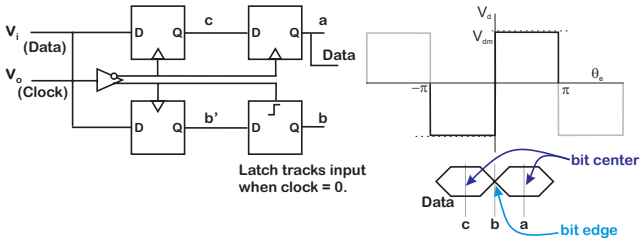

Fig. 2. Block diagram for a Bang-Bang phase detector used in clock-data recovery PLLs.

Daniel Abramovitch, Nanotechnology Group at Agilent Laboratories, 5301 Stevens Creek Blvd., M/S: 4U-SB, Santa Clara, CA 95051 USA, danny@agilent.com

The first step for any PLL design is the phase detector. Without a means of detecting the relative phase of an input signal and some sort of clock, there is no PLL. So, the design of phase detectors is critical. As described in [2], phase detectors vary greatly by the type of input signals that they deal with. Signals that deal with modulated sinusoids can be examined with a sinusoidal phase detector, which one gets by mixing the input and clock signals. At the other end of the spectrum are complex logic phase detectors such as the Alexander (or Bang-Bang) phase detector and the Hogge phase detector.

The Hogge detector is a linear phase detector that uses flip-flops and gates that can recover phase from NRZ data [3], [4]. This requires that the VCO clock period be the same as the data (bit) period.

The Bang-Bang detector is a binary phase detector using flip-flops that can recover phase from NRZ data [5], [6]. In this case, the detector provides literally one bit of information *i.e.,* that the clock is early or late. A slight extension of this allows that if the clock is exactly in phase with the reference, the phase detector provides a zero output. These detectors give up the linear performance of the Hogge detector but gain in being easy to put into an integrated circuit without needing manual calibration. A full rate bang-bang detector requires that the VCO clock period be the same as the data (bit) period. This is shown in Figure 2. A half rate bang-bang detector uses two oscillators, working in quadrature but at half the period as the data. This relaxation of the oscillator requirements can become important when the circuit technology is being pushed to accommodate fast data rates.

What is important to note about these different phase detectors is that some require digital logic blocks, some require flip flops, and all require some math. By having a class library of primitives for these components, it is fairly easy to construct and test any of these (or other) phase detectors.

III. Filter Files

As mentioned in Part I of this paper, the filter design can be passed from Matlab to the simulation through a *.flt* file. A filter file, *VCOFilt1a.flt*, which was used for this simulation. is shown in Table I.

Note that all the information needed to allocate the appropriate data structures for the filter can be found here. The filter file is for a SISO filter, but the filter type (IIR), sample rate ($f_S = 10^{12}$ Hz), gain (1), offset (0), and order (3) are included before the filter structure has to be allocated by the

978-1-4244-2079-7/08/$25.00 ©2008 AACC.

Name:	VCOFilt1a
Type:	iir
Sample rate:	1e+012
One Liner:	This approximates our lab VCO with modulation BW
Gain:	1
Offset:	0
Order:	3
Coef:	
a[3]:	-0.99993214390113
b[3]:	0.07994108331177e-13
a[2]:	2.99986427501169
b[2]:	0.23982324993532e-13
a[1]:	-2.99993213111056
b[1]:	0.23982324993532e-13
b[0]:	0.07994108331177e-13

TABLE I

FILTER FILE, *VCOFilt1a.flt*, WHICH WAS USED FOR THE SIMULATION IN THIS PAPER.

simulation. The simulation then allocates a third order IIR filter and populates it with the a_i and b_i values.

IV. POST PROCESSING

The simulation of the loop is accomplished with very rapid C/C++ code. A salient feature of this simulator is that it saves data to Matlab files for post processing using the Matlab API. It is a great feature for several reasons:

- It dramatically expands the post processing capabilities by using Matlab to work on vectors of simulation data.
- The simulation results can then be turned into a variety of measurements, including time domain, frequency domain, and histograms. based on the simulation results.
- It gives access to Matlab's handle graphics.

