# Reconfigurable Communication System Design

By: Anthony Gaught

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

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

May 2013

## Abstract

Due to its reconfigurable ability and high throughput performance, Field Programmable Gate Array (FPGA) is widely used in embedded applications such as automotive, communications, industrial automation, motor control, medical imaging, etc. In this study, a reconfigurable communication system using Quadrature Phase Shift Keying (QPSK) modulation is designed and implemented on FPGAs. Some preliminary work has already been done at Bradley University. For example, a QPSK system has been implemented on a Virtex4 FPGA board using MATLAB/Simulink and Xilinx system generator. The major drawback of Simulink-based tool is that the logic resource usage highly depends on the software automation and the design is not guaranteed to be compatible under different versions of software. Hardware description language (HDL) could provide a better solution in terms of logic resource usage and design compatibility.

The project goal is to implement a complete QPSK system on FPGA boards using HDL. It aims to construct the system with hardware-efficient modules which can be reusable and expandable with additional features later. Both the transmitter and receiver have been implemented on a single FPGA board which allows channel imperfections to be ignored. Separate FPGA boards have been used to split the design into transmitter and receiver sides. A carrier recovery circuit and a phase locked loop are used on the receiver side to correctly recover transmitted data. The designs have been successfully verified by simulation and implementation.

## Acknowledgments

First and foremost, I would like to thank my advisors for this project, Dr. Yufeng Lu and Dr. In Soo Ahn for the valuable guidance and advice. They inspired me greatly to work on this project. Their willingness to motivate me contributed tremendously to my project. I would also like to thank Bradley University for providing me with a good environment and facilities to complete this project.

# Table of Contents

## Contents

# List of Figures

## I. Introduction

Due to its reconfigurable ability and high throughput performance, Field Programmable Gate Array (FPGA) is widely used in embedded applications such as automotive, communication, industrial automation, motor control, and medical imaging, to name a few. Quadrature Phase Shift Keying (QPSK) is one of the digital modulation standards used in third and fourth generation wireless communication systems. In this project, a reconfigurable communication system using QPSK modulation will be designed and implemented on a pair of FPGAs using VHDL. An efficient verification flow is proposed and applied to the project. A carrier recovery circuit using a digital phase locked loop (PLL) is used to correct carrier offset. Phase ambiguity and channel effects are discussed.

## II. Background

Figure 1 shows a QPSK communication system. QPSK is a digital modulation scheme that transmits two bits of data per symbol. The data is split into the in-phase component, I(t) and the quadrature-phase component, Q(t). It is up-sampled and passed through a square root raised cosine pulse shaping filter in order to reduce inter-symbol interference (ISI) and improve bandwidth efficiency.

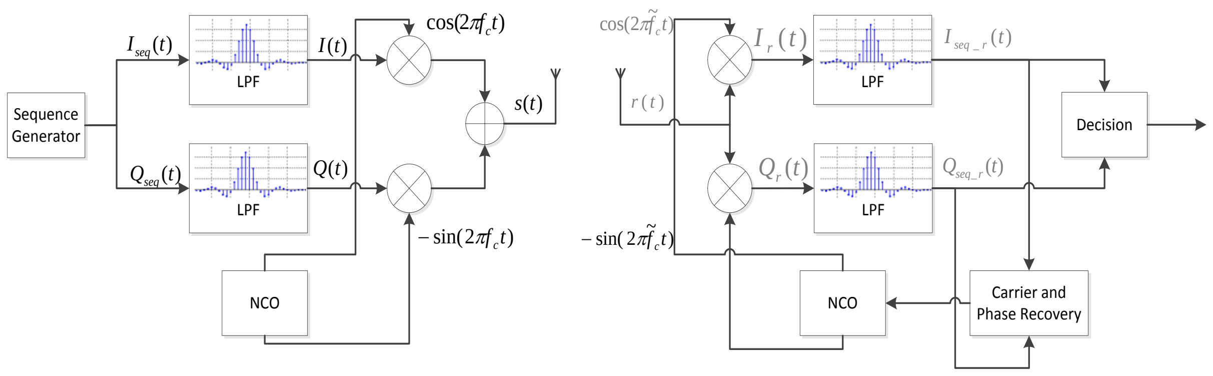


Figure 1 Block diagram of a QPSK communication system

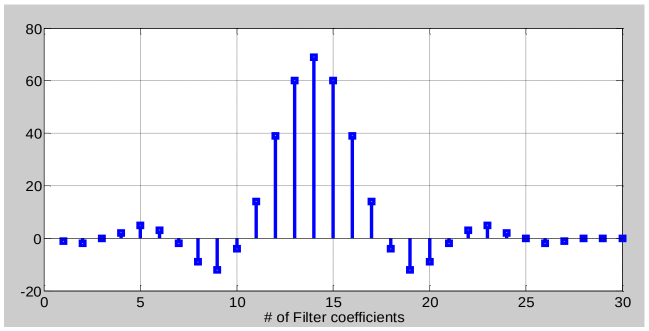


Figure 2 26-tap raised cosine filter with fixed-point 12-bit coefficients

Carrier signals from a numerically controlled oscillator (NCO) are used to modulate I(t) and Q(t). The mathematical representation for the transmitted signal s(t) can be seen in Equation 1.

$$s(t) = I(t)\cos(2\pi f_o t) - Q(t)\sin(2\pi f_o t) \tag{1}$$

The signal s(t) can be represented by a constellation plot in the x-y plane. The data is decoded based upon the phase of the received symbol. The QPSK constellation plot is shown in Figure 3.

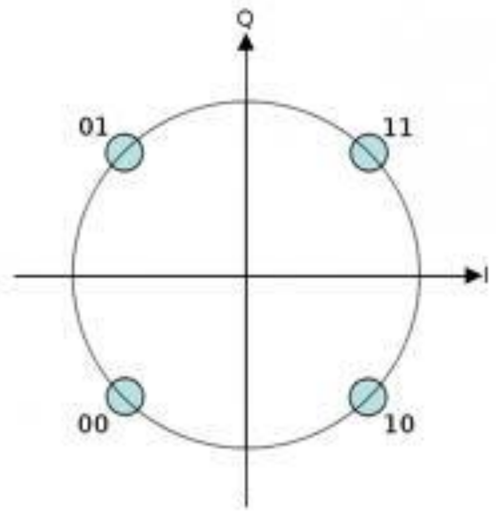


Figure 3 QPSK Constellation Plot

The received signal is demodulated by local cosine and sine carriers and then passes through a matched filter and down-sampled. This filter is identical to the filter used in the transmitter. Higher order M-ary modulation schemes can be used which transmit a larger number of bits per symbol such as 16QAM but with the increased data rate comes the need for a more accurate transmission and more complex demodulation schemes.

