Synchronization in Software Radios - Carrier and Timing Recovery Using FPGAs

Chris Dick
Xilinx Inc., 2100 Logic Drive, San Jose, CA 95124, USA
chris.dick@xilinx.com

fred harris
College of Engineering, San Diego State University, San Diego
fred.harris@sdsu.edu

Michael Rice
Department of Electrical and Computer Engineering, Brigham Young University
mdr@ee.byu.edu

Abstract

Software defined radios (SDR) are highly configurable hardware platforms that provide the technology for realizing the rapidly expanding third (and future) generation digital wireless communication infrastructure. Many sophisticated signal processing tasks are performed in a SDR, including advanced compression algorithms, power control, channel estimation, equalization, forward error control and protocol management. While there is a plethora of silicon alternatives available for implementing the various functions in a SDR, field programmable gate arrays (FPGAs) are an attractive option for many of these tasks for reasons of performance, power consumption and configurability. Amongst the more complex tasks performed in a high data rate wireless system is synchronization. This paper is about carrier and timing synchronization in SDRs using FPGA based signal processors. We describe and examine a QPSK Costas loop for performing coherent demodulation, and report on the implications of an FPGA mechanization. Symbol timing recovery is addressed using a differential matched filter control system. A tutorial style approach is adopted to describe the operation of the timing recovery loop and considerations for FPGA implementation are outlined.

1 Introduction

The ever-increasing demand for mobile and portable communication requires high-performance systems employing advanced signal processing techniques to allow operation as close as possible to the Shannon information theoretic bound [2]. However, not only must these systems provide exceptional performance, but due to market and fiscal pressures, they must be flexible enough to allow the rapid tracking of evolving and fluid standards. Software defined radios (SDRs) are emerging as a viable solution for meeting the conflicting demands in this arena. SDRs support multimode and multiband modes of operation to allow service providers an economic means of future-proofing these increasingly complex and costly systems.

During the last decade or so, radio system functionality has migrated from analog to digital implementations. We have observed, and continue to observe, the migration of the digital portion of a receiver along the signal conditioning chain, moving ever closer to the antenna. The implementation of these high-performance digital communication systems has been made possible by advances in semiconductor process technology, that has allowed the concept of *system on a chip* to become a reality. In a communication environment the hardware platform must execute sophisticated source coding algorithms, modulation, demodulation, power control, channel coding, multiple access (TDMA, FDMA, CDMA) schemes and many levels of synchronization, starting at the physical layer, and moving up through the open system interconnection (OSI) protocol stack.

The communication systems engineer has had a vast range of semiconductor technologies to choose from when developing such a system. These have included application specific standard parts (ASSPs), full custom silicon, instruction set based digital signal processors (DSPs) and high performance general purpose processors (GPP). In a current generation system, the hardware solution is often best provided

0-7695-0871-5/00 $10.00

using a heterogenous computing approach, using the appropriate type of silicon for each particular function. In the early 1990's field programmable gate arrays (FPGAs) also played a role in digital communication hardware, often being used for glue logic, bus interfacing, complex state machines and memory controllers. However, in recent years, FPGA technology has undergone revolutionary changes. The gate densities and clock speeds of recent generation FPGAs provide the communication system architect with a highly configurable logic fabric that can be used for realizing sophisticated real-time signal processing functions.

One of the challenging tasks in a communication system is carrier and symbol timing recovery. A large amount of time is spent solving these problems, and frequently a large amount of hardware and software in a SDR is dedicated to synchronization [1].

This paper examines techniques for developing, modeling and generating FPGA implementations of carrier and timing loops. The paper is organzied as follows. Section 2 provides an overview of synchronization in a digital communication system and highlights its importance in a coherent communication environment. Section 3 provides a brief review of phase locked loop (PLL) theory before describing a Costas loop for performing carrier recovery using quadrature phase shift keying (QPSK) modulation. The development of the FPGA design using Matlab [3] and Simulink [4] is outlined. The FPGA logic resource requirements and performance is reported. In Section 4 a technique suitable for implementing timing recovery in an FPGA is described. The method is based on a differential matched filter approach. Finally, in Section 5 we draw our conclusions.

2 Synchronization in Software Radios

At the simplest level, synchronization is the process of aligning the frequency and phase of a set of remote oscillators. This alignment occurs at two distinct levels. One is at the waveform level and is closely related to the physical layer of the communication process. The second is at the bit stream level and is more closely related to data representation at higher levels in the communication process. At the waveform level, synchronization entails oscillator frequency and phase alignment for waveform timing, for carrier acquisition, and chip alignment and hopping boundaries for spread spectrum modulation overlays. At the bit level, synchronization entails frequency and phase alignment of logical boundaries such as bits, words, frames and network event boundaries. Our discussion addresses the physical layer synchronization of waveforms

Henry David Thoreau stated, "If a man does not keep pace with his companions, perhaps it is because he hears a different drummer. Let him step to the music he hears, however measured or far away". This commentary on social tolerance runs counter to the needs of a synchronous communication system. In a modern digital communication system every user must march to the sound of the same drummer.

