Contents

1 Introduction

This example design presents a complete QPSK transceiver example design. This design involves two radio cores: the transmitter (tx) and receiver (rx) cores. This tutorial will discuss the theory of operation behind the design, the usage of the HDL Coder tool for IP core development, and the overall design of the QPSK transceiver.

2 Theory of Operation

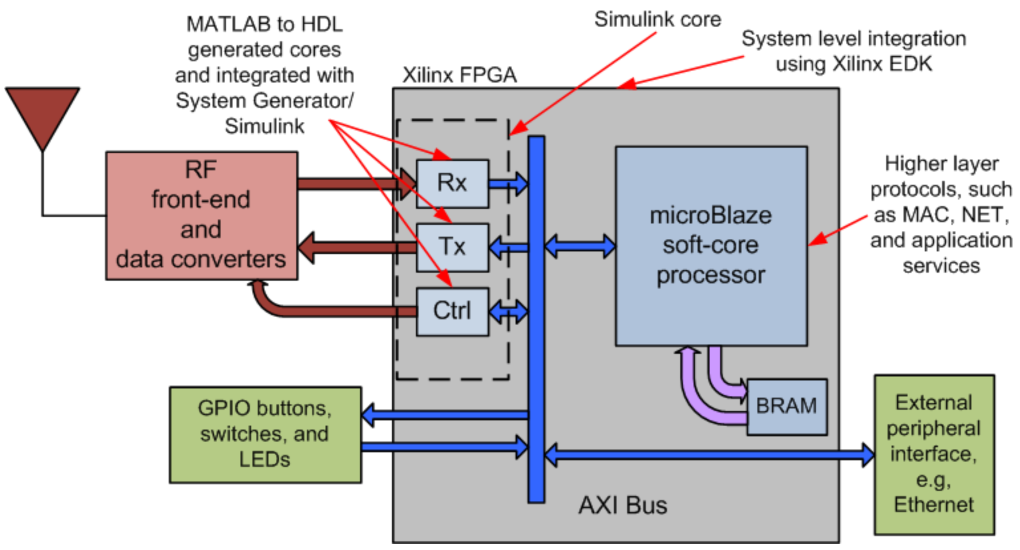
MATLAB to HDL generated cores and integrated with System Generator/ Simulink
Simulink core
System level integration using Xilinx EDK
Xilinx FPGA
RF front-end and data converters
Rx
Tx
Ctrl
microBlaze soft-core processor
Higher layer protocols, such as MAC, NET, and application services
BRAM
GPIO buttons, switches, and LEDs
External peripheral interface, e.g, Ethernet
AXI Bus

Figure 1. Block diagram showing system-level integration of tx/rx cores in a Xilinx FPGA.

The figure above shows the system-level operation of the QPSK transceiver presented here. The Tx/Rx cores, made by HDL Coder, are interfaced with the embedded microBlaze soft-core processor internally within the FPGA for data processing. These Tx/Rx cores are not only interfaced with the embedded processor, but also with the Toyon Chilipepper RF front-end for transmitting and receiving signals. This is how the designs discussed here interface to the radio side of the world, as well as the processor needed for more complex computations. This section will discuss the theory behind design of these cores, and how they interact with the rest of the system.

2.1 Quadrature Phase Shift Keying

Quadrature phase-shift-keying (QPSK) is a digital phase modulation scheme in which the carrier signal undergoes four changes in phase, representing four symbols. The four symbols are generated by specifying the amplitude of the in-phase (I) and quadrature (Q) components of the carrier. I, the in-phase component of the signal, is modulated on the basis function

$$\phi_I(t) = \sqrt{\frac{2}{T_s}}\cos(2\pi f_c t)$$

Q ,the out-of-phase (quadrature) component of the signal, is modulated on the basis function

$$\phi_Q(t) = -\sqrt{\frac{2}{T_s}}\sin(2\pi f_c t)$$

where T_s is the symbol period, and f_c is the carrier frequency.