Because the transmitter and receiver are using separate oscillators with an unknown amount of delay between them a frequency and phase offset exists. The constellation points are rotated because of the offset. A digital phase-locked loop (PLL) is used in the carrier recovery circuit in order to estimate the phase error. The phase error estimate is used to adjust the output of the direct digital synthesizer. The phase error is calculated using equations (2), (3) and (4) where $\phi$ denotes the phase error, I(n) and Q(n) are the outputs from decimators, $\hat{I}(n)$ and $\hat{Q}(n)$ are the estimated in-phase and quadrature-phase hard-decoded data.

$$\sin(\phi) = \frac{I(n)\hat{Q}(n) - Q(n)\hat{I}(n)}{\sqrt{\hat{I}^2(n) + \hat{Q}^2(n)}\sqrt{I^2(n) + Q^2(n)}} \tag{2}$$

$$\phi \cong \sin(\phi) \tag{3}$$

$$\phi \cong I(n)\hat{Q}(n) - Q(n)\hat{I}(n) \tag{4}$$

A proportional and integral (PI) controller is used to control bandwidth and damping factor. By properly selecting the $K_p$ and $K_i$ values, fast locking time and reduced jitter can be achieved. The bandwidth was chosen first and the other parameters are derived using equations (5), (6) and (7) where BW is the filter bandwidth.

$$\theta = 2\pi * BW \tag{5}$$

$$K_p = \frac{2\sqrt{2}\theta}{1+\sqrt{2}\theta+\theta^2} \tag{6}$$

$$K_i = \frac{4\theta^2}{1+\sqrt{2}\theta+\theta^2} \tag{7}$$

The system experiences phase ambiguity even after carrier synchronization is attained. Phase ambiguity is inherent in regeneration of the PSK carrier signal due to nonlinear operation of forming the fourth power of the received signal. The PLL can be locked onto an incorrect phase different from the correct phase by a multiple of 90 degrees. The data output by the receiver will be incorrect if the phase ambiguity is not resolved. The phase ambiguity can be resolved by sending a known sequence to measure the amount of rotation or by using a differential encoding scheme.

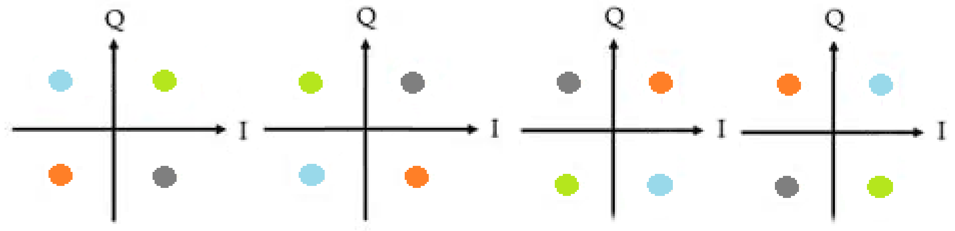


**Figure 4 Possible phase ambiguity states**

## III. Methods and Procedures

There are several methods to design and implement a communication system on FPGAs, each with its own benefits and drawbacks. The VHDL approach is hardware efficient and it is guaranteed to be compatible with any version of software but it requires prior knowledge and experience. In addition to prior knowledge and experience the designer must also be familiar with the system at the logic level. The design flow of HDL for FPGA implementation can be seen in Figure 4. MATLAB/Simulink can be used with a digital signal processing (DSP) block-set and core generator to design the system at a higher level. By using this approach the designer does not need to possess an in depth knowledge of how the system operates at a low level and it is faster than using VHDL directly. Unfortunately the design is constricted to a specific FPGA device family and certain versions of software. In addition this design method is less efficient than using VHDL. The design flow of the system generator for FPGA is shown in Figure 5.

