Software-defined Radio Using Xilinx

Anton S. Rodriguez, Michael C. Mensinger, Jr.

Advisors: Dr. In Soo Ahn and Dr. Yufeng Lu

Department of Electrical and Computer Engineering,
Bradley University, Peoria IL 61625

Abstract

A software-defined radio (SDR) allows for digital communication systems to easily adopt more sophisticated coding and modulation technologies, which is extremely important in meeting the ever-increasing demands of the wireless communication industry. An SDR has been constructed, using the Simulink tool, and implemented on the Virtex-II Field Programmable Gate Array (FPGA) development kit. In Simulink, the Xilinx system generator block set allows for easy fixed-point simulation, system testing, and implementation on hardware. The modulation scheme used in the system is Quadrature Phase-Shift Keying (QPSK). During transmission, the QPSK signal experiences frequency and phase shifts and the signal constellation rotates after coarse carrier synchronization at the receiver. A phase-locked loop (PLL) circuit is designed to lock onto the QPSK constellation and allow recovery of data from the received signal. However, the QPSK signal locking is still subject to phase ambiguity. To resolve the phase ambiguity, a differential coding scheme is also implemented.

Acknowledgments

Dr. In Soo Ahn

Dr. Yufeng Lu

Department of Electrical and Computer Engineering
Bradley University, Peoria IL 61625

Table of Contents

List of Figures

I. INTRODUCTION

SOFTWARE-DEFINED radios (SDR) provide a versatile wireless communication solution for a wide range of applications, including cellular telephones, global positioning systems, and military grade communications. The SDR is applicable in nearly any wireless communication system and when implemented on a Field Programmable Gate Array (FPGA), the true advantages of this hybrid system come to life.

The SDR is a very cost-effective system in many ways. Since all hardware is physically programmed using software, re-design becomes relatively simple. Rather than discarding old hardware, the SDR is simply reprogrammed, updated and loaded back onto the FPGA, saving both time and money. The SDR also provides a capability for high quality communication without a need for expensive broadcasting equipment.

In addition to its cost benefits, the SDR is also a very powerful and flexible system. In wireless communication, this means faster data rates and highly configurable modulation technology. For these reasons, the SDR is a rapidly emerging methodology in digital communication system design and implementation.

Quadrature phase shift keying (QPSK) is a modulation scheme which sends a pair of bits per symbol, increasing data rate by a factor of two. Other modulation schemes such as 8PSK and 16PSK increase data rate even further, sending triplets and quadruplets of bits per symbol. A typical problem in QPSK and in wireless communication is carrier synchronization, or the synchronization of the oscillator at the receiver with the oscillator at the transmitter [1]. In order to do so, a phase-locked loop circuit must be appended to the receiver [2]. This provides the local oscillator at the receiver with a frequency adjustment. However, once this correction is made, a static phase error called phase ambiguity will still exist. In QPSK systems [3], this phase ambiguity will be any multiple of 90 degrees. In order to properly decode transmitted data, a differential coding scheme is implemented which corrects any phase ambiguity.

II. BACKGROUND

QPSK Theory QPSK (or 4PSK) is a modulation scheme that encodes data based on the phase angle between two waveforms. The two waveforms are represented as an in-phase component, I(t), and an out-of-phase component, Q(t). This information is modulated with a carrier signal and transmitted. The mathematical representation of the transmitted signal, $s(t)$, is

$$s(t) = I(t)\cos(2\pi f_o t) - Q(t)\sin(2\pi f_o t) = \cos\big(2\pi f_o t + \phi(t)\big) \qquad (1)$$

where $I, Q \in \{\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\}$. Here f_o denotes the carrier frequency and the vectors I and Q carry one bit information each. $\phi(t)$ denotes the phase of the transmitted signal $s(t)$ (sign above must be negative due to complex number representation).

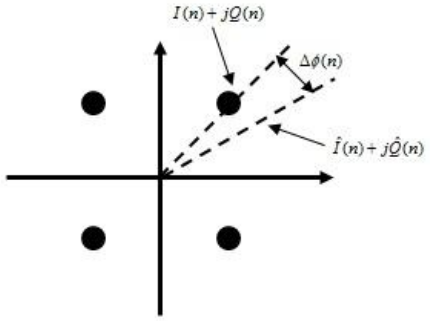

Fig. 1. QPSK constellation grid.

Once received, the signal is demodulated and the data can be extracted. The data can be represented as a constellation on an x-y plane in terms of symbols. The possible decoded symbols will be at 45 degrees, 135 degrees, 225 degrees, and 315 degrees as shown in Fig 1. Each of these constellation points, or symbols, represents two bits of information that are decoded based on their position in the constellation. The term quadrature is used to describe this method of modulation because the data is represented based on the quadrant the symbol is located in. In higher order phase-shift keying modulation schemes, such as 8PSK or 16PSK, the constellation becomes more crowded in

order to represent more symbols. Although the bits-per-symbol ratio has increased, the degree of accuracy needed for accurate transmission is also heightened, requiring a more complex demodulation scheme as well as more precise loop filter as discussed in section (d) of Methods and Procedures (III).