A. Time Domain

The time domain processing is relatively straightforward. One simply windows the time responses one wants to look at. Again, the stiffness of PLLs becomes a factor. For example, the plots in Figure 3, show including a long enough time run to view the complete dynamics of the signal phase (bottom plot) means that the signal plots (top plot) are essentially a blur. To actually see the shape of the clock and reference signals requires zooming in on the time responses as shown in Figure 4 and 5, where Figure 4 shows the at the beginning of the simulation and Figure 5 is at the end. All of these plots come from a single simulation run.

B. Histograms and Clock Edges

Matlab code can generate histograms of the recovered clock edges. This code takes a given signal an centers it around 0. It then looks for sign changes in the sample signal and interpolates between these using an estimate of the slope at the transition to get a reasonable measure of the zero crossing times. This is shown in Figure 6. From here, it can separate out leading and trailing edge transitions (since they should be interlaced) and plot these transitions (edge-to-edge, leading edge, and trailing edge) on a histogram plot along with their mean and standard deviation.

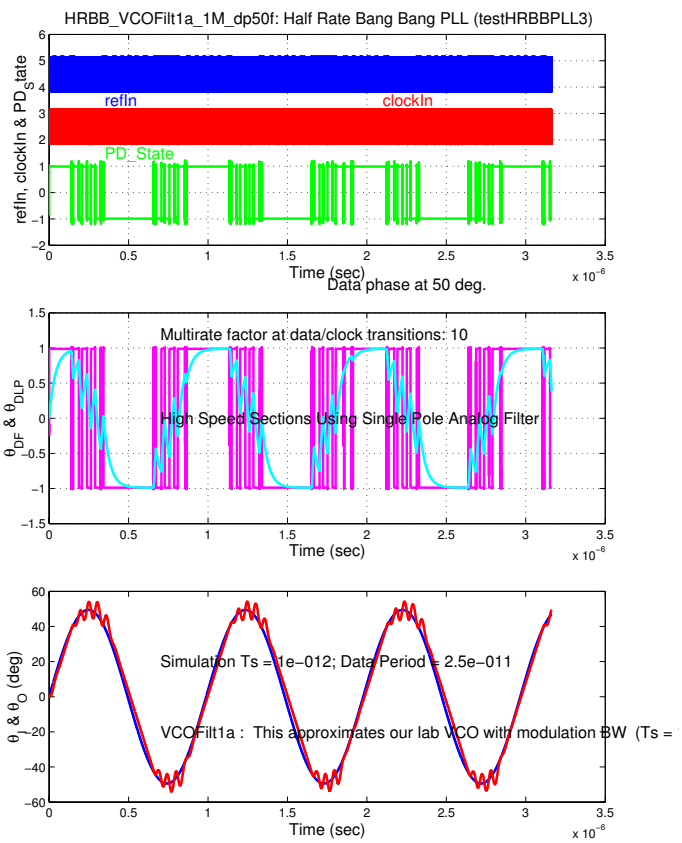

Fig. 3. Time domain response of Half Rate Bang Bang PLL simulation. Vertical fuzziness seen in phase detector output due to filters used by Matlab's decimation feature. The refIn signal is the data input. The clockIn is the recovered VCO clock. The PD State is the state of the phase detector. This is passed through two different low pass filters. The LP PD Out is low passed with a 4 GHz bandwidth. The VLP PD Out is the phase detector output passed through a 400 MHz bandwidth filter. In the bottom plot, the input phase is θ_i and the recovered clock phase is θ_o.

These edges are computed relative to adjacent edges. For example, the edge-to-edge time is computed by zero centering the signal and measuring the time between two successive zero crossings. The histograms of edge times are built up from these comparisons of adjacent edges, leading edges, and trailing edges. In other words, this generation of histograms comes from data taking akin to that of a real time scope. The interpolation of the zero crossing times using the estimated slopes shown in Figure 6 also lowers the quantization noise in computing the zero crossings. This is an important addition to the capabilities of the simulator and shows the importance of generating this with a systems view. Without this interpolation, the sample rate of the simulation would have to be increased dramatically to get the same zero crossing accuracy. This would slow down the simulation and lessen its practicality. By understanding that the time quantization could be minimized with this bit of interpolation in the post processing, the sample rate for the simulation could be set by the dynamics of the loop components and not the needs of the histogram measurement.