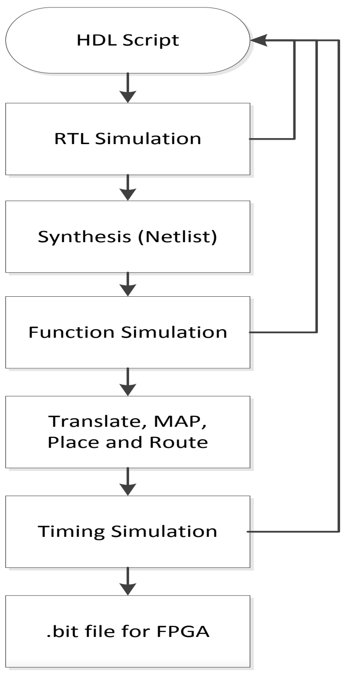


Figure 5 HDL design flow for FPGA using Xilinx tools

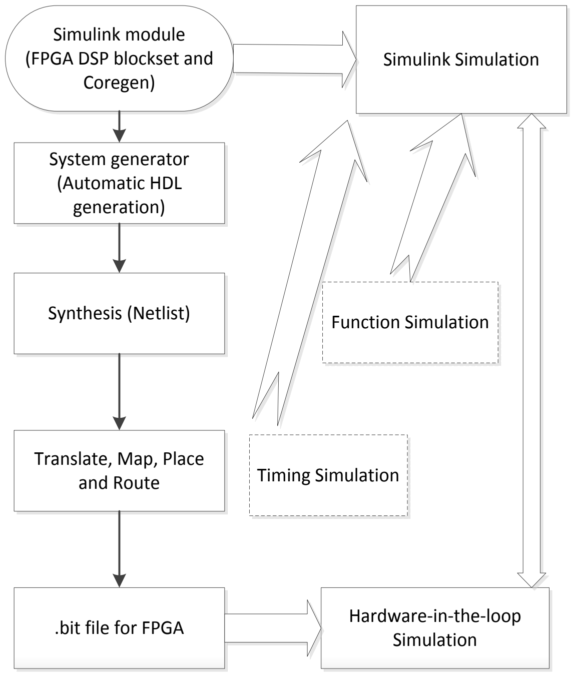
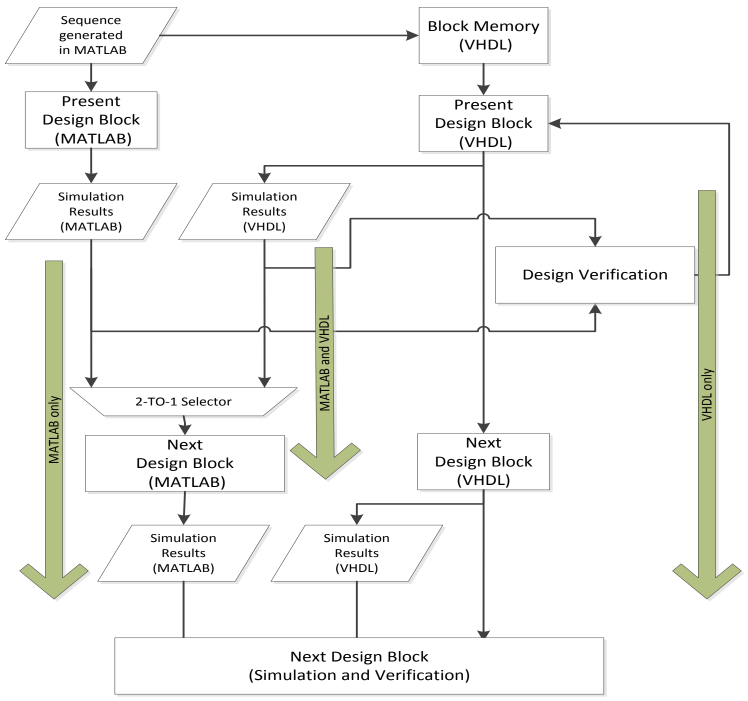

Simulink module (FPGA DSP blockset and Coregen)
Simulink Simulation
System generator (Automatic HDL generation)
Synthesis (Netlist)
Function Simulation
Translate, Map, Place and Route
Timing Simulation
.bit file for FPGA
Hardware-in-the-loop Simulation

Figure 6 System generator design flow for FPGA

For a low-cost reconfigurable system, manual optimization is still needed. Consequently, a design flow for QPSK system verification is proposed in Figure 6. The verification process includes three threads that run in parallel throughout the design. The first thread is to simulate the system or subsystem using MATLAB/Simulink only. It is the reference design for hardware implementation. The second thread is to design the system using VHDL only. It is the target design for hardware. The third thread is to simulate the system or subsystem using both VHDL and MATLAB. The simulation results are cross-checked and verified between VHDL and MATLAB.



Sequence generated in MATLAB
Block Memory (VHDL)
Present Design Block (MATLAB)
Present Design Block (VHDL)
Simulation Results (MATLAB)
Simulation Results (VHDL)
Design Verification
MATLAB only
MATLAB and VHDL
VHDL only
2-TO-1 Selector
Next Design Block (MATLAB)
Next Design Block (VHDL)
Simulation Results (MATLAB)
Simulation Results (VHDL)
Next Design Block (Simulation and Verification)

Figure 7 Design flow for QPSK system verification

Fixed point data representation is used throughout the VHDL design. It is a method used to keep track of where the decimal point is located in a binary system. The way to express a fixed point number is FIX_M_N where M is the total number of bits and N is the number of bits to the right of the decimal point for example s(t) is FIX_12_11 which means it is represented by 12 bits total of which 11 are to the right of the decimal point. 2's compliment is also used throughout the design.

## IV. Simulation Results

Figures 8-17 show that the VHDL simulation results match with the results obtained from the MATLAB/Simulink model. The signals shown in these plots represent key points throughout the design.

Proper implementation of the raised cosine shaping filter was one of the major hurdles to be handled upfront in the project. Although the first filter design approximated the model closely it was not accurate enough for the carrier recovery circuit. To correct this problem the number of coefficients and the number of bits used to represent them are increased. The accuracy of the received constellation could be improved by increasing the number of filter coefficients in the future.

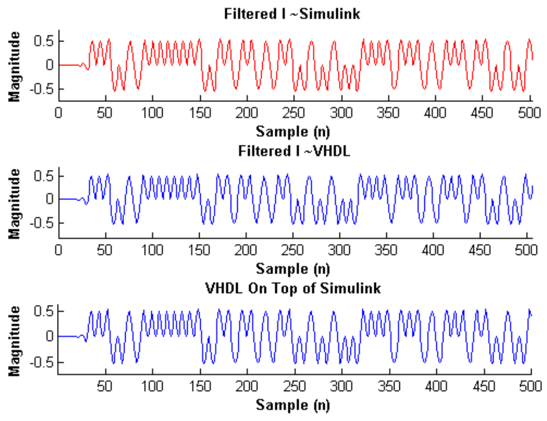


Figure 8 Filtered I waveform

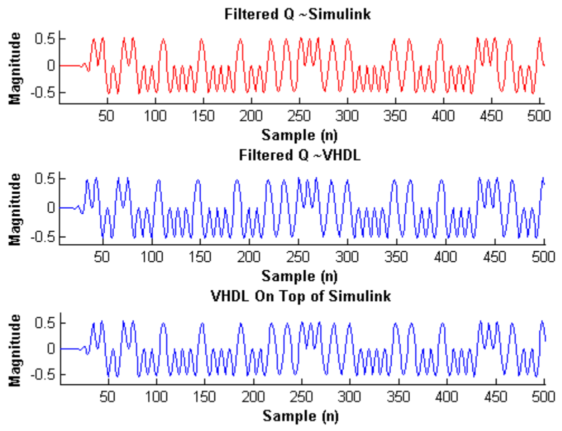


Figure 9 Filtered Q waveform

Because the modulated signal closely approximates the model one can infer that the entire transmitter system is operating correctly.