Multi-path Effect In wireless communication, multi-path fading is a real threat to reliable communication. During wireless transmission, a signal travels from the transmitter to the receiver via a number of different paths, which is multi-path effect as exhibited in Fig. 2. Point A represents the transmitter and point B the receiver. The sum of these signals at point B will result in what is called constructive or destructive interference.

In the QPSK application, the multi-path and relative motion between the transmitter and receiver cause amplitude attenuation, Doppler frequency shift, and phase shift in the constellation data points, as seen in Fig. 1. The Doppler frequency shift and coarse carrier synchronization cause the QPSK constellation to rotate and thus, decoding of the data is a real challenge. In order to lock onto the correct QPSK constellation and correctly recover the transmitted data, coarse and fine carrier synchronization is required.

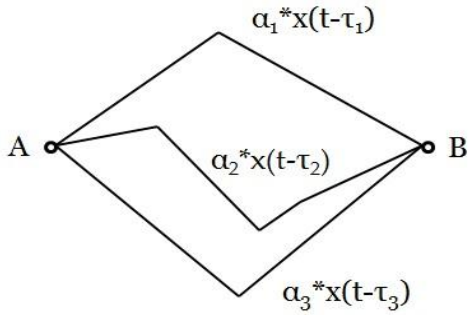

Fig. 2. Multi-path effect. Point A represents a transmitter and point B the corresponding receiver. During wireless communication, signals arriving at point B may take many different paths.

Carrier Synchronization Communication systems require local oscillators at both the transmitter and the receiver. The frequency and phase offset between these oscillators is inevitable due to various channel imperfections and transmission delay. Due to these imperfections, phase error is introduced in the received signal causing an incorrect representation of the transmitted data. In order to recover the data, a coherent detection is required and is achieved through a digital phase-locked loop (PLL). The coherent detection requires estimation of the instantaneous phase offset of every data point in the constellation grid with respect to where it should be. That information is then fed to a direct digital synthesizer which generates coherent sine and cosine carriers to correct the phase error. Upon completing carrier synchronization, data can be decoded. This is a crucial component of the SDR for successful demodulation of data.

Phase Ambiguity Every M-ary PSK system suffers from a condition called phase ambiguity, which is due to the nonlinear operation performed on the signal for carrier regeneration. The PLL locks onto a wrong phase and this incorrect locking characteristic introduces a static phase rotation. In QPSK systems, phase ambiguity occurs at any multiple of 90 degrees. This is to say that when the PLL "locks" onto the signal at the receiver, the symbol previously located in the first quadrant at the transmitter may now be located in a different quadrant at the receiver. The phase error exhibited in this symbol will also exist for every other symbol.

Taking a look at the effect of phase ambiguity in QPSK systems, it is apparent that each channel (I or Q) exhibits one of two different states and both channels combine for a total of four different states as seen by the constellation grid. The state of each channel may be either a replica of the transmitted signal or an inverted signal. Fig. 3 shows the I channel at both the transmitter and the receiver. The signal is inverted which will result in either a 90 or 180 degree phase ambiguity as seen in Fig. 4. If the Q channel has no inversion, the symbol will be seen in quadrant two with a 90 degree phase ambiguity, whereas if the Q channel is inverted, a 180 degree phase ambiguity will exist and the symbol will be located in the third quadrant.

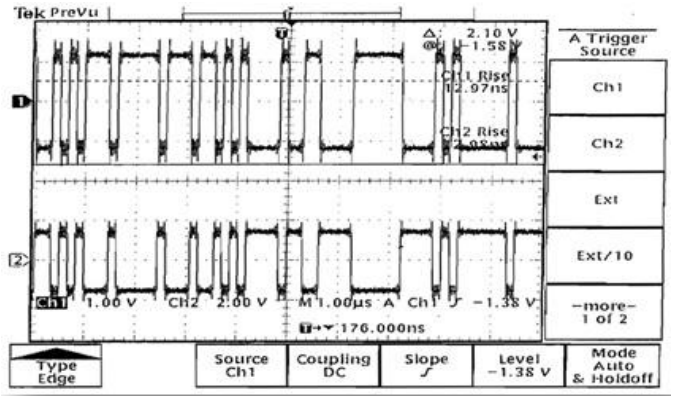

Fig. 3. Phase ambiguity in the I channel of a QPSK signal. The top plot represents the signal at the transmitter and the bottom plot, the signal at the receiver.

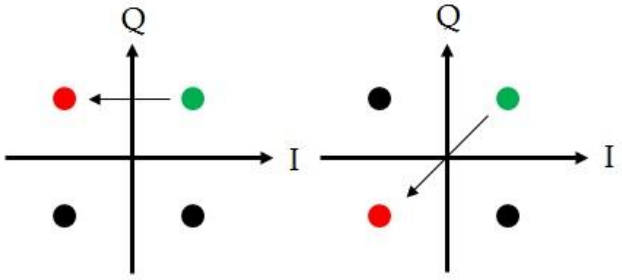

Fig. 4. Phase ambiguity possibilities for the signal in Fig. 3.