This need to align clocks in the communication process is related to how the physical layer moves data through the channel. Figure 1 presents a model of a conventional modulator and demodulator. Bits are delivered to the modulator at a specified transfer rate of, say 100 kbits/sec. The bit field is parsed into nibbles of widths typically 2, 3, or 4 bits. For the rest of this discussion we will use the 4-bit wide nibble. The 4-bit nibble is used to identify one of 16 (pre-selected) base-band waveforms to be placed on a carrier and delivered by the modulator to the channel. A DSP based way to accomplish this is to use the nibble as an address to a table that outputs four possible amplitudes for I(n) [3, 1] and for Q(n) [3, 1] of the wave shapes to be carried by the cosine (in phase) and by the sine (quadrature phase) carrier waveforms. Remember that since the sine and cosine of a carrier frequency are orthogonal (i.e. the average value of their product is zero), we can think of the sine and cosine as being two independent carriers in the same frequency band. When the cosine and the sine waveforms are each assigned the task of delivering one of 4-possible amplitudes we call the process 16-Quadrature Amplitude Modulation or 16-QAM.

The waveforms must be generated at the symbol (or nibble) rate of the modulator, which for this example is 25 kSymbols/sec. The shape of the waveform is chosen to minimize the bandwidth required to carry the waveform as well as to satisfy a constraint related to acceptable out-of-band spectral levels. The duration of the waveform extends over many symbol durations, typically on the order of 16 to 24 symbols, consequently, the waveforms are overlapped and the modulated waveform is the summation of a set of time shifted and scaled waveforms. In most implementations, the modulator constructs samples of the base-band waveform with digital filters that respond to the amplitudes delivered by the look-up table in response to each input nibble. The filter performs the simultaneous tasks of shaping the waveform, of merging the weighted overlapped symbols, and of raising the sample rate of the time series from input symbol

rate to output sample rate. The increase in sample rate is invoked so that the up-conversion or placement of the waveforms on a carrier can be accomplished by multiplying the waveforms samples with samples of a sine and cosine carrier in the DSP domain. For this example, let us up-sample to 50 MHz and then up-convert to the range 1-to-20 MHz. The samples of the heterodyned signal are converted to a waveform by a digital to analog converter (DAC) and low pass filter and most often up-converted again by additional analog processing.

The receiver's task is to invert the process just described. The signal is first subjected to an analog down conversion to the range 1-to-20 MHz, is sampled at 50 MHz and converted to digital data with an analog to digital converter (ADC). In the DSP based receiver the sampled signal is down converted from its digital carrier by samples of the sine and cosine carrier formed by the output of a local sine wave generator and then filtered by product-sums matched to the known transmitted wave shapes. The filters multiply the base-band input samples by a template replica and accumulate the product terms to gather all the energy of the transmitted signal as a single observable parameter, This parameter is delivered to a decision device that estimates the amplitude of the transmitted signal from the processed output of the filter.

We have been very cavalier in our description of the receiver's task. In order for the receiver to reverse the up-conversion operation by a matching down-conversion it has to use a phase coherent replica of the sinusoid delivering the waveform to the demodulator. In addition, when we perform the matched filtering operation, we have to align the template with the boundaries of the received waveform so we know when to start the product-sum and when to finish it. The first task of forming a frequency and phase-matched replica of the local oscillator for the down conversion operation is called carrier acquisition. The second task of collecting a set of input samples time aligned with the replica template is called (symbol) timing recovery. The demodulator must perform a task not performed at the modulator; by mining clues embedded in the received signal, it must replicate the carrier and symbol clocks associated with the signal in order to process the received signal. This differs considerably from local systems such as state machines or FPGAs with a distributed clock, or from a distributed system with shared clock (plesiosynchronous) in which the timing is available from a separate parallel distribution system. There the concern of aligning the clock and the data is a matter of differential delay or skew. In the communication case we must align totally asynchronous oscillators and clocks. When we teach this material in an undergraduate course, we attribute this task to a friendly genie (called PLL) and claim this material is beyond the scope of the course. When we build the equipment we have to don our genie suits.

3 Carrier Recovery

There are many options for implementing carrier phase and frequency synchronization in a digital communication system. At the heart of all synchronizers is the phase-locked loop (PLL).

