A PHASE LOCKED LOOP WITH ARBITRARILY WIDE LOCK RANGE FOR SOFTWARE DEFINED RADIOS

Salam Akoum (University of Utah, Salt Lake City$^+$; akoum@ece.utah.edu),
Behrouz Farhang-Boroujeny ($^+$; farhang@ece.utah.edu)

ABSTRACT

Phase locked loops (PLLs) are frequently used in Software Defined Radios (SDR) for carrier recovery and symbol timing synchronization. Unfortunately, conventional PLLs can function correctly only when the frequency offset remains within a relatively small and limited range. This limited lock range is a direct consequence of the inability of the phase detector to resolve any phase error that lies outside a given 2π range. In this paper, we propose an extended lock range PLL that benefits from the flexibilities offered when the design is implemented in software. We describe and examine the FPGA implementation of a tracking system that uses the extended range PLL to perform carrier recovery. The proposed system is applicable to both QAM and PSK signaling and shown to function at a relatively fast clock frequency using a very small percentage of the resources on a Xilinx Virtex-IV FPGA.

1. INTRODUCTION

Software Defined Radios (SDR) are highly configurable hardware platforms that provide the technology for migrating many radio functions, traditionally performed in analog circuits, to DSP based implementations. While these implementations can be performed on a number of different silicon devices such as digital signal processors (DSPs) or general purpose processors (GPPs), field programmable gate arrays (FPGAs) present themselves as an attractive option for complex processing tasks for reasons of performance, power consumption and speed, [1].

Synchronization is one of the most important tasks that needs to be performed for successful operation of any data receiver. The receiver should find the exact frequency of the carrier that is modulated by the transmit data. If modulation is done coherently, the receiver should also synchronize with the phase of the carrier, [2]. There are many ways to implement carrier recovery and frequency synchronization in a digital communication system. At the heart of all the methods is the phase locked loop (PLL), [1].

The phase detector (PD) whose task is to find the phase difference between the reference input and the PLL output is the key element in the PLL. This phase detector - whether implemented as a EXOR phase detector, a JK-flipflop detector or a Hilbert Transform detector (to name only a few) - proves unable to resolve any phase error that lies outside a pre-specified 2π range. The phase-frequency detector (PFD) implementation extends this range to a limit of 4π, [3], but this wider range cannot be further extended.

The ambiguity imposed by the limited range of the phase detector output is the main reason for the limited lock range of the PLL. However, an N-fold increase in the lock range would be possible if one could extend the latter range to $-N\pi$ to $N\pi$. This is exactly what we strive to achieve in this paper. We propose a method to extend the lock range of the conventional PLLs by *unwrapping* the phase of the PD. The term *wrapping* refers to the fact that when a phase angle cycles through a range that goes beyond the pre-specified base range, it wraps back to a point within the range. For example, if the base range is $-\pi$ to $+\pi$, any phase angle outside this range is brought back to the range by adding or subtracting as many multiples of 2π as required to go back to $[-\pi, \pi]$. When the PLL is implemented in traditional analog circuits, the phase detector has no memory and thus automatically generates phase angles that are within a base range, [2].

To illustrate the operation of our extended-range PLL, we develop a complete carrier acquisition and tracking system that is applicable to both QAM and PSK modulated signals. This system takes the demodulated received signal as input and presents, through a series of multi-stage filtering a noise free sine wave to the PLL. The output of the PLL is interpolated and then multiplied by the input signal to compensate for the carrier offset.

The implementation of the system using a recent system level design tool from Xilinx Inc. called System Generator for DSP is presented. Design decisions such as usage of polyphase structures and *coordinate rotation digital computer* (CORDIC) arithmetic are made to optimize for space and clock rate.

We begin by introducing the theory governing our extended-range PLL in Section 2. Section 3 presents a detailed analysis of the PLL based carrier recovery system. We discuss the FPGA implementation of the system in Section 4 and, finally, draw our conclusions in Section 5.

2. PLL WITH EXTENDED LOCK RANGE

An all digital receiver is implemented using a conventional digital phase locked loop (CPLL) such as that shown in Figure 1.