This phase ambiguity must be corrected for proper decoding of data. In order to do so, a differential coding scheme can be implemented into the system, which transmits the difference between two consecutive symbols of data rather than each symbol separately. By doing so, phase ambiguity is resolved by differencing the phases of two symbols.

III. Methods and Procedures

System Tools and Xilinx The entire QPSK communication system is created in the Simulink environment of MathWorks©. Simulink is a module based method for simulating and implementing an engineering system. The module libraries provide a broad range of modules for different tasks such as mathematical operations, signal processing algorithms, error calculations, filter design etc. Compounding various blocks from the communication and signal processing block sets into a pseudo-flowchart creates a fully functional system, without the traditional method of line by line coding.

The Xilinx system generator is a tool box running in Simulink environment for FPGA hardware design. It provides integrated design flow using hardware description language synthesis, core libraries, and FPGA implementation tools. Due to the discrete nature of FPGA architecture, data values are represented in a fixed-point binary format as opposed to a floating-point representation. The Xilinx block set includes a large number of functions which require fixed point arithmetic. Therefore, memory management and proper data representation require special attention, unlike floating-point signal processing. The Xilinx system generator block set is accessible in the Simulink library browser, and elements can be freely combined with other Simulink elements. It allows choosing the target FPGA device, system clock period, and other implementation options. It provides an efficient way to simulate and verify the design from system specification level to hardware realization level.

Fixed-point representation is essentially a method of representing a value with a designated number of bits and precision. For example, a data type, *UFix_14_12*, stands for a 14-bit unsigned fixed-point number with 2 bits of integer data and 12 bits of decimal precision. In digital systems, selecting a proper fixed-point representation is crucial for functionality and progress. Improper representation can lead to loss of precision, overflows, sign errors, and other discontinuities throughout the rest of the system.

In the QPSK communication system, the transmitted data has the representation of Fix_32_31, describing a 32 bit

signed number, with the most significant bit being the sign bit, and the remaining 31 bits defining the decimal precision. The range of the signed value is from -1 to 1. The overall system diagram of QPSK system is shown in Fig. 5. It includes the following function blocks: data generator, interpolator, upconverter, channel, downconverter, decimator, carrier recovery, and decoding.

Data Generation The QPSK system has the ability to transmit two bits of information for every symbol. The "Data Generation" block in Fig. 5 handles the process of converting two-bit ROM data to allocate bit streams for the I and Q channels. This data is then read, differential encoding applied, and each encoded symbol is grouped into a pair of one bits. These bits are then translated into corresponding amplitudes and polarities, as shown in Fig 1.

Converting real data from an image or a sound file requires special data conditioning prior to encoding or splicing of symbols. In our project, an 8 bit/pixel image is used. Each pixel is split into four consecutive two-bit symbols. This is completed using time division multiplexers to stream the symbols sequentially, and time division de-multiplexers to extract the corresponding symbols.

Raised-Cosine Filter After generating data, the QPSK signal is passed through a polyphase finite impulse response (FIR) interpolator. A common signal distortion in communication system is inter-symbol interference (ISI) among adjacent symbols. The effect is present when a "+1" changes to a "-1" abruptly, causing a high frequency spike and data distortion. The ISI degrades system performance unless properly corrected using channel equalization. In the QPSK system, the FIR filter generates raised-cosine pulses, which suppress the ISI. After being interpolated, the data is modulated by 12.5 MHz sine and cosine carriers and then sent through the channel. The received signal is down-converted and decimated for demodulation. The decimator utilizes the same FIR filter coefficients as the interpolator. The filter chosen for the project has a pass-band at 0.25π and has 64 taps. It has an attenuation of -30dB and a -6dB/decade stop-band at 0.325π. These characteristics shape the signal for adequate transmission, free of ISI and erroneous energy spikes.

Xilinx block set provides an FIR filter block for filter design. It uses distributed arithmetic structure. For the QPSK system which is inherently a multirate signal processing system, the distributed arithmetic FIR design is not an efficient implementation in terms of hardware resource and processing speed. A polyphase structure is used instead. The polyphase FIR raised-cosine filter is an efficient method used to interpolate and decimate, especially in a discrete-time system using an FPGA development system. It reduces the hardware resource and processing delay significantly.

Phase-Locked Loop The synchronization of the carrier frequencies is the most important piece of the communication system. In a wireless communication channel, various channel imperfections like ISI and Rayleigh fading due to multi-path effect exist. Without a synchronized local oscillator, data cannot be extracted correctly from the received QPSK signal. Implementing a phase-locked loop (PLL) is a standard method used to regenerate carrier frequency and correct for the channel imperfections introduced during transmission.

The QPSK waveform carries phase information that represents data. By performing the following computation

$$\sin(\Delta\phi) = \frac{\hat{I}(n)Y(n) - \hat{Q}(n)X(n)}{\sqrt{\hat{I}^2(n)+\hat{Q}^2(n)}\sqrt{X^2(n)+Y^2(n)}}, \qquad (1)$$

where $\Delta\phi$ denotes the phase error, $\mathrm{X(n)}$ and $\mathrm{Y(n)}$ are the outputs from decimators, the phase information can be extracted. $\hat{I}(n)$ and $\hat{Q}(n)$ are the estimated in-phase and quadrature hard-decoded data.