3.1 Phase Locked loops

The generic PLL is shown in Figure 2. PLLs are

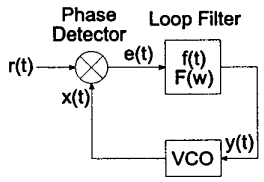

Figure 2: Basic phase locked loop.

servo control mechanisms whose controlled parameter is the phase of a locally generated replica of the incoming carrier wave. Phase locked loops have three basic components: a phase detector, voltage controlled oscillator (VCO) and a loop filter. The phase detector measures the difference between phase of the local oscillator and the input carrier. This signal is fed to a loop filter that governs the response of the PLL to variations in the error signal. The Loop filter is designed to track changes in the error signal, but not be overly responsive to receiver noise. The loop filter determines the type of disturbances the PLL can track, for example, a phase or frequency step. A detailed description of PLL operation can be found in [2]

In an all-digital receiver a digital phase-locked loop (DPLL) like that shown in Figure 3 is required. This DPLL employs a second order infinite impulse response (IIR) loop filter. The two filter coefficients k_i and k_p control the filter corner frequency and damping ratio. In the digital implementation, the VCO in Figure 2 is replaced with a direct digital synthesizer (DDS). The phase detector is implemented using the arc-tan functionel unit in the figure.

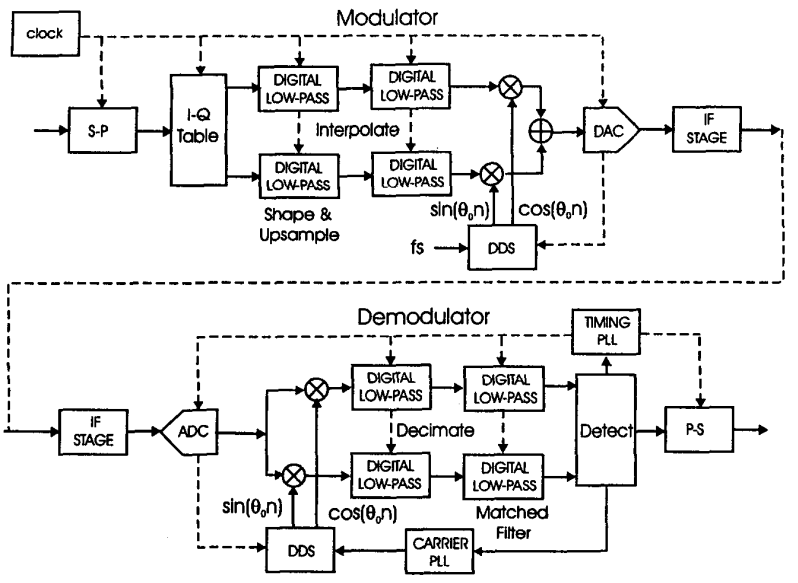

Figure 1: Modulator and demodulator for QAM and QPSK: Note asymmetry: PLLs at demodulator

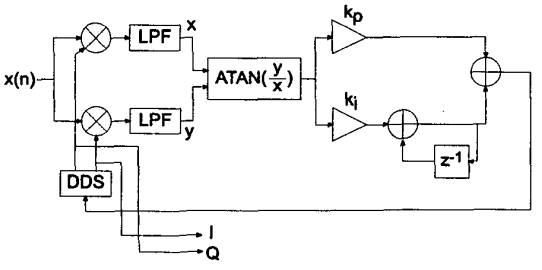

Figure 3: Digital phase-locked loop

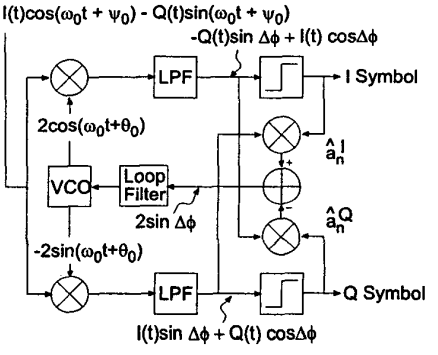

Figure 4: QPSK Costas loop. The carrier phase is obtained and the symbols detected in the one loop. $\Delta\phi = \psi_0 - \theta_0$.

3.2 QPSK Costas Loop

Communication systems employing QPSK modulation are very common. The basic Costas loop [2] can be enhanced to perform carrier recovery and symbol detection for a QPSK modulation scheme as shown in Figure 4.

To understand the operation of this loop, consider the scenario when the loop is reasonably near lock. The signal on the I processing arm (or *rail*) after the LPF is close to the data symbol value a_n^I and the signal on the quadrature arm is close to a_n^Q. The slicer enforces this by ignoring small perturbations in the signal, which could be due to the opposite-rail symbol if the loop is not locked, or the shaping of the pulse, or simply channel noise. The ± 1 symbol decisions feed a network that produces from the received baseband signals a phase difference signal $2\sin(\psi_0 - \theta_0)$. This signal, working with the loop filter and the VCO, operate like the basic PLL shown in Figure 2.