The CPLL have three basic components: a phase detector (PD), a loop filter and a voltage controlled oscillator (VCO). The phase detector measures the difference between the phase of the local oscillator and the input carrier $\theta[n]$. This phase error, $\epsilon[n]$, is fed into the loop filter $L(z)$ whose specifications are chosen to track changes in both carrier

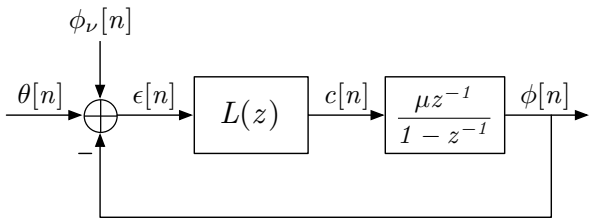

Fig. 1. Linear model of discrete PLL

frequency and phase offset. The output of $L(z)$, $c[n]$, controls the phase angle $\phi[n]$, output of the VCO.

The CPLL is kept in its lock range as long as $\epsilon[n]$ remains within the $[-\pi, \pi]$ range. This phase ambiguity caused by the PD limits the locking capabilities of the PLL. Herein, we propose a PLL with an extended lock range. We will refer to the new PLL as EPLL. The EPLL has the same building blocks as the CPLL above. The key element that differentiates the EPLL from the CPLL is the phase detector. To increase the lock range, the PD in the EPLL is equipped with a phase unwrapping mechanism.

A. Phase unwrapping mechanism

To illustrate the operation of the PD with the phase unwrapping mechanism, we first assume that the input to the EPLL is a complex-valued sinusoidal signal $\eta[n]$ of the form:

$$\eta[n] = ae^{j\theta[n]} + \psi[n], \tag{1}$$

where $\psi[n]$ is an additive noise, $\theta[n] = 2\pi\Delta f_c n + \theta_0$ and Δf_c is a carrier frequency offset.

As will be demonstrated in the next section, such complex sinusoidal signal can be obtained from the received signal through a demodulation process. The PLL strives to find and track the variations of $\theta[n]$.

The block diagram of the proposed EPLL is shown in Figure 2. The inputs to the PD are now the noisy complex-valued sine-wave $\eta[n]$, and the synthesized sine-wave $e^{j\phi[n]}$ fed back from the loop filter. The PD is designed to deliver an estimate of the phase difference $\epsilon[n] = \theta[n] - \phi[n]$.

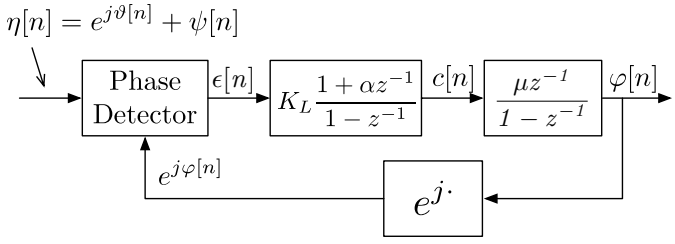

Fig. 2. PLL with extended lock range

The PD, usually implemented as an *arctan* function, wraps the phase estimate $\epsilon[n]$ to a pre-specified 2π range. We overcome this limitation by appending an unwrapping mechanism to the PD to reverse the effect of wrapping. To this end, we make the reasonable assumption that $\epsilon[n]$ varies slowly and proceed to examine the incremental values $|\epsilon[n] - \epsilon[n-1]|$. This ability of using memory to save $\epsilon[n-1]$ and thus track the signal variation is a key element in the successful operation of our EPLL.

The unwrapping mechanism performs the exact opposite function as wrapping. When a new phase estimate $\epsilon[n]$ is output from the PD, the unwrapper compares it to the previous estimate $\epsilon[n-1]$. The incremental value is then used to decide on whether wrapping has been done and consequently reverse its effect. For example, if $\epsilon[n] = \pi/4$ and $\epsilon[n-1] = 2\pi - \pi/8$, clearly, $\epsilon[n]$ has been wrapped, and the unwrapper will have to unwrap it by adding 2π, to obtain the unwrapped $\epsilon[n]$ equal to $2\pi + \pi/8$.