In our design example, the computed histograms are shown in Figure 7. In it, edge-to-edge (top plot), leading edge

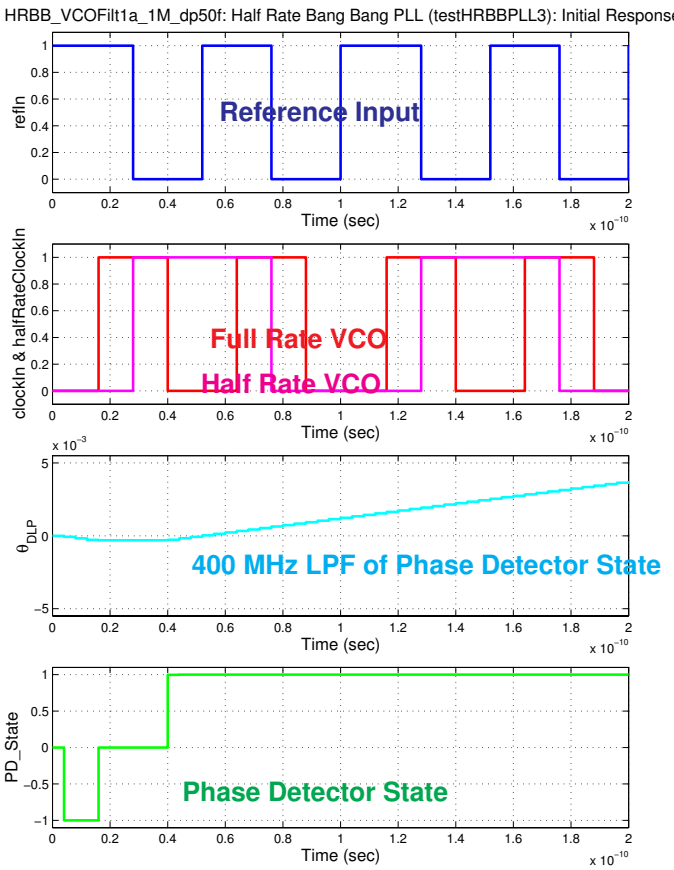

Fig. 4. Time domain response of Half Rate Bang-Bang PLL simulation. This plot is zoomed in to the beginning of the simulation time.

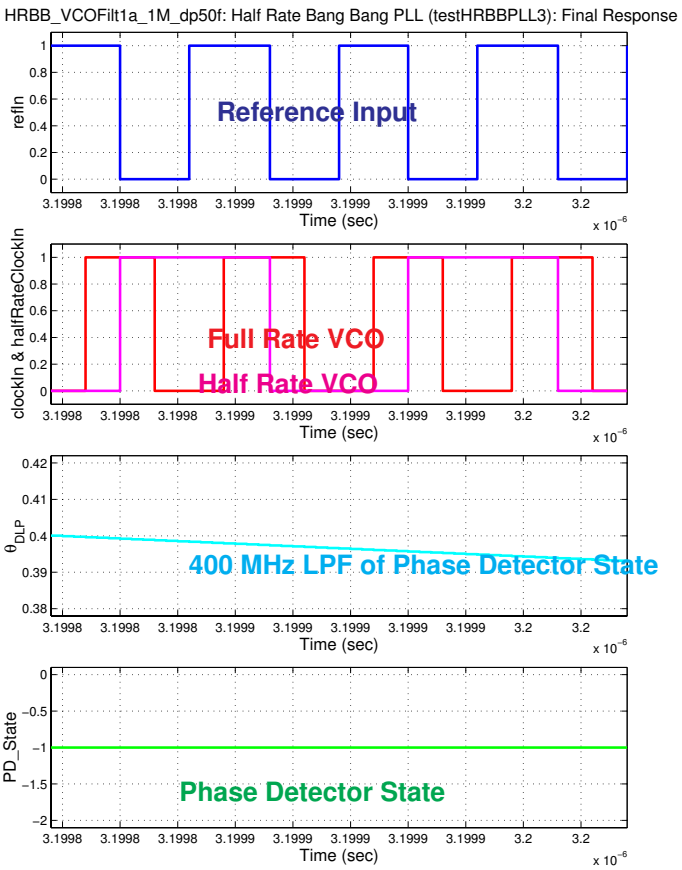

Fig. 5. Time domain response of Half Rate Bang-Bang PLL simulation. This plot is zoomed in to the end of the simulation time.