The phase error ($\Delta\phi$) in Fig. 1 is calculated and then used to adjust the frequency and phase of the local oscillator. As the error is reduced to zero, the transmitter and receiver frequencies become synchronized, allowing

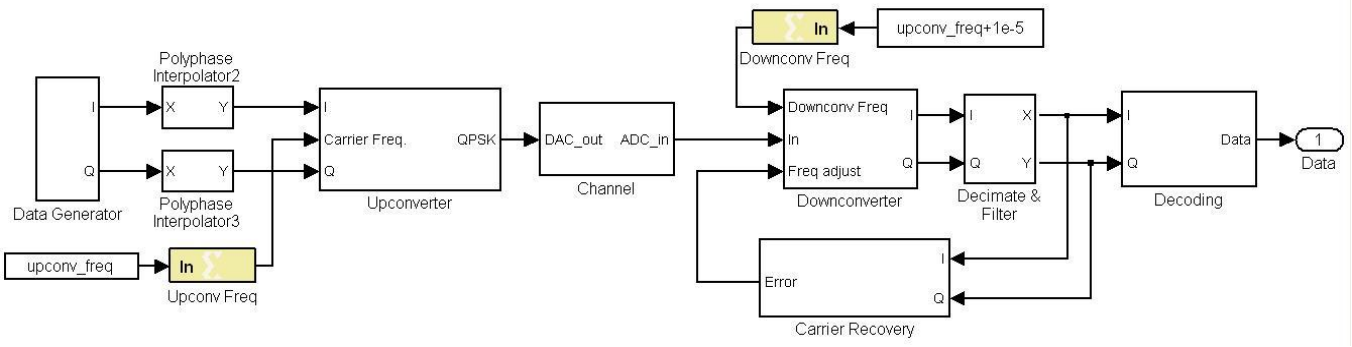

Fig. 5. Overall QPSK system block diagram.

for extraction of data from the $\hat{I}$ and $\hat{Q}$ channels.

Equation (2) is calculated for every data sample, meaning that this must be implemented in hardware and run at the symbol rate. The phase error calculation is illustrated in Fig. 6. This rapid calculation is occurring in the "Carrier Recovery" block of the system block diagram of Fig. 5 and serves as a feedback controller for the adjustment of the local oscillator.

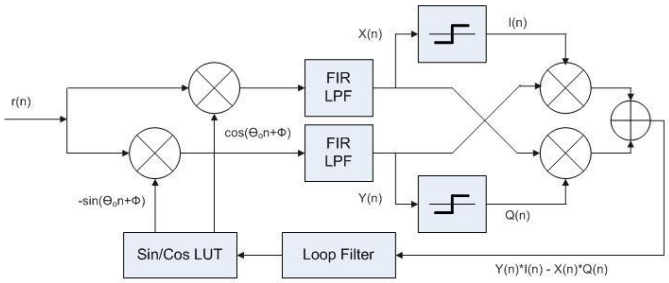

Fig. 6. Phase-locked loop.

Crucial to the design of the PLL is the loop filter, which provides direct control over the PLL bandwidth. The bandwidth of a PLL is the measure of the PLL's ability to track the input clock and jitter. A high bandwidth setting provides a fast lock time and tracks jitter on the reference clock source, passing it through to the PLL output, whereas a low bandwidth setting filters out reference clock jitter, but increases lock time [4].

To control phase error, proportional and integral (PI) control techniques are used via a loop filter. PI control is implemented by selecting values for K_p and K_i, which provide the user with control over bandwidth and a damping factor. By optimizing these parameters, the system can achieve carrier synchronization with both a fast locking time and reduced jitter. First, the bandwidth for desired operation is chosen and then K_p and K_i values are derived using Equations (3) and (4).

$$K_p = \frac{2\sqrt{2}\theta}{1+\sqrt{2}\theta+\theta^2} \tag{3}$$

$$K_i = \frac{4\theta^2}{1+\sqrt{2}\theta+\theta^2} \tag{4}$$

where $\theta = 2\pi \times BW$ and BW is bandwidth of the filter in Hz. The bandwidth is set at 1 kHz for carrier recovery and can be adjusted according to the desired behavior of the PLL with the considerations stated above.

Differential Coding Differential coding is necessary to compensate for the errors induced from phase ambiguity, an inherent problem of QPSK systems. As stated in Section II - D, differential coding is a part of the data conditioning where complex data is converted to a sequence of two-bit symbols. Differential encoding is then applied to the stream of two-bit symbols and later the decoding process is applied after the $\hat{I}(n)$ and $\hat{Q}(n)$ channels have been re-combined at the receiver end. The result is a data sequence corrected of phase ambiguity.

The encoding process calculates the difference between consecutive symbols and that value is transmitted. This is achieved by putting the symbol sequence through a discrete time integrator. Once the data is transmitted and received, the decoding process begins by feeding the encoded symbols through a discrete time differentiator. Differential coding consists of a total of four Xilinx blocks for implementation and provides a reliable method for correcting phase ambiguity.