3.3 QPSK Costas Loop Implementation

To produce a fixed-point arithmetic realization of the QPSK Costas loop in Figure 4 a combination of Matlab [3] and Simulink [4] where employed. After the quantized model was verified in the Simulink domain, a conventional FPGA implementation flow us-

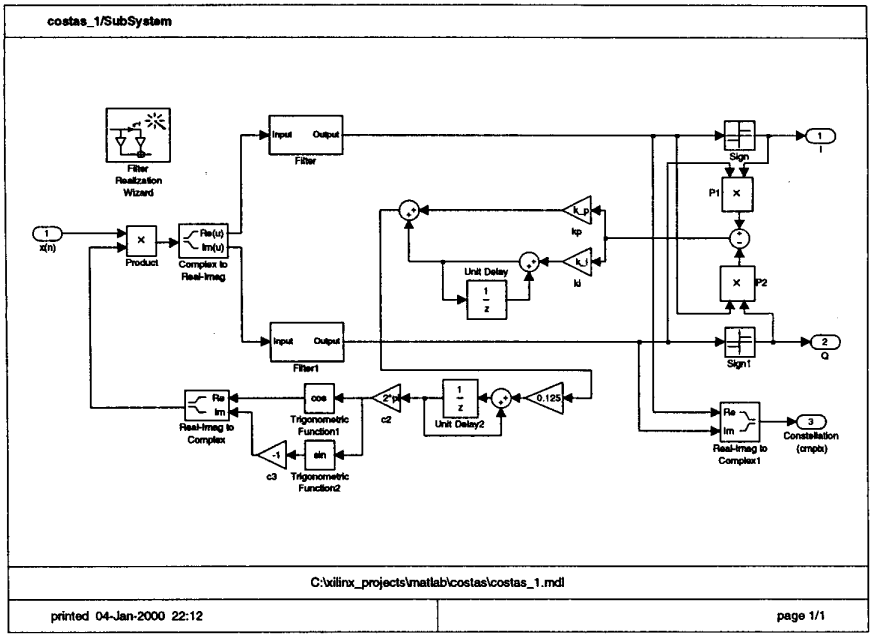

Figure 5: Simulink model of a digital QPSK Costas loop - floating-point arithmetic implementation.

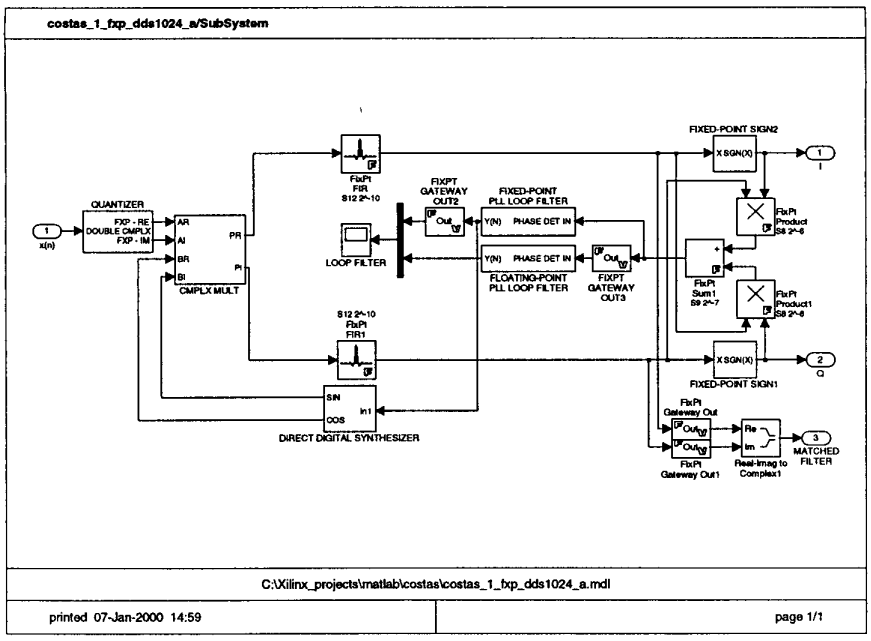

Figure 6: Simulink model of a digital QPSK Costas loop - fixed-point implementation.

ing VHDL and the Xilinx Core Generator are used to produce the final design.

Figure 5 shows the floating-point arithmetic simulink model while Figure 6 illustrates the fixed-point arithmetic, or *quantized*, model.

To verify the operation of the loop a system level design was developed that modeled a simple transmitter and channel that simulated a Doppler shift of the transmitter carrier wave. In practice, the Doppler shift is associated with movement between the transmit and receive platforms, as might be the case with a cellular handset user traveling in a car.