A QPSK modulator begins by demultiplexing the outgoing bit sequence to form these two independent binary messages, $I(t)$ and $Q(t)$. Modulation is then achieved by varying the amplitude of the basis functions depending on the message symbols. The transmitted QPSK signal can therefore be expressed as:

$$s_n(t) = \ I(t)\cos(2\pi f_c t + \phi_i) - \ Q(t)\sin(2\pi f_c\, t + \phi_i)$$

where ϕ_i is the phase offset at the transmitter.

Since $I(t)$ and $Q(t)$ can take one of two values, four phases (four symbols) can be produced. Notice that each symbol encodes two bits, the value of $I(t)$ and $Q(t)$ at time t. This is one advantage of QPSK: that it enables a carrier to transmit 2 bits of information per symbol instead of 1, effectively doubling the bandwidth of the carrier.

The QPSK signal can also be expressed as

$$s_n(t) = \sqrt{\frac{2}{T_s}}\,cos\left(2\pi f_c t + (2n-1)\frac{\pi}{4}\right), \qquad n = 1,2,3,4$$

The different values of n yield the four phases used in QPSK: $\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}$. In this form, QPSK is clearly shown to be a form of phase-shift-keying.

2.2 Packet Creation

The first step in implementation of this transceiver is creating the packet to be transmitted. In this example, the microprocessor creates the packets and generates the CRC. The packet starts off with a 32-bit header containing the length of the payload (the message to be transmitted). The payload ("hello world!") comes next, followed by a 16-bit CRC appended at the end for error detection.

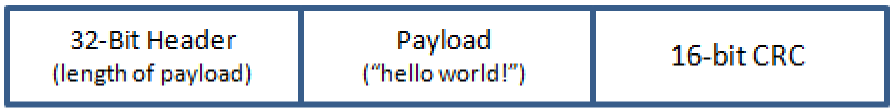

Figure 2. Structure of packet created by the microprocessor.

2.2.1 CRC

A CRC, or cyclic redundancy check, is an error-detecting code often used for detecting accidental changes to transmitted data. The CRC's value is derived from the remainder of a polynomial division between the packet's contents and an n-bit binary generator polynomial. Specification of a CRC is given by the definition of its generator polynomial, or check value. This generator polynomial assumes the role of the divisor, and takes the payload as the dividend. In logic, the division is performed left to right through a series of logical xor's on each binary "1" in the packet.

During packet creation, this polynomial division is done, and the n-1 bit CRC is formed and appended to the end of the packet.

The CRC is one bit shorter than the generator polynomial for error checking on the receiver side. Given same generating polynomial, the polynomial division should return a remainder of 0 if the contents of the packet have remained unchanged.

2.3 Transmitter

Once the packet has been created, it enters the transmitter core.

2.3.1 QPSK Modulation of Bytes to Symbols

The transmitter first forms the two binary messages.

Before handling the packet data, the transmitter addresses the issues of channel estimation and frame synchronization by inserting a 65-bit training sequence at the front of both messages. Here, the small set of Kasami sequences is used for its good cross-correlation properties. These training sequences are generated using objects in MATLAB's Communication Toolbox, then synthesized into hardware as LUTs (look-up tables).

The transmitter core then begins to load packet data from the processor into the internal buffer, and convert the bytes to symbols. The core accepts packet data from the processor and demultiplexes the data into its odd (I) and even (Q) bits. After creating these two components, the data is encoded with bipolar NRZ encoding—a non-return-to-zero line code which represents 0 as one physical level, i.e. -1, and 1 as another, i.e. +1. This bipolar NRZ encoding protects against DC offset errors during transmission, helping the message to be decoded properly.