IV. RESULTS

Simulink is an extremely helpful simulation tool that allows for verification of a system's operation without physically uploading it onto the FPGA board. Once the phase-locked loop was implemented into the system, the complete QPSK system shown in Fig. 5 has been downloaded to the FPGA board successfully. 14-bit high speed (up to 65M samples/second) digital-to-analog converter and analog-to-digital devices are used as the communication channel. When the $\hat{I}(n)$ and $\hat{Q}(n)$ channels of the received signal are combined in an x-y representation, they form the QPSK constellation grid. Without carrier synchronization, the QPSK constellation grid was not available; rather

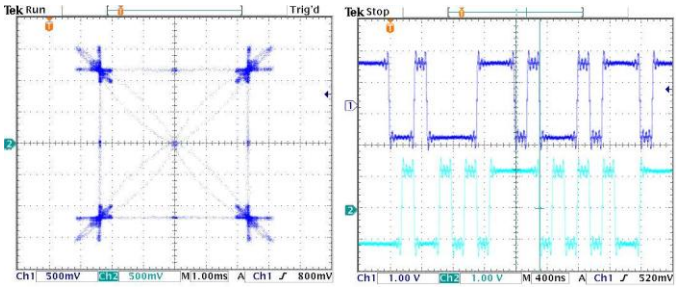

Fig. 7. Hardware implementation results shown on an Agilent oscilloscope. Left—QPSK constellation recovered after carrier synchronization. Right—I and Q channels at the receiver.

a rotating phasor was observed. The recovered constellation grid is shown in Fig. 7. The PLL tracks the carrier frequency offset and synchronizes the carrier frequencies successfully.

Once system operation is verified on the FPGA device, further simulation techniques are used to analyze the phase ambiguity condition of the system. Visual results are the desired approach, in order to verify and evaluate the design. The image shown in Fig. 8 is used for testing of the transmitter and receiver. In order to do so, image processing techniques are required to convert image data to a QPSK representation. After pre- and post-processing, transmission and recovery of the image can be completed. As seen in Fig. 8, a phase ambiguity of 180 degrees exists, resulting in the inversion of colors. An issue discovered during simulation is the shifting of the image. In Fig. 8, image (b) at the receiver is shifted down a number of rows. This is due to a suboptimal method of storing the received data. The delay of the system, the time it takes for the data to be generated, interpolated, modulated, transmitted, demodulated, decimated, and recovered, is directly related to the number of downward shifts displayed in Fig. 8b and Fig. 9b. The received data does not account for the system delay in its data storing algorithm and includes the data points where *no* real data has been received. This can be solved by properly setting a parameter in the data acquisition system block. Regardless, the images were received intact.

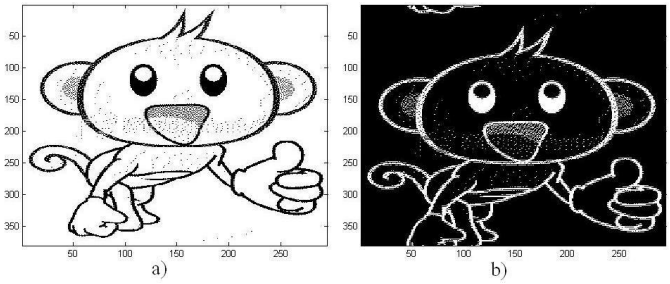

Fig. 8. Phase ambiguity in tested image. a) transmitted image. b) recovered image.

However, by means of differential coding, any phase ambiguity can be corrected. This can be seen in Fig. 9 where no color distortion occurs. In order to preserve color features of the image, additional pre- and post-modulation processing is required to convert the eight bit numbers of the red-green-blue map to a binary representation and then a QPSK representation. This is handled in hardware utilizing the time division multiplexer and de-multiplexer Xilinx blocks. The goal is to limit the amount of user processing required for successful data recovery. A successful simulation of sending the red component of a 25 by 25 pixel image produces the results in Fig. 9. Note in (b) the shifting of the image still exists and is proportion to the inherit system delay. See Fig. A2 for the result of a simulation where the data was stored and compiled properly, accounting for the system delay.

These simulations prove that our system can handle basic image data and full color image transmission is possible with the proper data storage and simulation settings. Hardware implementation will require a method of starting and stopping the saving of the image file and is not intended as part of the scope of the project.

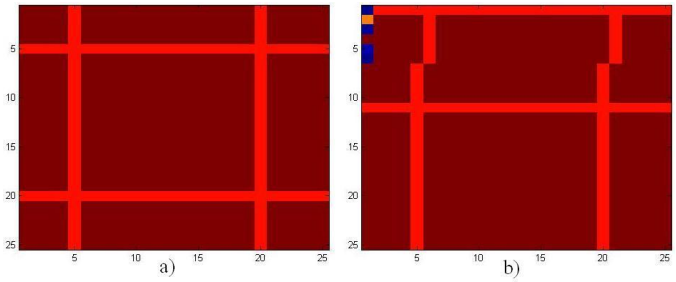

Fig. 9. 8-bit color preservation test after phase ambiguity correction. a) transmitted image. b) recovered image.