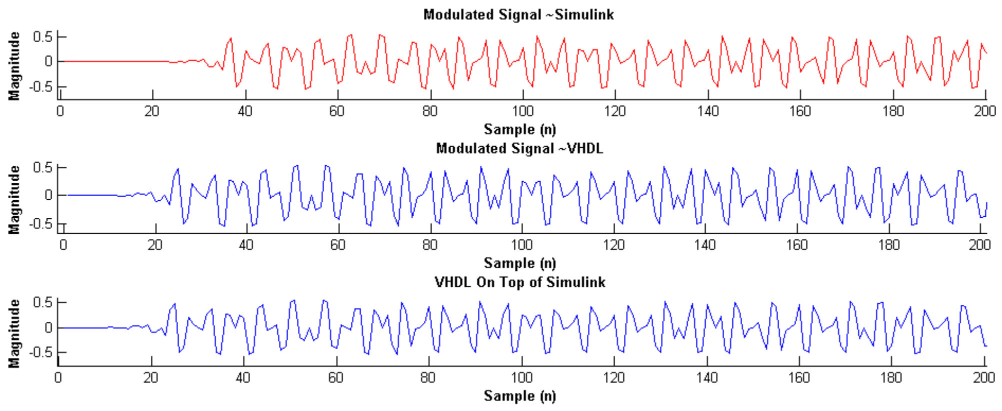


Figure 10 Modulated signal s(t)

For the demodulator to operate correctly the receiver sine and cosine carriers and the modulated signal must be synchronized. Because the I and Q demodulator output closely approximates the model it shows that the signals are properly synchronized.

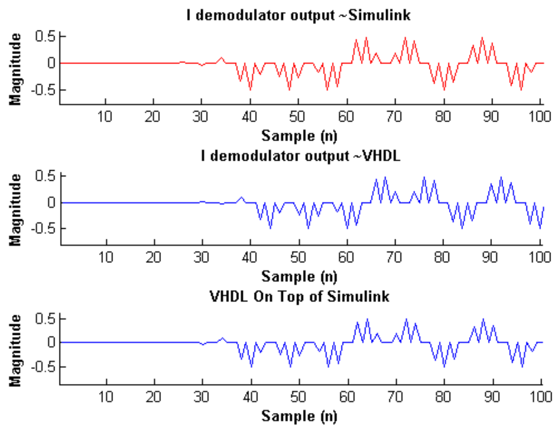


Figure 11 I demodulator waveform

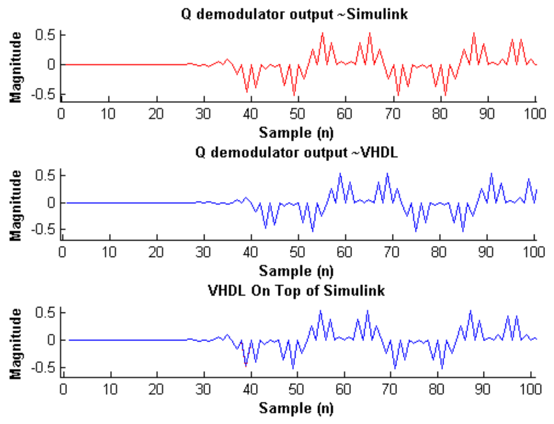


Figure 12 Q demodulator waveform

$\hat{I}(n)$ and $\hat{Q}(n)$ are the most important signals in the receiver. Any error on them will grow inside of the carrier recovery module. Figure 13 and Figure 14 show that there are a few places where the VHDL results deviate from the model slightly. $\hat{I}(n)$ and $\hat{Q}(n)$ could be improved by using more filter coefficients in both the matched filters and the pulse shaping filters. The error for $\hat{I}(n)$ and $\hat{Q}(n)$can be seen in Figure 15.