The transmitter generated a pseudo random complex sequence that was shaped by a multirate transmit filter with an excess bandwidth $\alpha = 0.25$ and an interpolation factor of 1-to-8. The channel model introduced a small frequency translation of the carrier. Therefore, the receiver was presented with a signal that had a frequency and phase offset compared to the nominal local oscillator. The purpose of the Costas loop is to track the frequency and phase offset to allow coherent demodulation of the transmitted waveform. The sequence of plots in Figure 7 provide some insight to the operation of the carrier recovery loop. Figure 7(a) shows the QPSK constellation diagram after the matched filter. Figure 7(b) is the corresponding eye diagram. The eye is clearly open and, in the absence of any channel impairments, the receiver can easily make correct symbol decisions using this waveform. The frequency translation applied to the transmitted signal causes the constellation to rotate as shown in Figure 7(c). The receiver eye diagram shown in Figure 7(d) clearly shows the eye is closed, indicating that valid symbols decisions cannot be made. The Doppler shift modeled in this experiment causes a frequency translation of the carrier. The corresponding phase slope is linear, with a gradient that corresponds to the magnitude of the frequency offset. One way to observe and quantify the performance of the carrier tracking loop is to monitor the phase function of the interfering signal and that of the oscillator in the Costas loop. This is shown in Figures 7(e) and 7(f). We observe that the loop attains lock after a few hundred samples. The difference between the two phase functions, or phase error, is presented in Figure 7(g). Finally the de-rotated constellation is shown in Figure 7(h).

The quantized model was developed using the Simulink fixed-point blockset. This approach allowed a high degree of design compression. After the Simulink floating-point model was completed and verified, approximately 30 minutes was required to gen-

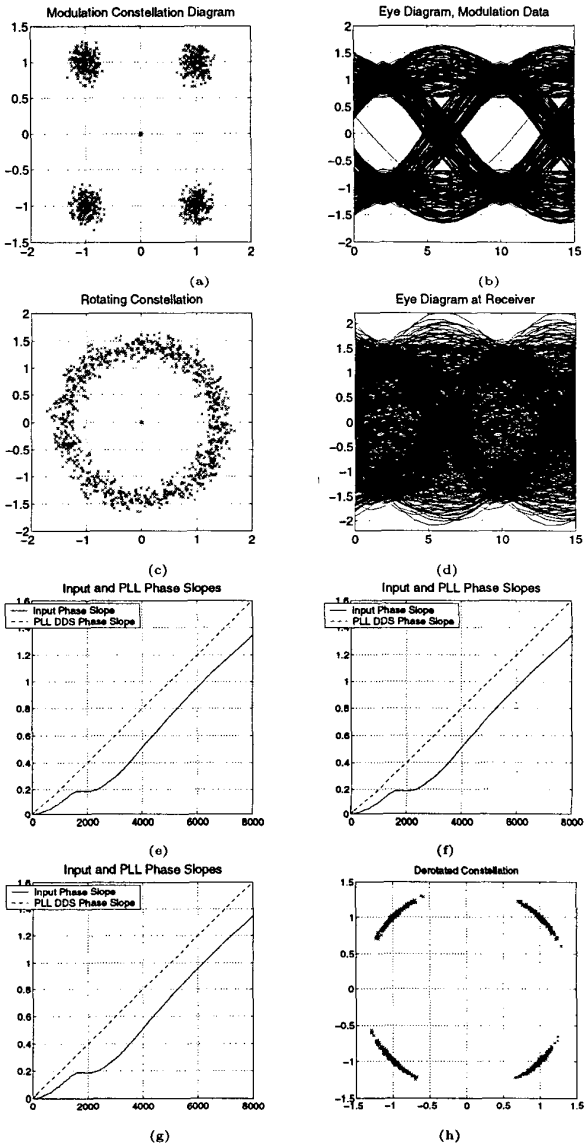

Figure 7: (a) QPSK modulation constellation. (b) Eye diagram - transmitter. (c) Rotating constellation due to frequency offset. (d) Eye diagram - receiver. (e) Input and output phase slopes. (f) Input and output phase slopes - exploded view. (g) Phase error. (h) De-rotated constellation.

erate the quantized solution. This approach also provided a simple mechanism to compare corresponding values in the floating-point and fixed-point models. For example, in Figure 6, a floating-point and fixed-point loop filter are operating in parallel. The two filter outputs are overlayed on a time-series plot - the *scope* token in the figure. This permits a simple method of observing the fixed-point system and evaluating its performance in the context of the ideal floating-point filter realization. The precision of the components in the loop filter can be interactively adjusted in real-time as the model is running. The conventional Matlab environment was also invaluable for performing detailed analysis of data exported to the workspace from the Simulink model.

3.4 FPGA Implementation

Several functional units are required to implement the carrier recovery loop. A complex heterodyne is employed to down-convert the input signal. This is of course recognized as a complex multiplier. There are two matched filters, one for each of the I and Q arms. The phase detector is straightforward, consisting of two 1-bit slicers (sign detector), two 2's complementers and a subtractor. The second-order loop filter is realized using two multipliers, an integrator and an adder. The local replica of the carrier wave is generated by a DDS.

The complex multiplier is implemented using 4 multiplications and two additions. The ADC samples are represented using 8-bits, while the heterodyning signal employs 12-bit samples. Each 8×12 multiplier occupies approximately 81 logic slices. The complete multiplier occupies 344 slices. the recursive nature of the Costas loop demands the use of purely combinatorial multipliers and adders.