V. CONCLUSION

The software-defined radio is a very flexible solution to the digital wireless communications industry. Xilinx provides a powerful design tool for FPGA implementation. In this project, a SDR with QPSK modulation scheme has been successfully designed using Xilinx tools. The PLL and differential coding scheme show robustness in recovering carrier frequency offset and solving the phase ambiguity problem of the QPSK system. It can be extended to other bandwidth-efficient digital modulations, overcoming the problems such as phase shift introduced by channel imperfections, signal attenuation, Doppler frequency shift etc. Moreover, the software simulation and hardware implementation procedures show that this type of hardware/software design methodology is well suited to a broad range of applications in communication and signal processing.

VI. REFERENCES

[1] Chris Dick, Fred Harris, and Michael Rice, "*FPGA implementation of carrier synchronization for QAM receivers*", Journal of VLSI Signal Processing, Vol. 36, pp. 57-71, Kluwer academic publishers, Netherland, 2004.

[2] Stephens, Donald R. Phase-locked loops for wireless communications digital and analog implementation. Boston: Kluwer Academic, 1998.

[3] Vinod Kumar Venkat Reddy Gari, "*FPGA-based QPSK transceiver design*", Technical Report, Department of Electrical and Computer Engineering, Bradley University, November 2008.

[4] Altera Corporation, "PLL & Clocking Glossary," Altera, 1995-2010. [Online]. Available: http://www.altera.com. [Accessed: May 1, 2010].

APPENDIX A

The overall system is shown in Fig. A1 and some simulation results are shown in Fig. A2.

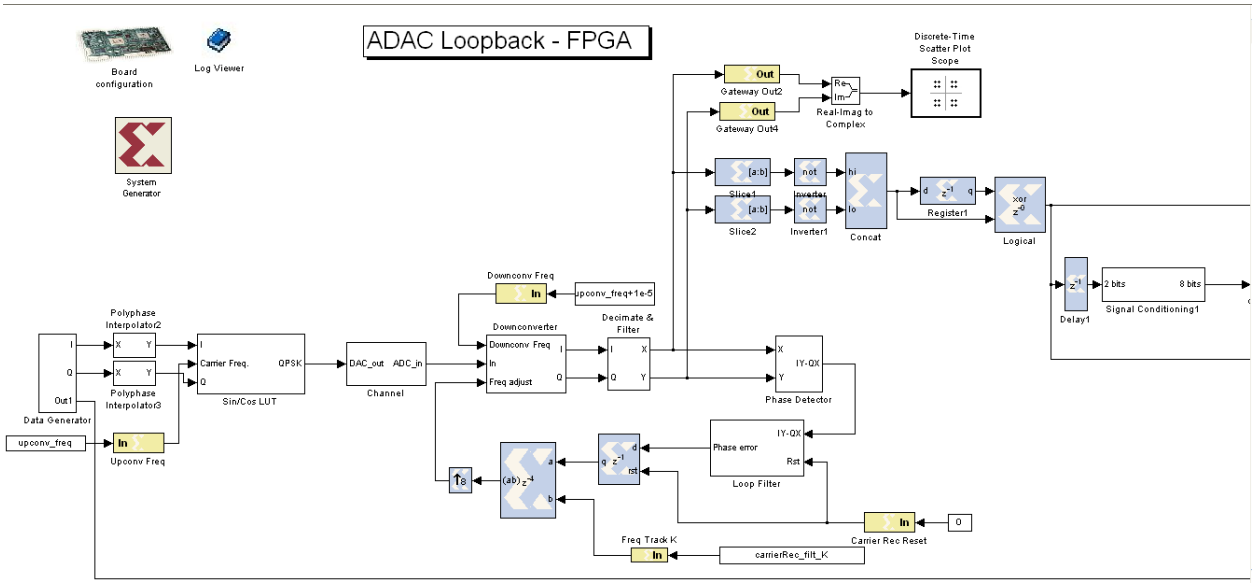

Fig. A1. Overall system block diagram; includes 8-bit to 2-bit data conditioning, differential encoding, and ADAC configured channel. This version is designed for the SignalWave Virtex-II development board. Hardware implementation is yet to be successful.

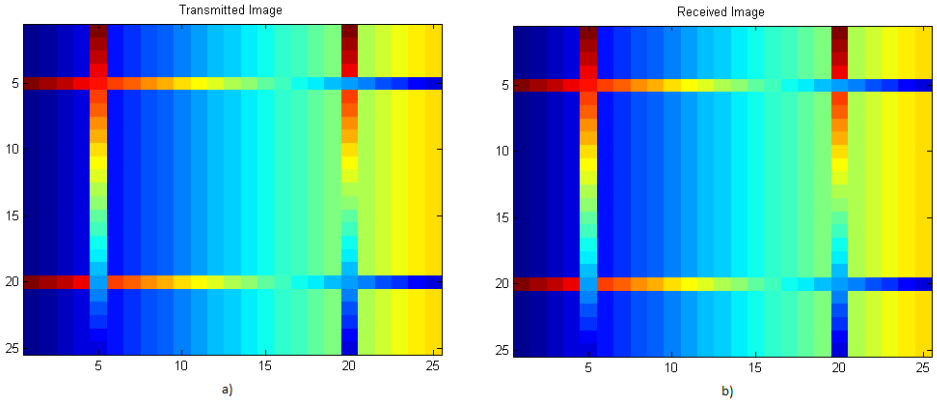

Fig. A2. Simulation result of a successful transmission and data recovery. The variety in color shows that the data was properly received, stored, and compiled back into an image.