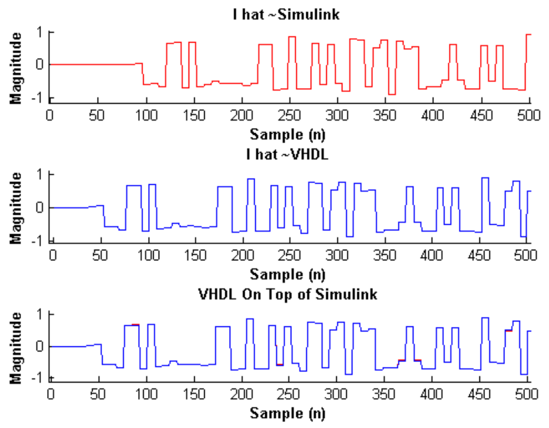


Figure 13 $I_{hat}$ waveform

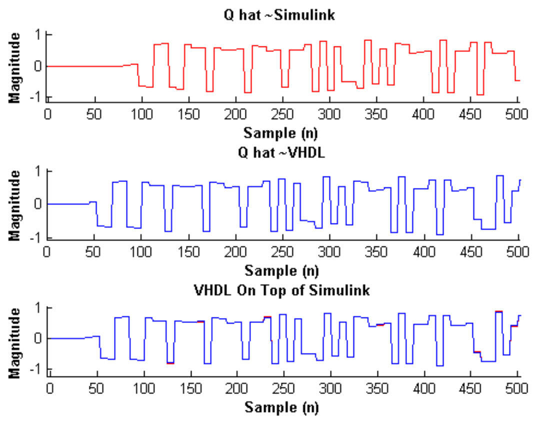


Figure 14 $Q_{hat}$ waveform

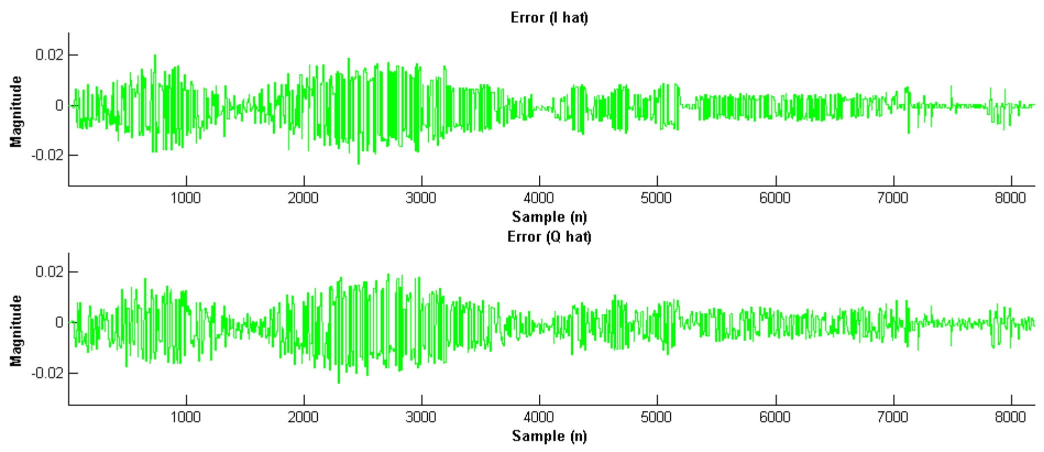


Figure 15 Error in $I_{hat}$ and $Q_{hat}$

Phase error is generated in the carrier recovery module and is used to synchronize the local carrier signals. Figure 17 shows the error in the carrier recovery output, phase error, which has a mean square error of 1.86 x $10^{-11}$.

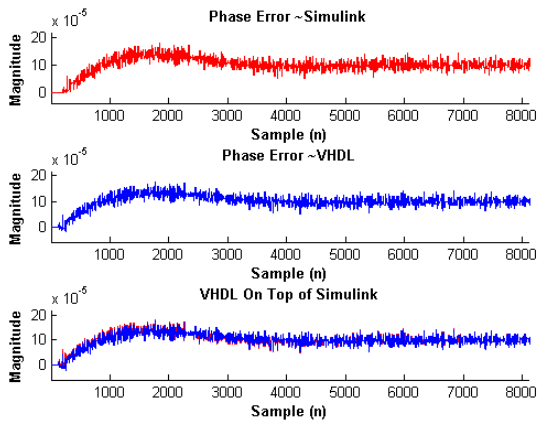


Figure 16 Phase error

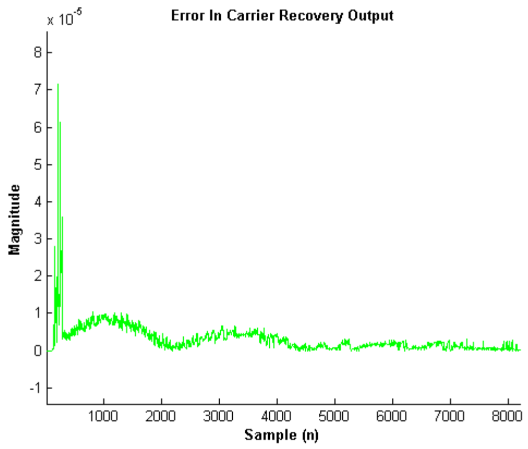


Figure 17 Error in phase error signal

## V. Hardware Results

Figure 18 shows the constellation plot for the transmitter side. Figures 19 and 20 show the constellation plot for the received data in which the transmitter and receiver are on the same FPGA. The difference between these two plots is oscillator offset. In the plot depicted in Figure 19 the offset is 500 Hz which the carrier recovery circuit can handle while the offset in the plot depicted in Figure 20 is 1500 Hz. Because 1500 Hz is outside of the PLL lock range the constellation is rotating. Figure 19 shows that the received symbols are tightly packed around the 4 target points.

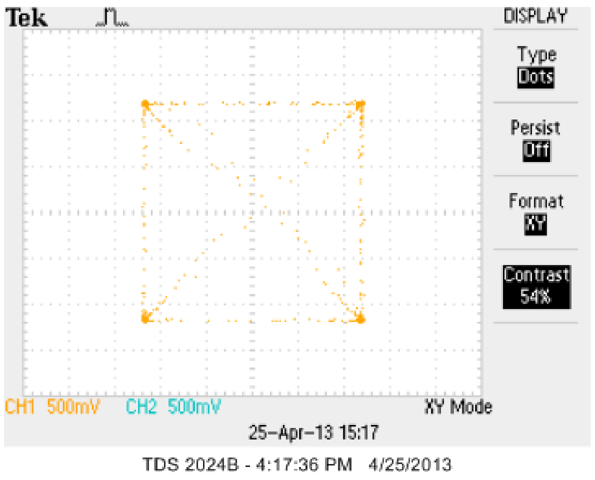
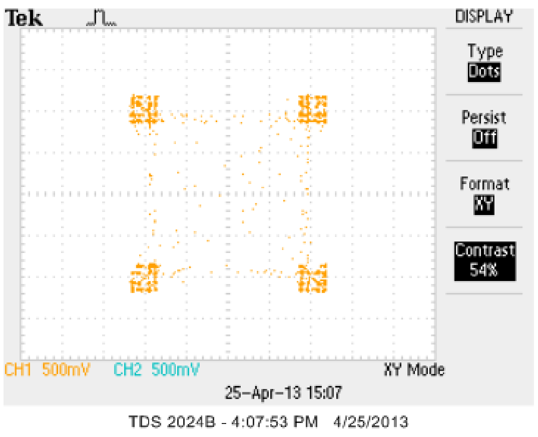


**Figure 18 Transmitted constellation plot**



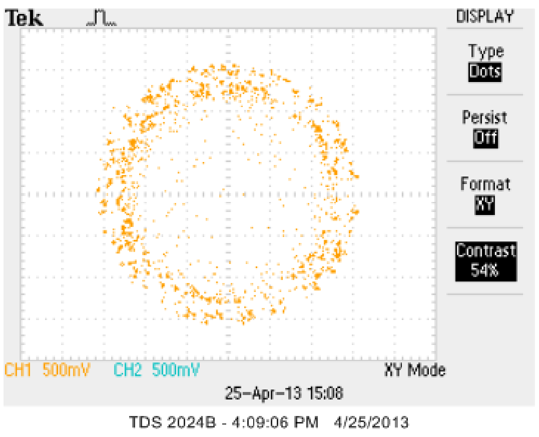
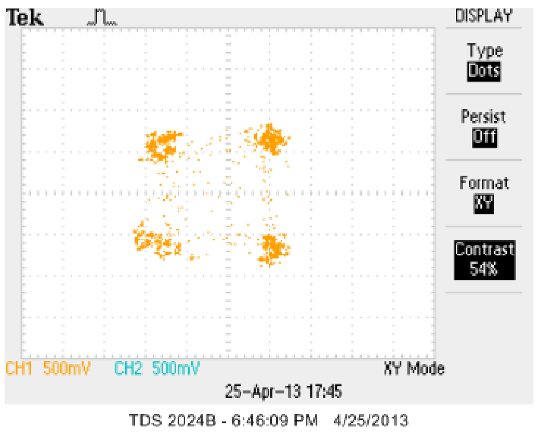