to leading edge (middle plot), and trailing edge to trailing edge (lower plot) times are plotted. If there were no variation in any of the edges, the histograms would in fact be lines. However, the jitter manifests itself as the histogram width. Another phenomena that is observable is a variation in edge-to-edge times in the top plot, even though the leading edge to leading edge and trailing edge to trailing edge times are each clustered around 25 pS. The bimodal distributions around 12 and 13 pS mean that the leading edge to trailing edge times are different than the trailing edge to leading edge times, possibly an indication of a DC shift in the signal. None of this is obvious from Figures 3 –5. Furthermore, this behavior is completely invisible in a modulation domain only simulation.

C. Frequency Domain Measurements

Once the simulation has been completed, the frequency content of the signals can be analyzed. Figure 8 schematically shows the spectral components of a measured sine wave without jitter. Note that the signal is composed of a noise floor and a spectral peak. In Figure 9, jitter has been added to the sine wave. We see that the noise floor is unchanged, but the spectral peak has been shortened and broadened. One of the great strengths of having an efficient simulator is the ability to generate enough data both with and without jitter on the reference, so that we can measure the effect of the loop on the output spectrums.

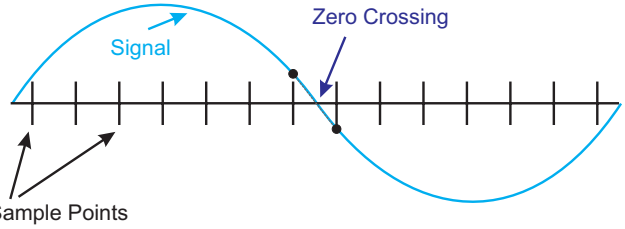

Fig. 6. Zero crossing interpolation. Unless the sample rates are carefully chosen to match the clock frequency and the samples are aligned to the signal, it is likely that zero crossings will occur between samples. Thus, by interpolating between the signal values at sample times around a zero crossing, one can get an improved estimate of the zero crossing time.

While it is faster to generate a spectrum using FFT methods, there is great flexibility in simulating spectrum analyzer calculations. Spectrum Analyzers are tools that predate the digital instrumentation era. In modern times, their functionality has been mimicked using digital computations. However, they are still useful for their high frequency measurement capability. The basic idea here is that a Spectrum Analyzer (SA) uses a tracking bandpass filter to isolate a narrow band of frequencies and gives the magnitude of that filtered output. The terms noise Equivalent Bandwidth and