APPENDIX B

PROCEDURE AND DESIGN

To adequately complete the Software-Defined Radio Using Xilinx project a great number of tasks were completed. The order of the tasks, their significance, and the most meticulous detail is crucial to a successful attempt at replicating our work for future projects. To help save time and confusion for those individuals, the following procedure has been compiled. A certain basic level of understanding the MATLAB and SIMULINK environments are required before beginning the project.

A. Initial Settings and Configuration for Xilinx-Simulink Model Design

1) System generator

Open SIMULINK in MATLAB.

Open a new SIMULINK model.

Open the SIMULINK block set library.

Locate the Xilinx Block Set and add the System Generator block to the model.

If using the Virtex-II hardware, also include the Board Configuration block located in the Lyrtech block set.

Assign appropriate values to the Xilinx System Generator block by double clicking on it to open its menu.

Virtex4 settings:

Compilation: Hardware Co-Simulation-XtremeDSP Development Kit-PCI and USB. Part: Virtex4 xc4vsx35-10ff668. Target: "./netlist". Synthesis tool: XST. Hardware descriptive language: VHDL. FPGA clock period: 20ns ~> 50 MHz. Clock pin location: leave blank or FIXED. Simulink system period (sec): 1/8. Block icon display: Default.

Virtex-II settings:

Compilation: Hardware Co-Simulation-Lyrtech-SignalWAVe EVB. Part: Virtex2 xc2v3000-4ff1152. Target: "./netlist". Synthesis tool: XST. Hardware descriptive language: VHDL. FPGA clock period: 20ns ~> 50 MHz. Clock pin location: leave blank. Simulink system period (sec): 1/8. Block icon display: Default.

Board Configuration: Clock type: Free Running Hardware Clock. Clock Source: ADC Clock. Check "copy bitstream to current directory."

The "target" option just means where it will place all the important bitstream files and other work files. This can be set to anything the user wants. The term "./" means that the working directory will be created in the current directory that MATLAB is using.

These are the necessary hardware configuration settings so that a hardware co-simulation can be run later.

2) Simulink – CRITICAL STEP

Click on the Simulation heading, open Configuration Parameters.

Change SOLVER setting to "discrete (no continuous states)" and ensure that "Type" is set to "Variable Step".

3) Design and Helpful Commentary

Begin the design process by ONLY using Xilinx block set components. Keep the Xilinx block set library available as it will be used frequently. Implementation should be use a module by module technique; verify that each one is operating properly before continuing onto the next. Be sure to run the "Comm_init.m" file to initialize critical data values for various components of the design. If no such file exists, further design work is necessary to calculate these values. The following modules are required for the QPSK project and are listed in sequential order:

- Discrete ROM with Sampled Counter – 8-bit stored data. Sampled at a period of 4.
  This will require a system to stop the ROM when all the data has been processed.
- 8-bit to 2-bit Time Division Multiplexing Converter. Converts period to 1.
- Differential Encoder (discrete integrator)
- QPSK 2 bit data representation – slice, then select amplitude: logic 1 = $-1/\sqrt{2}$; logic 0 = $+1/\sqrt{2}$.
  I (most significant bit) and Q (least significant bit) channels created.
- Polyphase Raised-Cosine Interpolators – One filter for each channel. Identical filter coefficients.
  Data is now up-sampled by 8, meaning that the sample period is now 1/8.
- 12.5 MHz modulation – channel I is modulated with a cosine and channel Q is modulated with a sine.
  Their results are summed together and are ready for broadcast. 0.25 corresponds to 1/4th of the Xilinx clock ~ 12.5MHz.

- Format signal for DAC and ADC recovery. Virtex-II has a 14-bit ADC and only recognizes positive values. Assistance is necessary to convert the unsigned 14 bit value to a signed 14 bit value so further data processing can be successful. Sample period is 1/8.
- Receiver Demodulation – ~12.5MHz oscillator with offset to simulate channel imperfections. It uses the local oscillator with a simulation "offset" and a corrective factor to synchronize with the transmitter clock and extract the $\hat{I}$ and $\hat{Q}$ channels. A cosine oscillator extracts the $\hat{I}$ channel and a sine oscillator extracts the $\hat{Q}$ channel. Sample period is 1/8.
- Polyphase Raised-Cosine Decimator – One filter per channel. Sample period is 1.

After demodulation, two concurrent things will occur: the phase-locked loop, and data extraction, conversion, and output. The phase-locked loop will be discussed first knowing that without its functionality, no data can be extracted.