**Figure 19 Received constellation plot on same board** **Figure 20 Received constellation plot frequency out of PLL range**

Figures 21 and 22 show the constellation plot for the received data in which the transmitter and receiver are on separate FPGAs. The difference between these two plots is oscillator offset. In the plot depicted in Figure 21 the offset is 500 Hz which the carrier recovery circuit can handle while the offset in plot depicted in Figure 22 is 1500 Hz. Because 1500 Hz is outside of the PLL lock range the constellation is rotating. Figure 21 shows that the received symbols are not tightly packed around the target points. This scattering could be caused by channel noise and fractional time resolution in the decimation of the upsampled data.



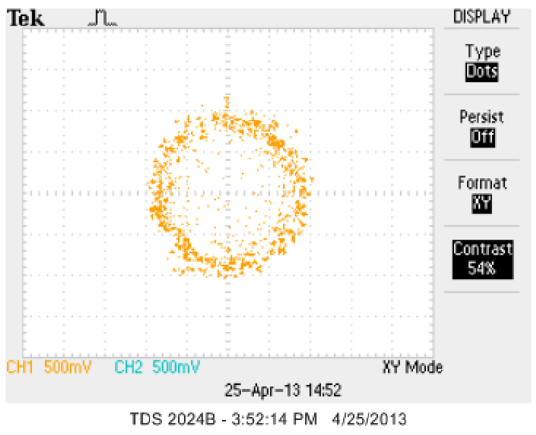


Figure 21 Received constellation plot separate boards

Figure 22 Received constellation plot frequency out of PLL range

The plots in Figure 23 represent the demodulated I and Q data for the four possible states of the constellation. The plot on the left represents no phase ambiguity. The received I and Q data is correct. The plot that is second from left shows 180 degrees of phase ambiguity. With 180 degrees of phase ambiguity both I and Q channels at the receiver side are inverted. The third plot from the left shows 90 degrees of phase ambiguity. In this state I on the received side is the transmitted Q data and Q on the receiver side is the inverse of the transmitted I data. The rightmost plot represents 270 degrees of phase ambiguity in which I on the receiver side is the inverse of the transmitted Q data and Q on the receiver side is the transmitted I data.

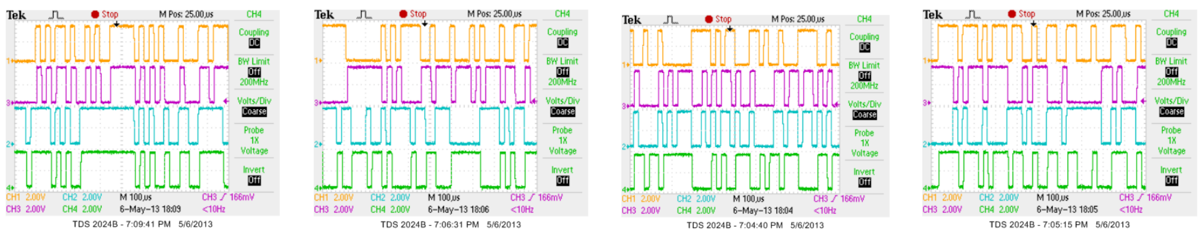


Figure 23 Observed phase ambiguity

Figure24 shows that the received data for both one FPGA and two FPGA setups matches the transmitted data very well once the propagation delay is accounted for.

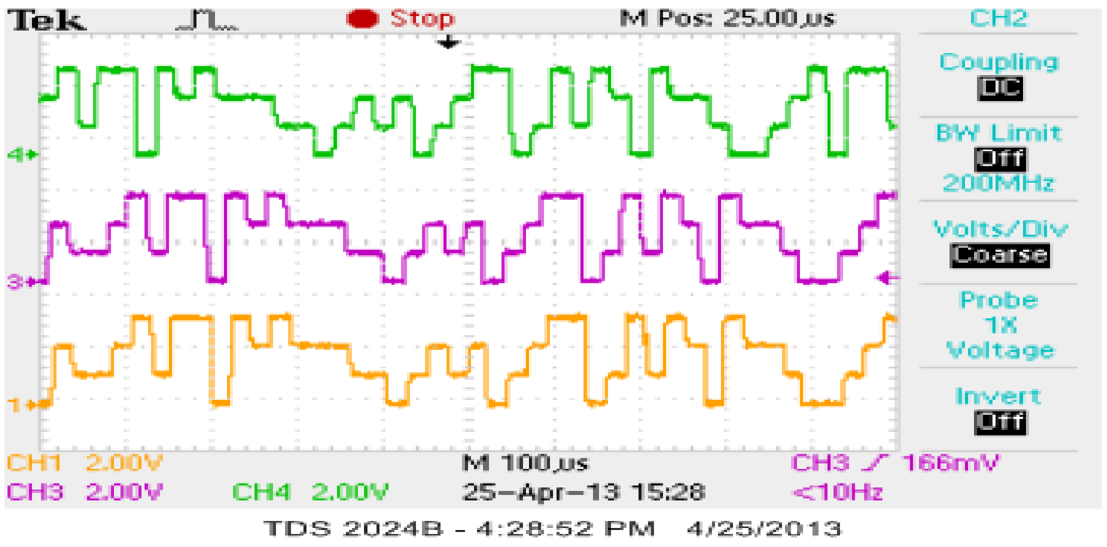


**Figure 24 Transmitted vs. received data (From top to down) a) Transmitted data, b) Received data( transmitter and receiver on the same FPGA board), c) Received data(transmitter and receiver on separate FPGA boards)**

## VI. Conclusion

FPGA design is flexible to build digital communication systems. The methods used for modulation can be reconfigurable which means that new designs can be implemented by simply downloading to the board. In this project a QPSK communication system has been successfully designed and implemented on a pair of Spartan 3E starter boards using VHDL. An efficient verification flow has been applied to the design. Carrier recovery circuit and digital phase locked loop are used to resolve carrier offset which is essential for decoding of the transmitted signal. Channel effects and phase ambiguity were observed on the receiver FPGA.