Keeping track of the incremental phase values and adding proper multiples of 2π to $\epsilon[n]$ thus allow us to unwrap the phase estimate of the PD and consequently increase the lock range of the PLL. The unwrapping mechanism is based on the simple update equation:

$$\epsilon[n] = \epsilon[n] + 2\pi \times \text{round}\left(\frac{\epsilon[n-1] - \epsilon[n]}{2\pi}\right) \tag{2}$$

This unwrapping scheme is very easy to implement in software defined PLLs and unwrapping can be done to any range $[-N\pi, N\pi]$ one finds suitable.

After unwrapping, the phase estimate is fed into the loop filter. The parameters of the loop filter depend on the design specifications and follow directly from the analysis of the CPLL, for details see [2] or [3].

Note that when the frequency offset is non-zero, the unwrapped output $\phi[n]$ may grow indefinitely. To avoid this problem, one needs to wrap $\phi[n]$ to $[-\pi, \pi]$ before feeding it back to the PD.

3. Carrier acquisition and Tracking system

To illustrate the operation of the EPLL, we present a case study of a non-data aided carrier acquisition and tracking system applicable to QAM and PSK modulated signals. The system, shown in Figure 3, takes the demodulated received signal $y[n]$ as input and outputs the carrier compensated signal $y'[n]$.

The system can be divided into three parts: (i) Front-end processing to remove the data dependency of the input signal, (ii) EPLL to track the variation of the phase/frequency offset, and (iii) Interpolator and phase compensator to produce the desired output.

A. Front-end processing

Ignoring the channel noise, the demodulated QAM input signal $y[n]$ finds the following form, [2]:

$$y[n] = (x_R[n] + jx_I[n])e^{j\theta[n]} \tag{3}$$

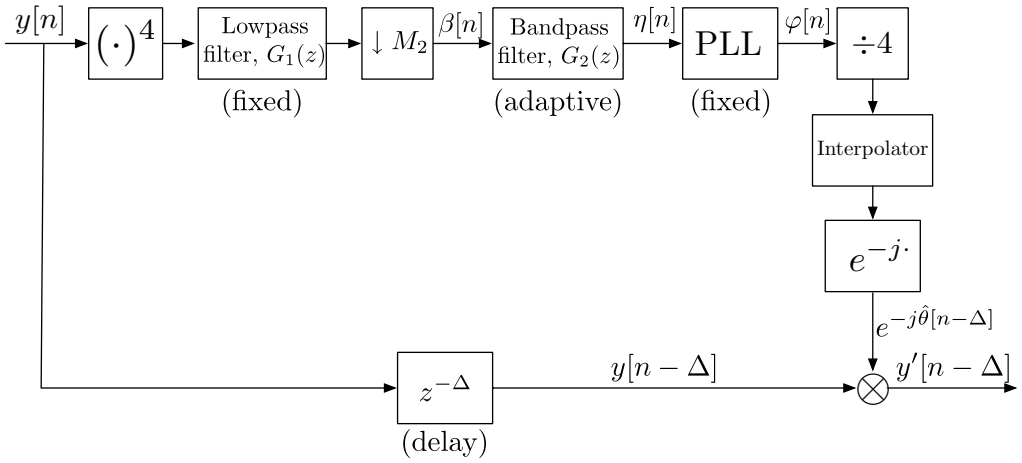

Fig. 3. Block diagram of a fine carrier acquisition and tracking procedure for QAM systems

where $x_R[n]$ and $x_I[n]$ are respectively the real and imaginary parts of the complex baseband signal, and $\theta[n]$ is the phase difference between the modulated received signal and a locally generated carrier at the receiver. Assuming a carrier frequency offset Δf_c and a phase offset θ_0, we get

$$\theta[n] = 2\pi\Delta f_c n + \theta_0. \tag{4}$$

The PLL has to estimate $\theta[n]$ and remove the undesirable factor $e^{j\theta[n]}$ from the right-hand side of (3).

The unknown complex factor $x_R[n] + jx_I[n]$ causes a phase ambiguity that does not allow direct estimation of $\theta[n]$ from $y[n]$. This problem is resolved by taking the fourth power of $y[n]$. The output of the $(\cdot)^4$ block in Figure 3 is obtained as, [2],

$$y^4[n] = m_{y4}e^{j4\theta[n]} + v[n]e^{j4\theta[n]} \tag{5}$$

where $m_{y4} = \text{avg}[x_R^4[n] + x_I^4[n] - 6x_R^2[n]x_I^2[n]]$ and $v[n]$ is a zero-mean wideband signal.