Two matched filters are required. One for each of the I and Q processing arms. These filters are 97-tap symmetrical FIR filters with 12-bit coefficients and support 9-bit precision input samples. The filters were generated using the Xilinx Core Generator [6] filter compiler. The implementation employs serial distributed arithmetic. Taking advantage of the symmetrical coefficient data, each filter occupies 248 Virtex FPGA [5] logic slices. Using 9-bit precision input samples, the filter requires 10 clock cycles to compute a new output. The bit-clock for the filter is a function of the FPGA speed grade. Typical values are between 100 and 150 MHz. This translates to a sample throughput of 10 to 15 MSamples/sec.

The multipliers in the loop filter occupy most of the logic resources for this sub-system. The coefficient parameters are represented using 16-bits while the input samples are carried with 8-bit precision. The complete loop filter occupies 80 slices.

The DDS was implemented using a simple phase truncation architecture [7]. Using the quantized Simulink model, a 1024-point sin/cos look-up table (LUT) with 12-bit precision samples was found to be adequate for the application. Using quarter wave symmetry [7], the sin/cos LUT requires only a single Virtex block RAM (BRAM) [5]. The dual-port nature of the BRAM permits both the I and Q samples to be computed simultaneously. The DDS phase accumulator consists of a 28-bit adder and register. These components occupy a modest 14 logic slices.

The complete QPSK Costas loop occupies approximately 1000 logic slices.

4 Timing Synchronization

Symbol decisions are based on the matched filter output at the end of each symbol period. The detector samples the matched filter output at that time and uses this information to decide which symbol was most likely to have been sent. In order to make these decisions, the detector must know when the symbols begin and end — this is called *timing synchronization.* The best way to illustrate timing synchronization is with an eye-diagram such as the one for I channel of QPSK illustrated in Figure 8. The eye diagram is produced by plotting segments of successive matched filter outputs on top of each other. The end of the symbol interval corresponds to the point in time where the eye diagram is the most open as shown.

There are three basic methods to determine the optimum sampling point.

1. The first method finds the point where the slope of the matched filter output is zero and is illustrated in Figure 8 (a). If the current timing estimate is too early, then the slope of the matched filter output is positive indicating that the timing phase should be advanced. If the current timing estimate is too late, then the slope of the matched filter output is negative indicating that the timing phase should be retarded. This method implements maximum likelihood timing synchronization.

2. The second method uses the zero crossings in the matched filter output to estimate the times in between the optimum sampling points as shown in Figure 8. Zero crossings are found by searching

for sign changes between matched filter outputs $s(n-1)$ and $s(n+1)$. A sign change means $s(n)$ resides on a zero crossing trajectory. Positive going zero crossings $s(n)$ are added to an averager while negative going zero crossings are subtracted. This method is sometimes called a Gardner loop.

3. The third method estimates the optimum sampling point by finding the position that minimizes the variance in the matched filter output. Typically, the search is performed by estimating the variance at the next interpolation point. If the variance increases, then the timing estimate is not advanced. If the timing decreases, then the timing estimate is advanced. This method is also called a dither loop.

A block diagram of the maximum likelihood timing synchronizer is illustrated in Figure 9. The output of the matched filter produces the points on the eye diagram and the derivative matched filter produces the slope at each of the points. The slope of the matched filter is multiplied by the sign of the matched filter output[1] to produce an error signal which is averaged (i.e. filtered) and input to a trigger generator. The trigger generator outputs a trigger signal which downsamples the matched filter output. The downsample rate usually such that matched filter outputs are sampled once or twice per symbol. Fractionally spaced equalizers require two samples/symbol. Symbol spaced equalizers or unequalized systems require one sample/symbol. In what follows, it is assumed that the system requires 2 matched filter samples/symbol since systems that only require 1 matched filter sample/symbol can simply discard every other matched filter output[2].

The matched filter is a digital filter which processes a signal that has been sampled by an asynchronous clock usually running at twice the symbol rate. Since it is unlikely that the optimum sampling point corresponds to one of the two samples per symbol period, interpolation is used to to generate data samples in between these sample points. This concept can be illustrated using Figure 9. Imagine the data, sampled at 2 samples/symbol is upsampled by a factor M. (M is can be as low as 8 for QPSK or as high as 128 for 256 QAM.) Upsampled impulse responses of the matched filter and derivative matched filter are used to process this data. Every M-th point (appropriately phased) is then available to the rest of the system. Now the purpose of the trigger generator is clear – it indicates which of the matched filter outputs is to be used for equalization and data symbol decisions.