- Phase Detector – Cross multiply the $\hat{I}$ and $\hat{Q}$ channels and subtract their results according to equation (1) and Fig. 6. The result is the instantaneous phase information. This information is sent to the Loop Filter.
- Loop Filter – This uses the phase information to condition an output that can be used to adjust the local oscillator. There are 3 constants that define the performance of this filter: K, K_i, K_p. These are calculated using equations (3) and (4). If these values are not calculated correctly the system will not lock or exhibit unpredictable behavior. A loop filter consists of a proportional gain (K_p), and integrator gain (K_i), and a final gain outside the loop (K). The output of this filter is up-sampled by 8 so that it can be directly subtracted from the local oscillator's clock value for each incoming sample. This avoids the usual requirement for a separate sine/cosine look-up table and instead reuses the one already in place. This correcting factor is fed into the Receiver Demodulation module as stated above.

Now that a lock has been achieved, and a nice, 4-point constellation plot can be plotted, it is time for the data to be extracted and represented in the same form it was in prior to transmission.

- Determine logical levels – A simple slice is used to extract the MSB of the $\hat{I}$ and $\hat{Q}$ channels and determine logic. In QPSK, the threshold for logic is at 0 in both the $\hat{I}$ and $\hat{Q}$ channels, where the optimum value is $\pm 1/\sqrt{2}$. This makes it easy as only the sign bit is sufficient for data extraction. Invert and then concatenate the $\hat{I}$ and $\hat{Q}$ to create an accurate representation of the 2-bit value that was transmitted.
- Differential Decoder (discrete differentiator)
- System Delay Compensator – This is not so much a module as it is an entirely necessary component for data recovery. This delay is set to 2. The reason is to compensate for the total system delay (from data generation to transmission to $\hat{I}$ and $\hat{Q}$ data extraction). This is required for accurate time division de-multiplexing. Without this factor the reconstructed data is not accurate.
- 2-bit to 8-bit Data Conditioning – Time division de-multiplexing. This combines 4 sequential 2-bit symbols to reconstruct the original transmitted data in 8-bit values. Sample period is 4.
- Data Storage and Run Time – export to MATLAB Data storage should be limited to the last [size of transmitted data] values received. This ensures that any of the preamble information used for synchronization or padding is not affecting the meaningful data. Also, ensure that the simulation has run for the entire length of time required for transmission, and then stops. The formula for simulation run time is: length(input)*4 + (system_delay)*4. Depending on the data conditioning (8-bit/2-bit convert) scheme used, these components will not be multiplied by 4, but rather the number that results from dividing the original number of bits by 2. ($^{8}/_{2} = 4$). The system_delay of this project is 20. This time is set so that all the data that was transmitted is received and stored, and not halted while data is still being processed by the various system components that require time to complete their tasks.

4) Hardware Co-Simulation

a) VIRTEX4

Hardware co-simulation on the Virtex4 is relatively easy in its generation and configuration. In the System Generator block, click "Generate" and allow 5 to 10 minutes for complete synthesis of the design into hardware language. If it takes longer than 10 minutes, it may be worth shutting down MATLAB and trying again. Upon completion, a new SIMULINK model window will appear with the Xilinx hardware

co-simulation block. Drag this block into the original design model; connect all the input components such as the loop filter constants and the modulation frequency values. Close the other window that the synthesis generated. Double-click the hardware co-simulation block and make sure the correct bitstream file is selected. Also select the method of communication (PCI and USB or JTAG) accordingly and click OK. Click the play button to begin simulation.

b) VIRTEX-II

The Virtex-II hardware co-simulation is more complicated than the Virtex4. It requires a DSP model to run in tandem with the FPGA bitstream. Generation of the bitstream is identical to the Virtex4 (refer to the above section), however no hardware co-simulation block is generated. The bitstream should be located in the current directory or in the "netlist" or "working" directory specified in the System Generator block. This bitstream file needs to be included in the Real-Time Workshop configuration parameters of the DSP model. Please refer to the Lyrtech tutorials for a reference DSP design model. This is part of the Lyrtech development board that allows the FPGA and DSP chips to interact with each other. The ADAC is controlled by the DSP chip, thus it is required for wireless transmission.

With the DSP model in the foreground:

In the menu bar of the DSP model, next to the play button for simulation, change the simulation mode from Normal to External.

Click Simulation in the top menu.

Select Configuration Parameters.

Select the "Real-Time Workshop" menu in the tree on the left.

Under the Target Selection segment, set System Target file to "rt_swave.tlc" This can be selected by clicking Browse.

Leave TLC options blank.

Check the box next to "Generate makefile".

Set "Make command" to: make_rtw

Set "Template makefile" to: rt_swave.tmf

Click on "Interface" from the menu tree on the left,

Under the "Data exchange" segment, set "interface mode" to External Mode.

Under "Host/Target Interface", Set Transport Layer to "SignalMaster Communication for DSPLink".

Still under that segment, set "MEX-file arguments" to: 'autodetect' 'fpgaload' 'name_of_your_bitstream.bit'

Be sure that the bitstream file is located in the current working directory of MATLAB.

Click OK.

In the menu bar of the DSP model, click the "external hardware connect" button, next to the play button, to begin the hardware simulation. Select the serial port that corresponds to the Virtex-II hardware and a corresponding baud rate, and allow for the system to download the FPGA bitstream first, then the DSP out file. This may take up to 5 minutes and is automatic. Note that if any other window is clicked during the download process, the status bar vanishes. This does not mean SIMULINK has stopped working or the download stopped.