Recalling (4), equation (5) shows that $y^4[n]$ is the sum of a complex-valued sine-wave with frequency $4\Delta f_c$ and phase $4\theta_0$ - the parameters that the PLL strives to estimate - and a wideband signal, $v(t)e^{j4\theta[n]}$, that may be treated as noise.

Note that since for M-PSK signals, the fourth power rule cannot be applied, a power of M block is used instead. The rest of the system remains unaltered. Here, for the sake of demonstration, we assume the input is a QAM signal.

Direct application of the output of $(\cdot)^4$ block to the PLL does not lead to a satisfactory result, because proper operation of the EPLL requires a relatively clean sine-wave. One can obtain a cleaner (less noisy) version of the desired sine-wave, by passing $y^4[n]$ through a low-pass filter $G_1(z)$. After low-pass filtering, the result is decimated to save on the computational complexity.

Depending on the quality of the channel, the output of the low pass filter and decimator may still be noisy and thus could not be used as an input to the EPLL. To further *clean* the input signal to the PLL, one may use another stage of filtering: a bandpass adaptive filter $G_2(z)$ that is designed to automatically tune itself to the desired tone at $4\Delta f_c$.

The filter parameters that are used in the front-end processing should be chosen with special care. The passband edge of $G_1(z)$ is selected according to the frequency offset Δf_c. If the frequency offset is such that $|\Delta f_c| < \Delta f_{max}$, the passband edge f_{pb} is selected equal to $4\Delta f_{max}$.

The proper operation of the line enhancer $G_2(z)$ requires the input to be a narrow-band signal burried in a wide-band noise. In order to ensure the presence of the wide-band noise, the stop-band edge f_{sb} of the low-pass filter $G_1(z)$ and the decimation rate M_2 are chosen such that the signal spectra over the transition band and the aliased spectra add up to a flat spectra. This dictates the following relationship between f_{pb}, f_{sb} , M_2 and the input sampling rate f_{s1}.

$$\frac{f_{pb} + f_{sb}}{2} = \frac{f_{s1}}{2M_2} \tag{6}$$

Note that the adaptive line enhancer $G_2(z)$ that we use here is an FIR-LMS adaptive filter. The spectra of the signals before filtering, after $G_1(z)$ and after $G_2(z)$ are presented in Figure 4. One can clearly see the gradual enhancement in the quality of the signal that will eventually be fed into the EPLL.

B. EPLL

We follow the EPLL presented in Section 2. To be able to compensate for both carrier frequency and phase offset, we choose a second order PLL, [3]. The parameters of the loop filter of the EPLL are selected according to the sampling rate at the input of the phase detector, the noise bandwidth and the damping factor. These computations, as mentioned earlier, follow directly from the analysis of digital CPLLs,([2], [3]).

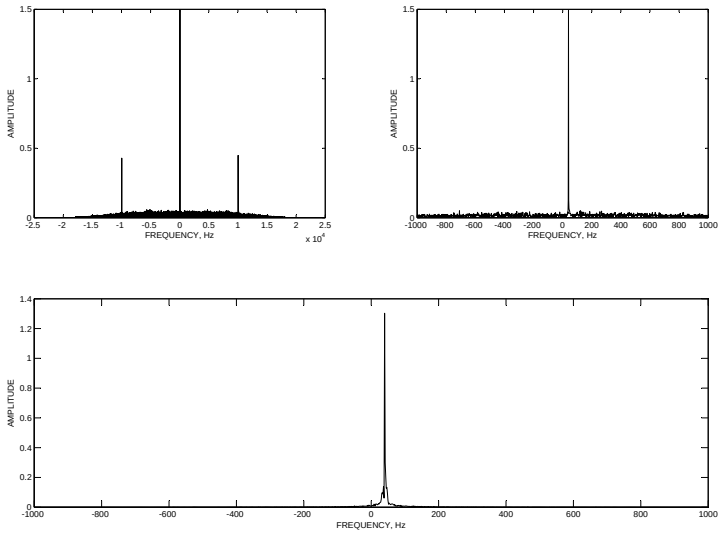

Fig. 4. The power spectrum of $y^4[n]$ (a) before filtering, (b) after low pass filtering and decimation, (c) after the line enhancer