A polyphase partition is the most efficient way to perform the interpolation. To implement the polyphase matched filter and derivative matched filter, the matched filter and derivative matched filters of Figure 9 are replaced by the polyphase partitions of the upsampled impulse responses of the two filters as illustrated in Figure 10. The length of each polyphase filter partition depends on the excess bandwidth and the required out-of-band attenuation. For example, using a square-root raised cosine filter, the impulse response of a filter with 40% excess bandwidth is about 10 symbols long. At a sampling rate of 2 samples/symbol, the filter is 20 samples long and at a sampling rate of 16 samples/symbol, the filter is 160 samples long. This represents an 8-to-1 upsampling. Continuing with this example, the polyphase partition consists of a bank of 8 filters, each with 20 taps and operating at 2 samples/symbols. Each of the filters outputs data at 2 samples/symbol but with a different phasing on the interpolation (hence the name "polyphase").

The trigger generator of Figure 9 is replaced by a subsystem that outputs the index associated with the proper phase of the matched filter output corresponding to the maximum eye opening as illustrated in Figure 11. The top counter is a modulo-1 counter with free running period equal to the one-half the symbol interval (remember — 2 matched filter outputs/symbol). The counter increment is altered by the output of the loop filter so that the roll-over period is either increased (when the timing needs to be advanced) or decreased (when the timing needs to be retarded). Thus an accumulator register and adder are required. The second counter cycles through the polyphase filter indices in synchronism with the top counter. Whenever the top counter rolls over, the accumulator of the lower counter contains the index of the desired polyphase filter. This counter simple cycles through the integer indices of the polyphase partition (in the example it needs to count 0,1,...,7,0,1,... and so requires 3 bits).

The performance of the receiver is tied directly to the performance quantization error of the matched filter outputs and the accuracy of the timing estimate. In a well designed system, the quantization noise introduced by the matched filter should be less than dynamic range required to achieve the desired performance. The quantization noise in the filter is reduced

[1]The sign of the matched filter is used to give the proper sign on the slope for negative matched filter outputs.

[2]Of course, care must be taken to throw away the right sample. The sample corresponding to the maximum eye opening should be kept, while the other sample is discarded.

by 5 dB for each additional bit of precision in the filter coefficients. For example, QPSK requires approximately 11 dB E_b/N_0 to achieve a bit error probability of 10^{-6} so that this system only needs 3 bit matched filter coefficients. 16-QAM on the other hand, requires 30 dB E_b/N_0 to achieve a bit error probability of 10^{-6} so that 6 bits are required in the matched filter in this system. An S-stage FIR matched filter with b-bit data at the input with c-bit coefficients performs S multiplies and outputs the sum of the products. At most, the filter output grows to $b+c+\log_2 S$ bits. However, the values of a square-root raised cosine FIR filter are such that $b+c+2$-bits is an upper bound on the accumulator size.

An FPGA implementation of this system requires resources for two polyphase FIR filters, a possible sign change for the multiplier, an IIR filter with one pole and one zero for the loop filter $G(z)$, and two counters to track the phase index and the timing of the phase index. The polyphase filter requires the hardware for one S-tap FIR filter and memory for M sets of S-coefficients. Since each tap/coefficient pair requires a multiply, the hardware requirement for the polyphase matched filter implementation is S-stage shift register (b bits/register), S multiplies ($b+c$ bits/multiply), and one add requiring a $b+c+2$ bit accumulator. The hardware and memory requirements for the derivative matched filter are identical.

Usually, the form of the loop filter is identical to that used in the carrier recovery loop described in the previous section. Except for the coefficient values, the FPGA implementation of this filter is identical to the loop filter used in the carrier recovery loop. The loop filter operates at the upsampled rate of $2M$ samples/symbol (16 samples/symbol in the example).

The key design parameter for the index computer is the size of the accumulator in the top counter. The accumulator needs to be large enough to accommodate the timing resolution and fine tuning of the phase of the overflow. The timing resolution is the overflow interval (1.0) divided by the count and is 1/8 in the example so that a 3-bit accumulator is the smallest accumulator that can be used. Increasing the size of the accumulator allows increased resolution in the phasing of the overflow without causing excessive jitter in the timing estimate.

5 Conclusion

The continuing evolution of communication standards and competitive pressure in the market place dictate that communication system architects must

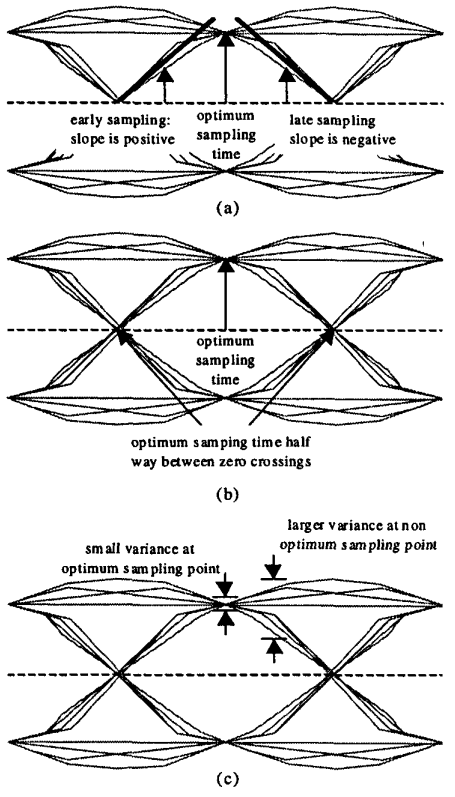

Figure 8: Eye diagram for the QPSK I channel and their relationship to common timing synchronization techniques: (a) Maximum Likelihood estimation using the slope of the eye at the current estimate, (b)Zero-crossing detector or Gardner loop, (c)Minimum-variance of dither loop.