## VII. REFERENCES

[1] Anton Rodriguez, and Michael Mensinger Jr., "*Software-defined Radio using Xilinx*", Senior Project Report, Department of Electrical and Computer Engineering, Bradley University, Peoria Illinois, May 2011.

[2] Anthony Gaught, "*Software-defined Radio Symbol Generator*", Junior Project Report, Department of Electrical and Computer Engineering, Bradley University, Peoria Illinois, May 2012.

[3] Anthony Gaught, Alexander Norton, and Christopher Brady., "*FPGA-based 16 QAM communication system*", EE 568 Report, Department of Electrical and Computer Engineering, Bradley University, Peoria Illinois, April 2012.

[4] Leon Couch, *"Digital and analog communication systems"*, 8th ed., Boston: Pearson, 2013

[5] Charles Roth Jr., and Lizy John, "Digital systems design using VHDL", 2nd ed., United States: Thomson, 2008.

[6] *Spartan-3E Starter Kit Board Manual*, Xilinx, San Jose, CA, 2009.

## VIII. APPENDIX A

### top_module

This module instantiates all of the second level modules. The delay select process allows the user to add delay to the transmitted signal which will effectively increment the phase ambiguity by 90 degrees. The modulated signal that is sent to the DAC is offset because the hardware is unipolar.

Outputs:

| | | |
|---|---|---|
| modulated_signal | FIX 12_11 | analog signal through a DAC |
| index | std_logic_vector (3 bit) | leds 0-2 |
| ones, tens, hundreds | the number of oscillator offset steps (5 Hz each) | displayed on LCD |

## transmit

This module creates the modulated signal.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |

Outputs:

| | |
|---|---|
| modulated_signal | FIX12_11 |
| ch_one | used to display transmitted I on an oscilloscope |
| ch_two | used to display transmitted Q on an oscilloscope |
| selection_o | outputs data from the symbol generator |

## symbol generator

This module instantiates clk_two, counter, and memory.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |

Outputs:

| | |
|---|---|
| Selection | 4 bit data (only the lower 2 bits are used) |

**clk_two**

This module is a simple clock divider.

Inputs:

| | |
|---|---|
| clk | std_logic |

Outputs:

| | |
|---|---|
| clk_out | std_logic |

**counter**

This module creates a count from 0 to 1023 which is used as an index for the memory module.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |

Outputs:

| | |
|---|---|
| count | unsigned (10 bit) |

**memory**

This module holds 1024 4-bit numbers that were calculated using MATLAB. On a rising edge, data_out takes on the value stored at index.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |

Outputs:

| | |
|---|---|
| index | std_logic_vector (4 bits) |

## symbol_mapper

This module uses the top bit to create I and the bottom bit to create Q.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |
| i_q_select | std_logic_vector (2 bits) |

Outputs:

| | |
|---|---|
| i_t_symbol | used to display transmitted I on an oscilloscope |
| q_t_symbol | used to display transmitted Q on an oscilloscope |
| i_t | FIX_12_11 |
| q_t | FIX_12_11 |

## interpolator

This module instantiates pulse_shaper_filter. First the data is up sampled and then passes through the FIR pulse shaping filter. The lowest 8 bits of the filter output is discarded.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |
| data_in | FIX_12_11 |

Outputs:

| | |
|---|---|
| data_out | FIX_16_15 |

## pulse_shaping_filter

This module is the implementation o a FIR filter.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |
| Xin | FIX_12_11 |

Outputs:

| | |
|---|---|
| Yout | FIX_24_23 |

## direct_digital_synthesizer

This module creates the sine and cosine carriers which are made up of 512 points that were created in MATLAB. In the transmitter side the off_set is set equal to 0.25 while it is 0.25 - phase_error on the receiver side.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |
| off_set | FIX_32_31 |

Outputs:

| | |
|---|---|
| carrier_cos | FIX_12_11 |
| carrier_sin | FIX_12_11 |

**modulator**

This module creates the modulated signal.

Inputs:

| | |
|---|---|
| Clk | std_logic |
| Reset | std_logic |
| i_t_filt | FIX_16_15 |
| q_t_filt | FIX_16_15 |
| carrier_cos_t | FIX_12_11 |
| carrier_sin_t | FIX_12_11 |

Outputs:

| | |
|---|---|
| modulated_signal | FIX_12_11 |

**recieve**

This module demodulates the transmitted signal and displays the results on an oscilloscope.

Inputs:

| | | |
|---|---|---|
| Clk | std_logic | |
| Reset | std_logic | |
| phase_adjustment | FIX_28_27 | user defined phase error signal |

| | | |
|---|---|---|
| modulated_signal | FIX_12_11 | |

Outputs:

| | | |
|---|---|---|
| modulated_signal | FIX_12_11 | |
| ch_one | used to display received I on an oscilloscope | |
| ch_two | used to display received Q on an oscilloscope | |
| index_o | used to display the down sampling index | |
| phase_error_o | used to display phase error on an oscilloscope | |
| selection | std_logic_vector (2 bits) | received data |

**demodulator**

This module demodulates the transmitted signal. The index signal is the down sampling index which is calculated in the downsampler module. It is used here to adjust the required delay to eliminate phase ambiguity. This method works with the single FPGA setup but it does not work with 2 FPGAs. I believe that minor tweaking of the algorithm that calculates the index position would fix the issue for the implementation of 2 FPGA boards.