The EPLL strives to track the time-varying phase angle $4\theta[n]$. Since the desired phase that should be removed from the demodulated signal is $\theta[n]$, the output of the PLL is divided by 4 before any further processing. This divide-by-4 block is shown in Figure 3. Note that the factor of 4 can alternatively be compensated for inside the EPLL. For this purpose, we include a factor of $\frac{1}{4}$ in the transfer function of the loop filter, to effectively generate a phase output equal to $\phi[n]/4$, and feed back $e^{j4\phi[n-1]}$ to the PD instead of $e^{j\phi[n-1]}$.

C. Interpolator and phase compensator

The output of the EPLL is an estimate of the M_2-fold decimated phase error in $y[n-\Delta]$ where Δ is the delay caused by the low-pass filter $G_1(z)$, the adaptive filter $G_2(z)$ and the EPLL. This delay is evaluated as $N_{g1}/2$ where N_{g1} is the length of $G_1(z)$. The line enhancer does not cause any delay, since its output is an estimate of the input. Similarly, the EPLL does not cause a delay, since upon convergence, the output $\phi[n]$ is equal to $\upsilon[n]$.

The samples of $\theta[n]$, output of the PLL ($\phi[n] = \theta[nM]$), need to be interpolated to the original sampling rate f_{s1}. There are many methods to carry out this interpolation. Here, we choose to use the simple linear interpolation method assuming that $\phi[n]$ is a piece-wise linear phase.

After interpolation, the samples of $\theta[n]$, now at the same rate as the demodulated signal $y[n]$, are converted into the phase compensation factor $e^{-j\theta[n]}$. This factor is finally multiplied by the delayed signal $y[n-\Delta]$ to yield the desired output $y'[n]$ of the carrier recovery system:

$$y'[n] = y[n-\Delta]e^{-j\theta}[n]. \tag{7}$$

4. FPGA IMPLEMENTATION

In this section, we present the hardware architecture of a 4-QAM carrier recovery system based on the EPLL structure. The System Generator implementation of the designed system is shown in Figure 5.

System Generator, an add-on to Simulink provided by Xilinx, produces a highly optimized FPGA realization since each module used in the architecture maps to an FPGA library component that has been carefully constructed and optimized for the FPGA target device. Moreover, the System Generator provides us with a visual representation of the system that not only serves as the design specification, but as the behavioral simulation model and the source definition for the hardware. The system Generator implementation also facilitates the rapid investigation of various design options in the system. The various parameters used in the data flow are specified as variables in the MATLAB Workspace, [1], [4].

The system was synthesized for a XILINX VIRTEX-IV XC4VSX35 FPGA. DSP48 slices were used to measure the resources utilization of the FPGA. Throughout the circuit, signals were represented using 16 bit vectors, 10-12 bits were used to represent the fractional part of the numbers depending on the growth incurred in each block as the signals propagate. The carrier recovery system occupies 3 out of 192 DSP48 slices available on the FPGA chip and can run at a maximum clock frequency of 20 MHz.