The I and Q components are then modulated back together to form the transmitted symbol, with every 2 bits (one from $I(t)$ and one from $Q(t)$) forming one outgoing symbol, $s_n(t)$, as described above. The two bits define the real and imaginary parts of each outgoing symbol, thereby determining the symbol's phase. The four symbols can be visualized in the signal constellation of QPSK, which plots the real part $I(t)$ and imaginary part $Q(t)$ of each QPSK symbol.

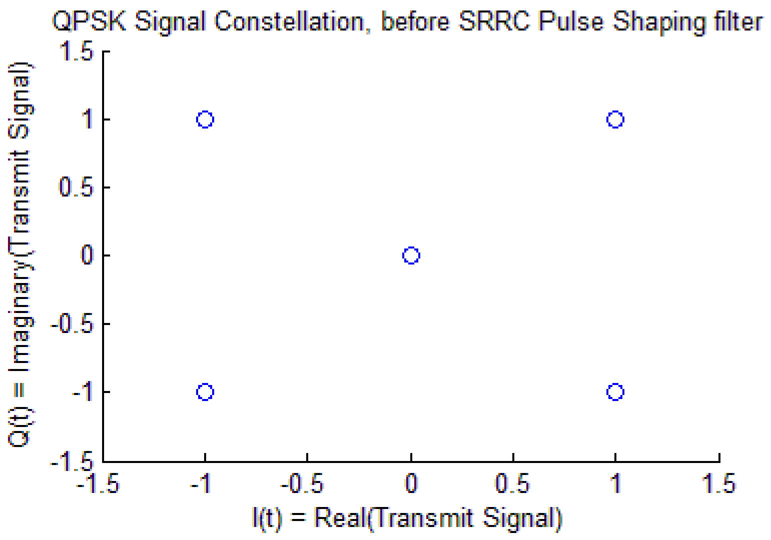

Figure 3. QPSK Constellation of transmit sequence before pulse shaping, and after NRZ encoding. NRZ encoding results in the points being placed at -1 and 1, instead of 0 and 1.

After translating bytes to symbols, the symbols are oversampled at an oversampling rate of 8. Oversampling the symbols by a factor of 8, when each symbol encodes 2 bits, results in an oversampling of 4 samples per bit. The oversampling process serves to provide multiple sampling points on the received waveform such that the data can be sampled near the maximum amplitude at the receiver.