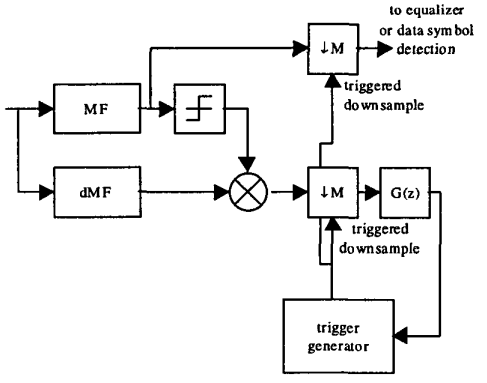

Figure 9: Block diagram of an approximation to the maximum likelihood symbol timing estimator.

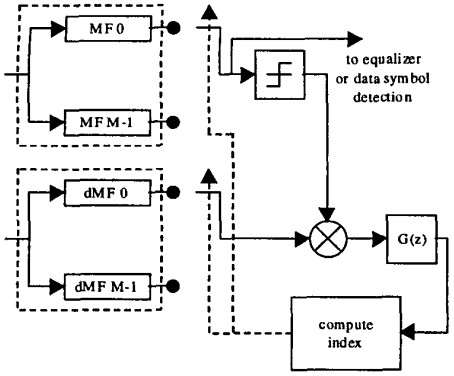

Figure 10: A polyphase partition of the matched filter and derivative matched filter that implements the interpolation needed to enhance the resolution of the timing estimate.

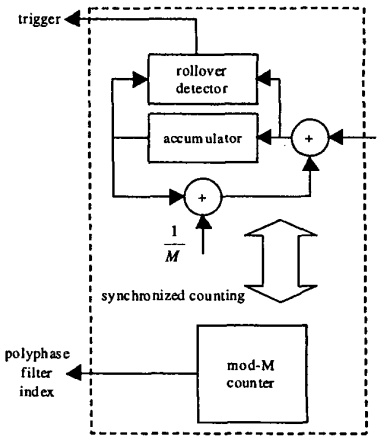

Figure 11: The counters used to trigger the sampling of the polyphase matched filter and to track the polyphase index (i.e. the interpolation phase).

start the engineering design and development cycle while standards are still in a fluid state. Third and future generation communication infrastructure must support multiple modulation formats and air interface standards. FPGAs provide the flexibility to achieve this goal, while simultaneously providing high levels of performance. The SDR implementation of traditionally analog and digital hardware functions opens-up new levels of service quality, channel access flexibility and cost efficiency.

The software in a SDR defines the system personality, but currently, the implementation is often a mix of analog hardware, ASICs, FPGAs and DSP software. The rapid uptake of state-of-the-art semiconductor process technology by FPGA manufacturers is opening-up new opportunities for the effective insertion of FPGAs in the SDR signal conditioning chain. Functions frequently performed by ASICs and DSP processors can now be done by configurable logic. This paper has provided an overview of how carrier and timing synchronization can be implemented in an FPGA. While the QPSK Costas loop and differential matched filter servo loops described in the paper form the key components of a QPSK modulation system, a number of additions and refinements (for example carrier acquisition) are required to produce a complete system. These aspects are the basis of on-going research.

References

[1] H. Meyr, M. Moeneclaey and S. A. Fechtal, *Digital Communication Receivers*, John Wiley & Sons Inc., New York, 1998.

[2] B. Sklar, *Digital Communications Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[3] The Mathworks Inc, *Matlab, Getting Started with Matlab*, Natick, Massachusetts, U.S.A, 1999.

[4] The Mathworks Inc, *Simulink, Dynamic System Simulation for Matlab, Using Simulink*, Natick, Massachusetts, U.S.A, 1999.

[5] Xilinx Inc., *The Programmable Logic Data Book*, 1999.

[6] Xilinx Core Generator System, http://www.xilinx.com/products/logicore/coregen/index.htm

[7] C. H. Dick and f. j. harris, "Direct Digital Synthesis - Some Options for FPGA Implementation", *SPIE International Symposium On Voice Video and Data Communication: Reconfigurable Technology: FPGAs for Computing and Applications* Stream, Boston, MA, USA, pp. 2-10, September 20-21 1999.