Inputs:

| | |
|---|---|
| Clk | std_logic |
| Reset | std_logic |
| index | std_logic_vector (3 bits) |
| modulated_t_full | FIX_12_11 |
| carrier_cos_r | FIX_12_11 |
| carrier_sin_r | FIX_12_11 |

Outputs:

| | |
|---|---|
| i_r_resize | FIX_12_11 |
| q_r_resize | FIX_12_11 |

**decimator**

This module passes the data through a matched filter and then it is down sampled. The second most significant bit and the least 7 significant bits are discarded after filtering. This module uses the same pulse shaping filters that are used in the transmitter.

Inputs:

| | |
|---|---|
| Clk | std_logic |
| Reset | std_logic |
| data_in | FIX_12_11 |

Outputs:

| | | |
|---|---|---|
| data_out | FIX_16_15 | I_hat or Q_hat |
| index_o | std_logic_vector (3 bits) | down sampling index |

**downsampler**

This module finds the best point to down sample the data. It accomplishes this goal by down sampling at all 8 possible points and searches for the minimum value. Once the minimum position is found the index is set to the minimum position + 4.

Inputs:

| | | |
|---|---|---|
| Clk | std_logic | |
| Reset | std_logic | |
| data_in | FIX_24_22 | Full data precision from the filter |

Outputs:

| | | |
|---|---|---|
| index | std_logic_vector (3 bits) | down sampling index |

**carrier_recovery**

This module uses I_hat and Q_hat to generate phase_error which is used to correct for carrier offset.

Inputs:

| | |
|---|---|
| Clk | std_logic |
| Reset | std_logic |
| I_hat | FIX_16_15 |
| Q_hat | FIX_16_15 |

Outputs:

| | |
|---|---|
| phase_error | FIX_32_31 |

**rece_decision**

This module uses I_hat and Q_hat to generate phase_error which is used to correct for carrier offset.

Inputs:

| | | |
|---|---|---|
| clk | std_logic | |
| reset | std_logic | |
| i_r | FIX_16_15 | |
| q_r | FIX_16_15 | |

Outputs:

| | | |
|---|---|---|
| i_r | std_logic | decoded I data |
| q_r | std_logic | decoded Q data |
| phase_error | FIX_32_31 | |
| i_r_adj | FIX_12_11 | signal to be displayed on an oscilloscope |
| q_r_adj | FIX_12_11 | signal to be displayed on an oscilloscope |

**decision_module**

This module creates a multilevel signal which can be displayed on an oscilloscope from the transmitted and received I and Q bits.

Inputs:

| | | |
|---|---|---|
| clk | std_logic | |
| reset | std_logic | |
| t_select | std_logic_vector (2 bits) | |
| r_select | std_logic_vector (2 bits) | |

Outputs:

| | | |
|---|---|---|
| t_data | FIX_12_11 | signal to be displayed on an oscilloscope |

| | | |
|---|---|---|
| r_data | FIX_12_11 | signal to be displayed on an oscilloscope |

## output_interface

This module takes in several signals and displays the user's choice on an oscilloscope using a DAC. Another DAC is used to transmit the modulated signal on the FPGA that contains both transmitter and receiver. The FPGA that contains only the receiver use an ADC to receive the signal instead of the second DAC.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |
| t_select | std_logic_vector (3 bits) |
| one_in | 12 bit data |
| two_in | 12 bit data |
| three_in | 12 bit data |
| four_in | 12 bit data |
| five_in | 12 bit data |
| six_in | 12 bit data |
| seven_in | 12 bit data |
| eight_in | 12 bit data |

Outputs:

| | |
|---|---|
| one_out | DAC channel |
| two_out | DAC channel |
| three_out | DAC channel |
| four_out | DAC channel |
| dac_en | DAC enable signal |
| clk_out | DAC clock signal |

**mux**

This module is a 5-channel 12-bit mux.

Inputs:

| | |
|---|---|
| sel | std_logic_vector (3 bits) |
| a | 12 bit data |
| b | 12 bit data |
| c | 12 bit data |
| d | 12 bit data |
| e | 12 bit data |

Outputs:

| | |
|---|---|
| data_out | 12 bit data |

**dac_module**

This module instantiates a clock divider and dac_control. The clock divider creates a clock which is used inside of dac_control and is also sent to the DAC hardware.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |
| one_in | 4 bit control bits and 12 data bits |
| two_in | 4 bit control bits and 12 data bits |

Outputs:

| | |
|---|---|
| one_out | std_logic |
| two_out | std_logic |
| clk_out | std_logic |

**dac_control**

This module takes in a 16 bit number of which the most 4 significant bits are control bits. The remaining 12 bits are data bits which determine the analog output of the DAC.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |
| one_in | 4 bit control bits and 12 data bits |
| two_in | 4 bit control bits and 12 data bits |

Outputs:

| | |
|---|---|
| one_out | std_logic |
| two_out | std_logic |
| finished | std_logic (used as an enable signal for the hardware) |

**adc_module**

This module instantiates a clock divider and adc_control. The clock divider creates a clock which is used inside of adc_control and is also sent to the ADC hardware.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |
| one_in | 4 bit control bits and 12 data bits |
| two_in | 4 bit control bits and 12 data bits |

Outputs:

| | |
|---|---|
| one_out | std_logic |
| two_out | std_logic |
| clk_out | std_logic |

## adc_control

This module takes in a serial transmission from the ADC hardware and converts it into a 12 bit number.

Inputs:

| | |
|---|---|
| clk | std_logic |
| reset | std_logic |
| one_in | std_logic |
| two_in | std_logic |

Outputs:

| | |
|---|---|
| one_out | 12 bits |
| two_out | 12 bits |
| finished | std_logic (used as an enable signal for the hardware) |

**rotary_encoder**

This module takes input from the user and sets the count_out which is used to adjust the receiver oscillator. Ones_out, tens_out, and hundreds_out represent the number of steps the oscillator is adjusted and is displayed on the LCD.

Inputs:

| | | |
|---|---|---|
| clk | std_logic | |
| reset | std_logic | |
| rotary_A | std_logic | |
| rotary_B | std_logic | |
| rotary_center | std_logic | not used in the design |

Outputs:

| | |
|---|---|
| ones_out | 4 bit data |
| tens_out | 4 bit data |
| hundreds_out | 4 bit data |
| sign_out | std_logic |
| count_out | FIX_28_28 |