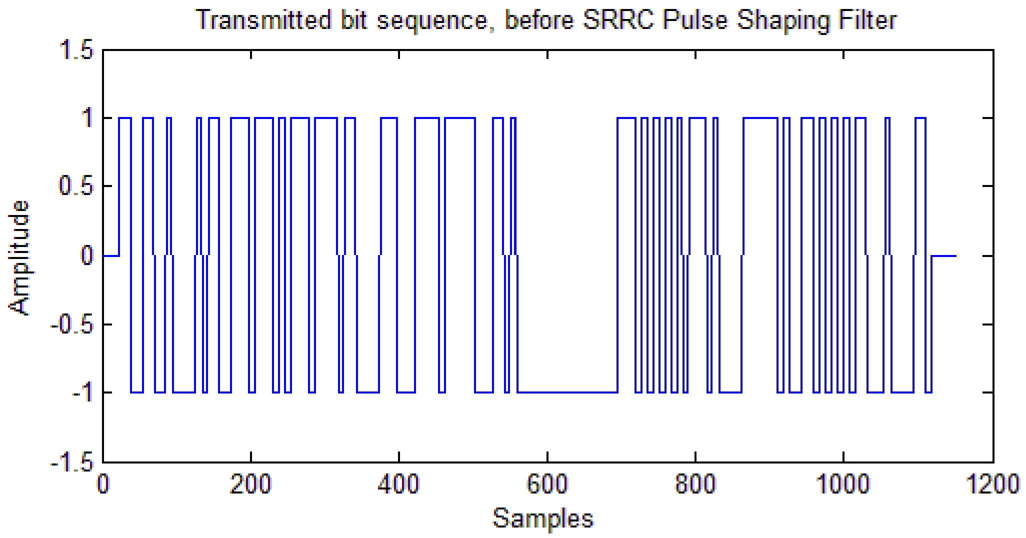

Figure 3. Transmitted bit sequence after QPSK Modulation, oversampling and NRZ encoding—before pulse shaping. NRZ encoding allows signal to swing from -1 to +1, rather than 0 to 1. The first 2*65*4 bits are from the 65-bit training sequence, added to the front of the both I and Q components. The next 32*4 bits form the header, containing the length of the payload. The following 12(message length)*8(# bits per decimal number)*4(oversampling) bits contain the binary translation of the message. The last 16*4 bits contain the CRC.

After oversampling, the transmitter convolves the outgoing bit sequence with a pulse shaping filter to control the spectrum of the transmission. The idea behind pulse shaping is to limit the effective bandwidth of the transmission, allowing the signal to fit bandwidth constraints. A frequently used filter in digital communication systems, including the filter used in this example, is a square root raised cosine filter.

2.3.2 SRRC Pulse Shaping Filter

There are at least two compelling reasons to use SRRC filters:

1. The net frequency response of the SRRC filters at the transmit and receive sides combine to form a Raised Cosine filter.
2. The equal distribution of the filtering between transmit and receive achieves matched filtering, which maximizes SNR (signal-to-noise ratio) and minimizes BER (bit error rate).

Raised Cosine filters are commonly used for pulse shaping due to their ability to band-limit a signal, while avoiding ISI. The impulse response of the Raised Cosine filter is zero at all times $t = nT_s$, except $n = 0$, because it is designed to avoid intersymbol interference (ISI). Therefore, if the transmission is sampled correctly at the receiver, the original symbol values can be recovered completely.

In digital communications, a practical implementation of Raised Cosine filtering is to split the filtering between the transmitter and receiver. This is done by applying an SRRC filter on both the transmit and receive sides.

Mathematically, the impulse response of a SRRC filter $H_{SRRC}(t)$ can be expressed as:

$$H_{srrc}(t) = \frac{\sin\left[\pi \frac{t}{T_s}(1-\beta)\right] + 4\beta \frac{t}{T_s} cos\left[\pi \frac{t}{T_s}(1+\beta)\right]}{\pi \frac{t}{T_s}\left[1-\left(4\beta \frac{t}{T_s}\right)^2\right]}$$

where T_s is the symbol period, and β is the roll-off factor. These two values characterize the frequency response of the SRRC filter.

The roll-off factor, β, is a measure of the excess bandwidth of the filter, i.e. the bandwidth occupied beyond the Nyquist bandwidth of $\frac{1}{2T_s}$. It takes values $0 \leq \beta \leq 1$, and is typically expressed as a percentage of the bandwidth. The bandwidth occupied by the raised cosine filter, or equivalently the matched SRRC filters, is typically taken as the non-zero portion of its frequency spectrum.

The impulse response of an SRRC filter is also the square root of the response of a raised-cosine filter, $H_{RC}(f)$:

$$|H_{srrc}(f)| = \sqrt{|H_{RC}(f)|}$$

Implementation of the transmit filter consists of convolving the outgoing symbols at the transmitter with the transmit SRRC filter. The signal is then sent through the channel and convolved with the matched receiver SRRC filter, producing the frequency response of a Raised Cosine filter:

$$H_{rc}(f) = H_{T,srrc}(f) * H_{R,srrc}(f)$$

where $*$ denotes convolution. The formation of a Raised Cosine filter allows elimination of ISI.

In addition to eliminating ISI, the SRRC filters are also chosen for their properties as matched filters. There exists an optimum filter for each transmitted pulse shape—what is called a matched filter—which filters out additive, uncorrelated noise, thus giving the maximum signal-to-noise ratio (SNR) and lowest BER (bit error rate). The impulse response of this ideal filter is actually just a reversed and time delayed copy of the transmitted signal. The equal distribution of filtering between the transmitter and receiver fit this profile, and achieves matched filtering, therefore providing the maximum SNR, and lowest BER.