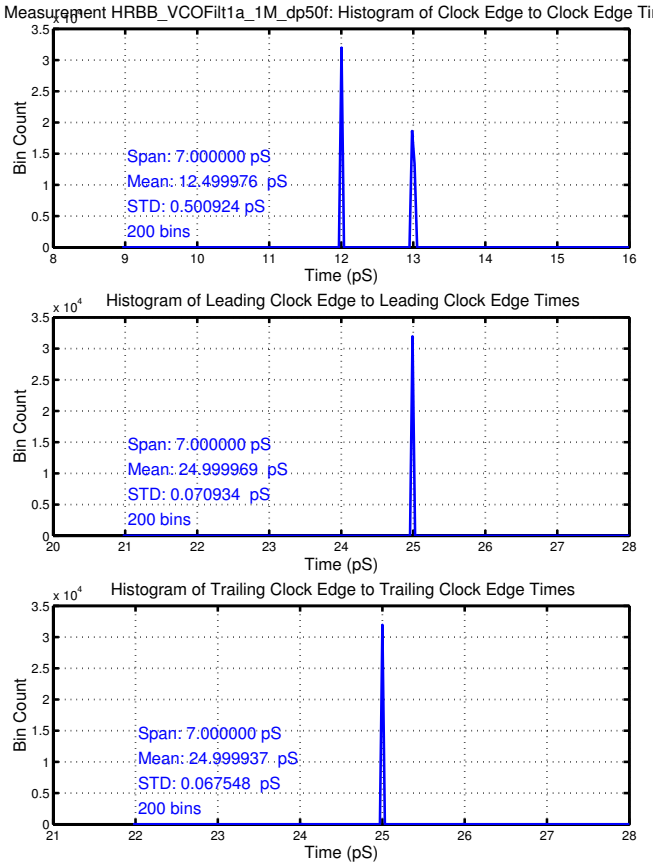

Fig. 7. Time domain jitter measurements of Half Rate Bang Bang PLL simulation. Top plot: edge-to-edge jitter. Center plot: leading edge to leading edge. Bottom plot: trailing edge to trailing edge.

Resolution Bandwidth come from these devices where

Noise Equivalent Bandwidth
$\approx$ Resolution Bandwidth
$\stackrel{\triangle}{=}$ 3 dB bandwidth of the tracking bandpass filter .

In truth, the filter center frequency does not change, but the signal is demodulated to some intermediate frequency where the filter center frequency resides (possibly DC).

To generate a spectrum analyzer simulation in Matlab, one has to return to what is actually being done. The tracking filter is implemented by mixing the data signal with a sinusoid of the desired frequency. The mixed signal is then integrated over a finite number of periods of the desired frequency so that only the component of the signal at the mixing frequency remains after the integral. A further low pass filtering step can be done to eliminate high frequency noise or other imperfections in the integral.

One of the key steps in the integration is to make sure that the integral is done over an integer number of periods of the desired frequency. That is, for each chosen frequency for which we want the spectrum we want an integral of

$$I = \frac{2}{MT_i}\int_{t_0}^{t_0+MT} s(t)\sin(2\pi f_i t)dt \qquad (1)$$

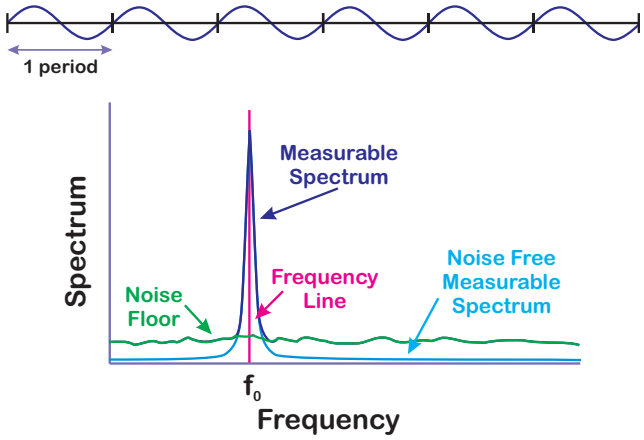

Fig. 8. Schematic representation of spectrum with no jitter.

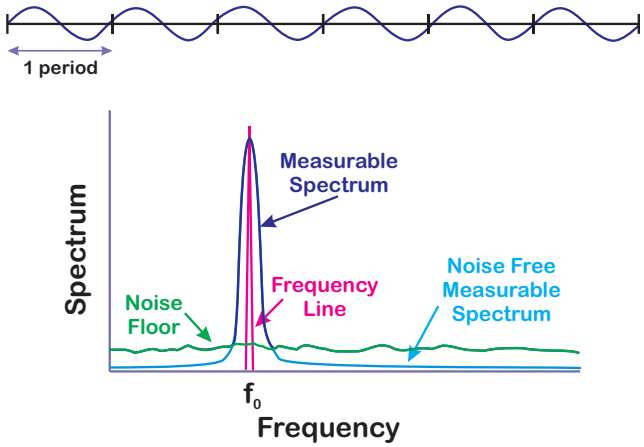

Fig. 9. Schematic representation of spectrum with jitter.

and

$$Q = \frac{2}{MT_i}\int_{t_0}^{t_0+MT} s(t)\cos(2\pi f_i t)dt \qquad (2)$$

where $T_i = \frac{1}{f_i}$ and M is an integer. This is shown schematically in Figure 10.