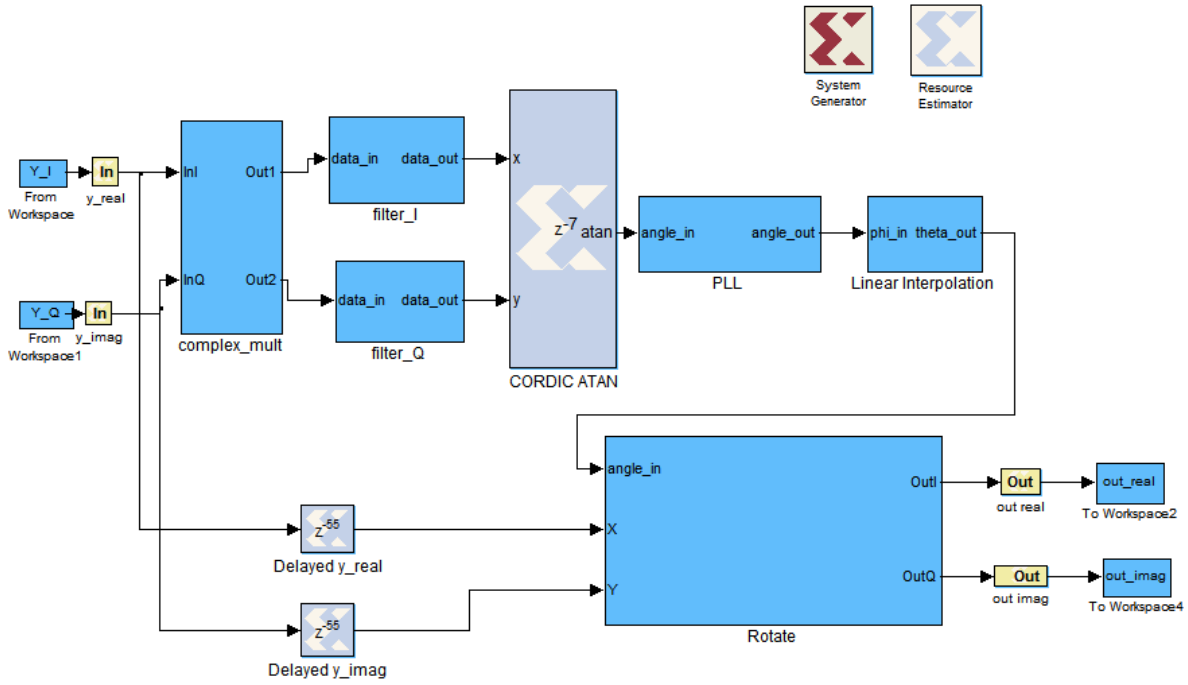

Fig. 5. The carrier acquisition system in system generator for DSP

A. *Front-end implementation*

The front-end processing begins by two stages of complex multiplication to raise the input signal to the power of four. Each complex multiplication block is implemented using adders/subtractors to save on the amount of resources used, [5].

For the low pass filter and the decimation, we use a polyphase filter structure to reduce the complexity of the implementation. Two polyphase low-pass filters are required here, one for each of the I and Q processing arms. These filters are 125-tap symmetrical FIR filters with 10 bits coefficients and support 16-bit precision input samples. The filters were generated using the Xilinx Core generator MAC-FIR block, [6]. The implementation of each filter occupied 1 DSP48 slice.

The line enhancer was not implemented for this particular presentation. This can be justified by the fact that we are implementing a QPSK carrier recovery system. If we were to implement a higher order QAM modulation, this adaptive LMS filter is required.

B. *EPLL implementation*

The extended lock range PLL implementation is shown in Figure 6. This implementation begins by extracting the phase of the filtered baseband signal through an *arctan*(I,Q) computation.

There are many options for computing arc-tangents, one alternative that is suited to an FPGA implementation is the *coordinate rotation digital computer* (CORDIC) arithmetic [7]. The CORDIC algorithm is highly suitable for FPGA designs because it consists of a series of additions and subtractions, these functions are very efficiently implemented by FPGA technology and require a small number of logic slices, [8].

The CORDIC algorithm converges only for input angles in the interval $[-\pi/2, \pi/2]$. In order to support the full range $[-\pi, \pi]$, a coarse angle rotation is first performed to map the input arguments into quadrant 1, the CORDIC algorithm micro-rotations are then performed and finally a quadrant correction is applied to account for the coarse angle rotation, [1].

The number of fine-angle iterations required to carry the arc-tan operation and the bit precisions of the CORDIC elements are investigated through simulation. The phase detector we used was implemented using 4 iterations and 16 bit signal precision.

To unwrap the output of the phase detector, the angle output of the CORDIC *arctan* block is compared to the output of the EPLL. This difference $\epsilon[n] = 4 \times \theta[n] - 4 \times \phi[n]$, along with the previous phase estimate $\epsilon[n-1]$ are fed to an *unwrapping* block where proper multiples of 2π are added to $\epsilon[n]$ depending on the incremental value $\epsilon[n-1] - \epsilon[n]$. The unwrapping block consists of multiplexers, adders and