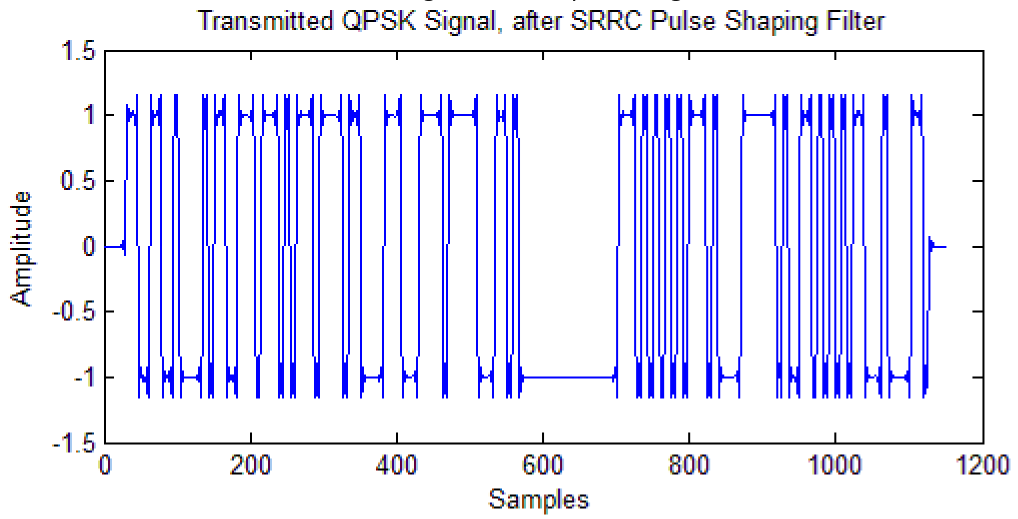

Figure 4. Transmitted QPSK signal after Pulse Shaping. The previously rectangular waveforms are now smoothed out around the edges, bandlimiting the signal, while avoiding ISI.

Coefficients of the SRRC filter are designed using MATLAB's Signal Processing Toolbox. The coefficients are generated prior to synthesis, and loaded onboard the FPGA as LUTs.

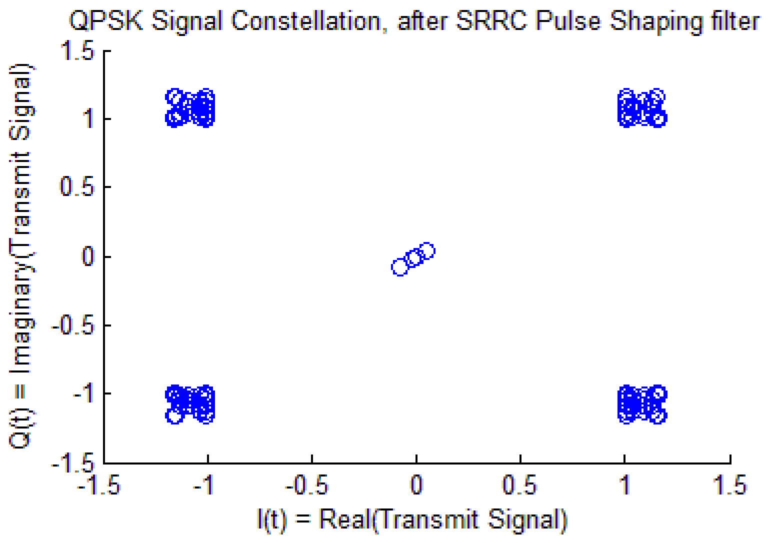

Figure 5. Signal constellation of transmitted QPSK signal, after pulse shaping filter. The data shown here is sampled at the times of maximum amplitude.