When we use a discrete approximation – as will happen with our simulation, these integrals turn into sums.

$$I = \frac{2}{MT_i}\sum_{k=0}^{N} s(kT_S)\sin(2\pi f_i k\frac{T_S}{T_i}) \qquad (3)$$

and

$$Q = \frac{2}{MT_i}\sum_{k=0}^{N} s(kT_S)\cos(2\pi f_i k\frac{T_S}{T_i}) \qquad (4)$$

However, since our simulation will have a fixed sample rate, most if not all integrals will involve the period of the sine wave not corresponding to an integer number of sample points *i.e.,*

$$MT_i \neq NT_S, \qquad (5)$$

where M and N are both integers, T_S is the sample period of the simulation, and T_i is the period of the frequency,

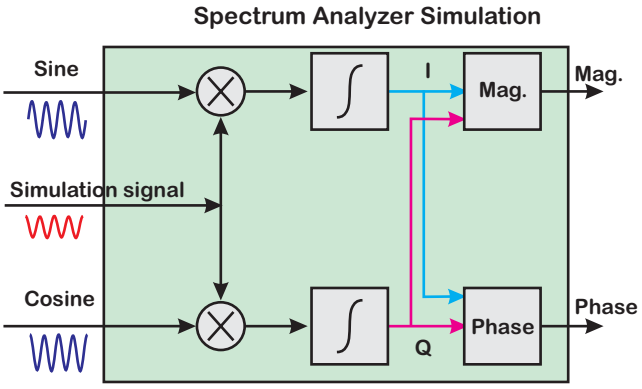

Fig. 10. Spectrum analyzer simulation integral.

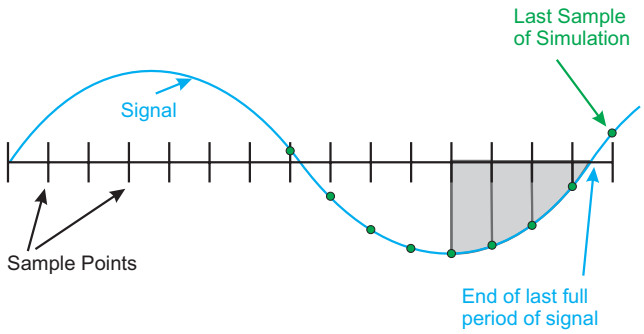

Fig. 12. The tail portion of a sampled sine wave. The integral must be adjusted for each new frequency to properly integrate over this fractional portion of a sample.

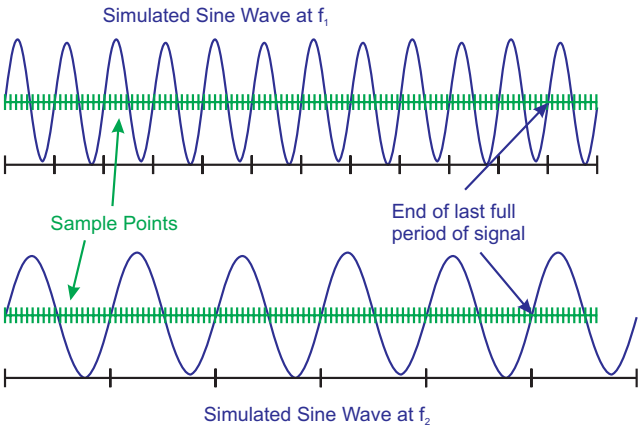

Fig. 11. Two different sine waves sampled at a constant frequency. Note that the end of integer periods of each wave do not correspond to a sample point.

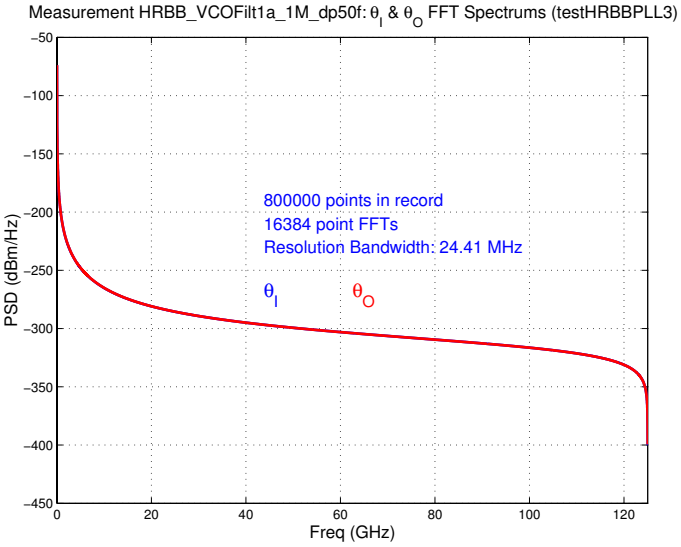

Fig. 13. Spectrums of input and clock phases. This plot uses FFTs.