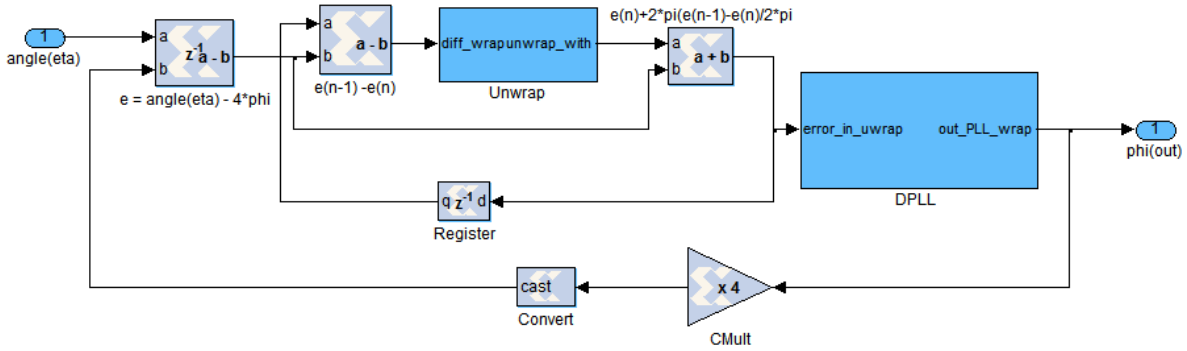

Fig. 6. The EPLL in system generator for DSP

subtractors. For the sake of illustration, we unwrapped the signal up to $[-10\pi, 10\pi]$.

After unwrapping, the signal is fed into the PI second order loop filter. This is implemented as two first order IIR filters. An IIR filter of one pole and one zero followed by an integrator.

After the loop filter, the phase is *wrapped* back to $[-\pi, \pi]$ before interpolation. This is performed by comparing each $\phi[n]$ with π if positive and by $-\pi$ if negative. If $|\phi[n]| > \pi$, 2π, respectively -2π, is added to the phase output. This wrapping mechanism is implemented using two multiplexers and four adders/subtractors.

C. post-PLL implementation

After extracting the required $\phi[n]$, one needs to interpolate the samples to recover the input sampling rate. We use linear interpolation. The slope of successive samples of the phase is computed and the intermediate samples are calculated accordingly.

Correct operation of the linear interpolation requires unwrapping the samples of the phase output of the PLL before computing the intermediate steps. This need to unwrap is clear from the fact that the phase $\phi[n]$ is assumed to be a piece-wise linear phase.

Linear interpolation is implemented using a number of adders/subtractors and multiplexers, an accumulator and an upsampler whose purpose is to add M_2 zeros after each input signal to allocate space to the intermediate samples being computed.

The interpolated phase samples are finally used to compensate for the input signal. The CORDIC algorithm in rotation mode is used here to compute the sine and cosine of the phase $\theta[n]$. The complex phasor $\cos(\theta) + j\sin(\theta)$ is then multiplied by the delayed input signal using a complex multiplication block similar to the one used to raise the input signal to the power of 2 (twice). This multiplication yields the final output $y'[n]$.

5. CONCLUSION

In this paper, we presented a method to increase the lock range of the conventional phase locked loop. This was made possible by the flexibilities offered when the system is implemented in software. To illustrate the functionality of the extended-lock range PLL (EPLL), a carrier recovery system was designed and implemented on a XILINX VIRTEX-IV FPGA. The implementation was done using System Generator for DSP and was optimized to occupy 3 DSP48 slices while running at a maximum clock frequency of 20 MHz.

6. REFERENCES

[1] C.DICK, F. HARRIS, and M. RICE, "Synchronization in software radios - carrier and timing recovery using fpgas," *In Proceedings of the IEEE symposium on Field-Programmable Custom Computing Machines*, 2000.

[2] B. Farhang-Boroujeny, *Software Radio*, 1st ed., September 2007 - Available at http://www2.elen.utah.edu/ farhang/.

[3] R. Best, *Phase-locked Loops: Design, Simulation and Applications*, 5th ed. McGraw-Hill, 2003.

[4] "System generator for dsp reference guide," Xilinx Inc., Tech. Rep., August 2007.

[5] "Complex multiplier v2.1," Xilinx Inc., Tech. Rep., April 2005.

[6] "Mac fir v5.1," Xilinx Inc., Tech. Rep., April 2005.

[7] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on Electronic Computers*, September 1959.

[8] "Cordic v3.0," Xilinx Inc., Tech. Rep., April 2005.