An oscillator at the transmitter provides a sinusoidal carrier signal which is multiplied by the QPSK data to modulate the signal up to a passband center frequency. For simulation purposes, this frequency offset is applied in the channel.

2.4 Channel

In communication systems, the channel refers to the transmission medium through which information is sent. In this example, a physical channel is emulated in the project's test bench. The transmitted signal is padded on either side with zeros, to simulate the timing delays of a physical channel. The transmission is then corrupted by additive, white, Gaussian noise (AWGN) with the desired SNR, to model the timing errors caused by random jitter in physical channels.

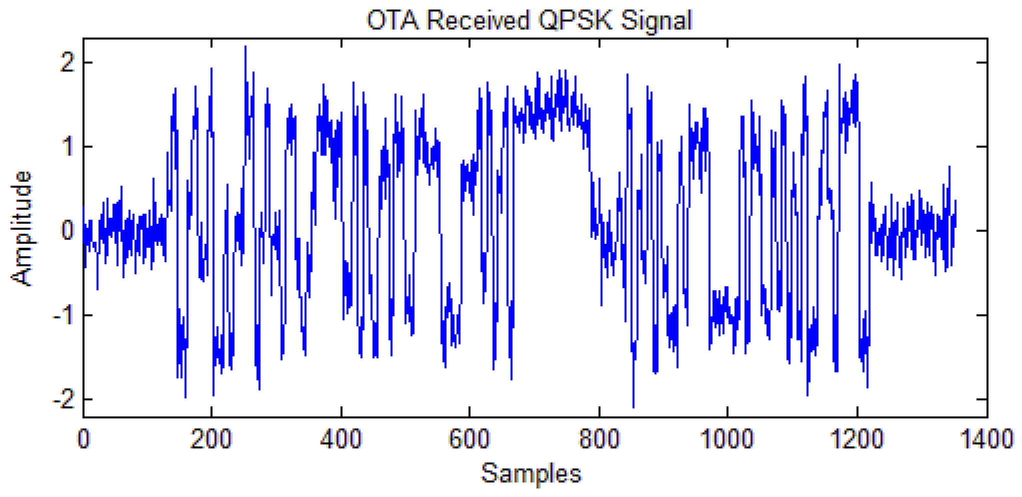

Figure 6. Over-the-air (OTA) QPSK received signal, after corruption by AWGN channel, with 10db SNR.

2.5 Receiver

After transmission through the channel, the signal is received and enters the receiver core. The received signal can be described mathematically as follows:

$$r(t) = I(t)\cos(2\pi f_c t + \phi_i) - Q(t)\sin(2\pi f_c\, t + \phi_i) + w(t)$$

where T_s is the symbol period, f_c is the carrier frequency, ϕ is the phase offset at the transmitter, and $w(t)$ is the AWGN received during OTA transmission.

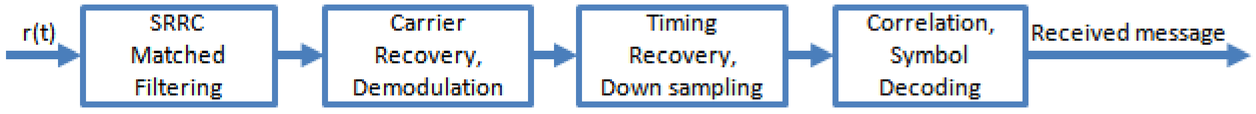
r(t)
SRRC Matched Filtering
Carrier Recovery, Demodulation
Timing Recovery, Down sampling
Correlation, Symbol Decoding
Received message

Figure 7. Overview of cores in the receiver chain.

The receiver first performs matched filtering on the signal, as described above. The corrupted received signal is convolved with the matched SRRC filter.