f_i for which we want to know the spectrum. This can be seen schematically in Figure 11, where neither of the two sine waves have periods that end on an integer number of samples. The solution for an accurate integral is to compute the integral over the fractional sample period, as shown in Figure 12. This involves making a new tail end integral for each new frequency.

The plots of Figures 13– 16 show spectrums computed from the PLL simulations, using the spectrum analyzer simulation described above. Figure 13 and 14 show spectrums of reference signal phase, θ_i, and clock phase, θ_o. The plot in Figure 13 is rapidly computed using an FFT. However, the resolution bandwidth is limited to being a power of 2 and the starting frequency bin is at 0. The highest frequency bin is a function of the sample rate and the number of points in the time measurement. To match the accuracy of spectrum analyzer measurements, the Matlab based spectrum analyzer simulation was used. This does Fourier component calculation over specified frequencies rather than having a fixed frequency span based on the number of points as an FFT does. The cost is in considerably more computation time. The plot in Figure 14 is computed with a spectrum analyzer simulation – gives the user complete frequency selectivity. This is seen in the two plots in Figures 15 and 16, where different resolution bandwidths and start and stop frequencies are selected.

The plot of Figure 16 shows a spectrum that is zeroed in on the frequencies of interest with a very narrow resolution bandwidth. In particular, a resolution bandwidth of 1 MHz was achieved with a sample rate of 10^{12} Hz. This requires a minimum number of 1 million sample points. It is also useful to note that unlike two plots generated on a spectrum analyzer with different frequency ranges, the plots in Figures 15 and 16 are self-consistent as the are generated from the same time response data.

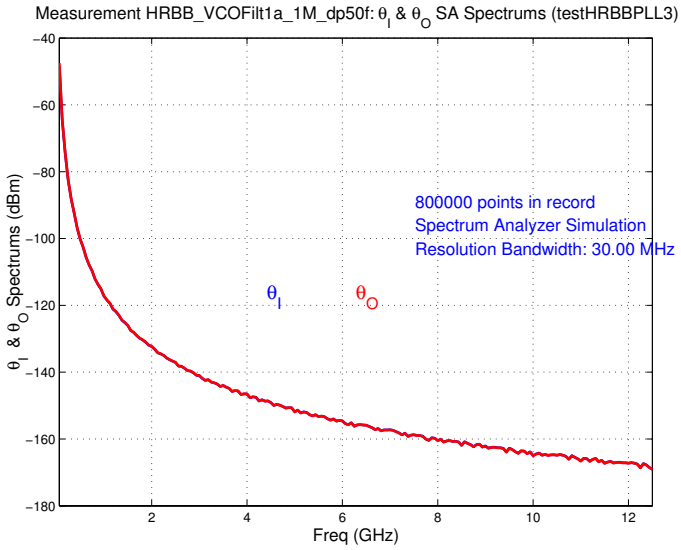

Fig. 14. Spectrums of input and clock phases. This plot uses a spectrum analyzer simulation. Note that the spectrum analyzer allows focusing in on a much smaller set of frequencies than the FFT method.

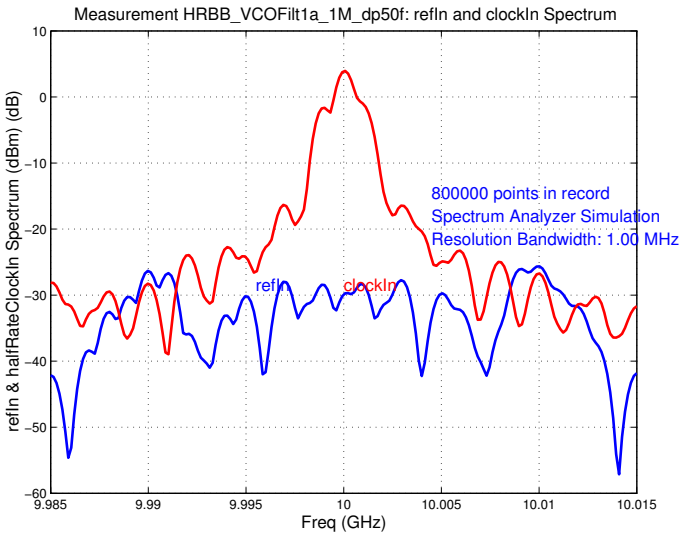

Fig. 16. Spectrums of input and clock signals. This plot is generated using the spectrum analyzer simulation. This plot focuses on frequencies around 10 GHz, with a 1 MHz resolution bandwidth.

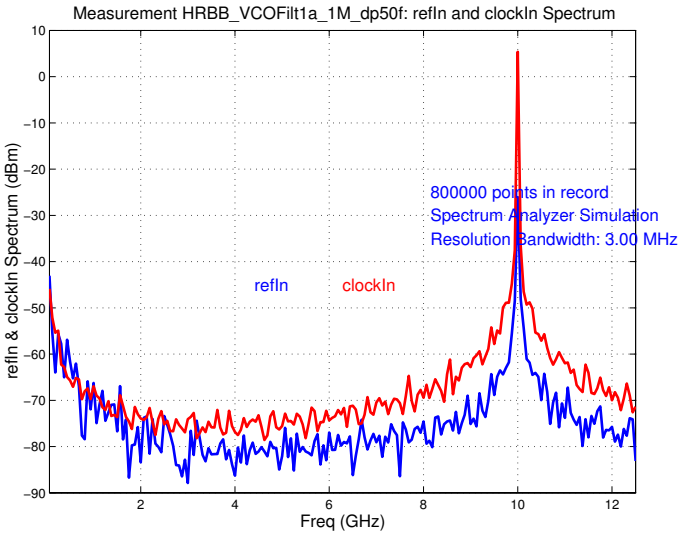

Fig. 15. Spectrums of input and clock signals. This plot is generated using the spectrum analyzer simulation. This plot gives a broad view of the spectrum with a 30 MHz resolution bandwidth.

V. Conclusions

Part II of this paper has discussed post processing methods used in the PLL simulation discussed in Part I [1]. A design example showing how a high speed Half Rate Bang Bang PLL can be simulated so that the time domain data and the phase response can be seen. The data can further be post processed to show jitter results in histograms and frequency domain results. This allows comparisons with measurements made of circuit prototypes with laboratory instruments.

Parts I [1] and Part II of this paper have tried to show the benefits of doing a full time domain simulation of a PLL. The combination of efficient loop simulation and extensive post processing reveals considerable data dependent behavior of the PLL in a way which is completely invisible to modulation domain only simulators.

References

[1] D. Y. Abramovitch, "Efficient and flexible simulation of phase locked loops, part i: Simulator design," in *Submitted to the 2008 American Control Conference*, (Seattle, WA), AACC, IEEE, June 11–13 2008.

[2] D. Y. Abramovitch, "Phase-locked loops: A control centric tutorial," in *Proceedings of the 2002 American Control Conference*, (Anchorage, AK), AACC, IEEE, May 2002.

[3] C. R. Hogge, Jr., "A self correcting clock recovery circuit," *IEEE Journal of Lightwave Technology*, vol. LT-3, pp. 1312–1314, December 1985.

[4] D. Shin, M. Park, and M. Lee, "Self-correcting clock recovery circuit with improved jitter performance," *Electronics Letters*, vol. 23, pp. 110–111, January 1987.

[5] J. Alexander, "Clock recovery from random binary signals," *Electronics Letters*, vol. 11, pp. 541–542, October 1975.

[6] R. C. Walker, "Designing bang-bang plls for clock and data recovery in serial data transmission systems," in *Phase-Locking in High-Performance Sytems - From Devices to Architectures* (B. Razavi, ed.), pp. 34–45, New York, NY: IEEE Press, 2003.