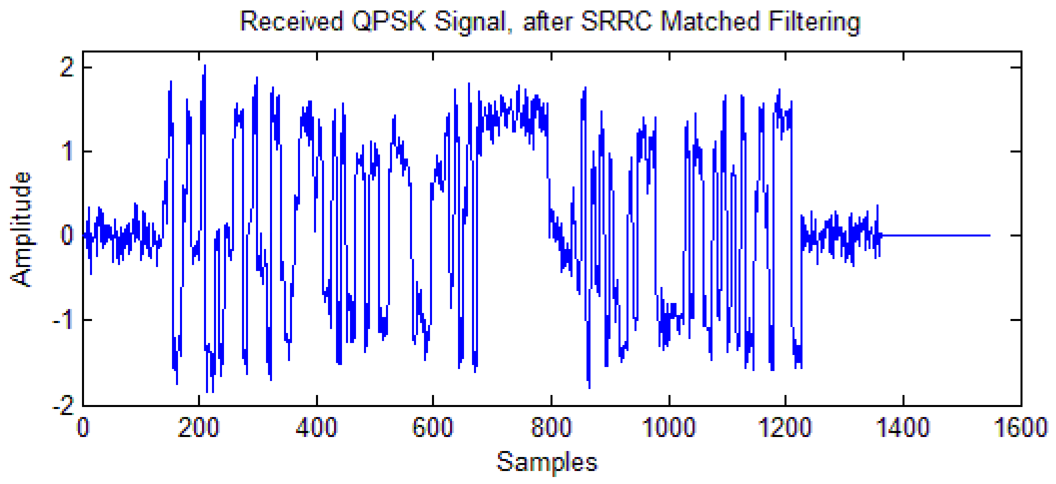

Figure 8. Received QPSK signal, after receive filtering with an SRRC matched filter.

2.5.1 Costas Loop – Carrier Recovery and QPSK Demodulation

The filtered signal is sent to a phase-locked loop (PLL) for carrier recovery. Ideally, the transmitted carrier frequency is known at the receiver, and only its phase must be recovered for proper demodulation. However, due to oscillator drift at the transmitter, the actual carrier frequency will deviate slightly from the expected frequency. The presence of this frequency offset causes the signal constellation to rotate, as shown in Figure 8.

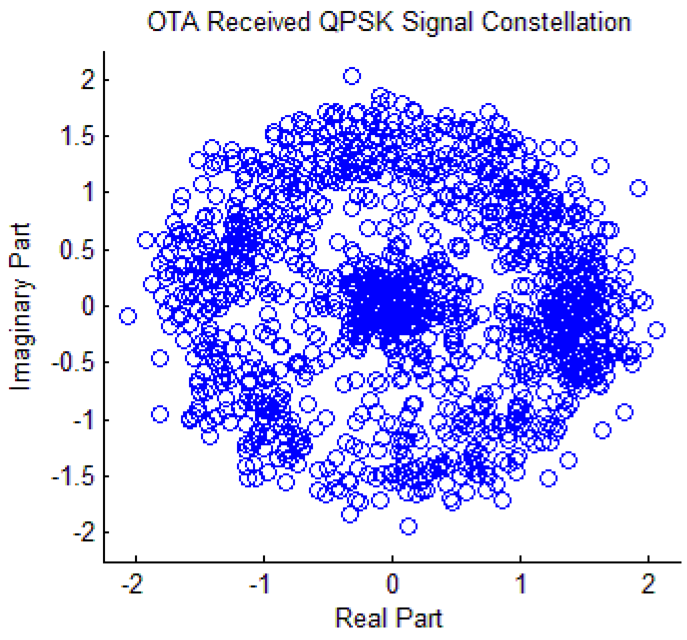

Figure 9. OTA Received signal constellation, after corruption by AWGN channel, with 10db SNR.

The carrier recovery aims to remove this rotation from the received signal constellation, to allow accurate symbol decisions to be made. As seen in this example, the Costas Loop is used to remove this frequency offset so that the signal can be directly processed at baseband.

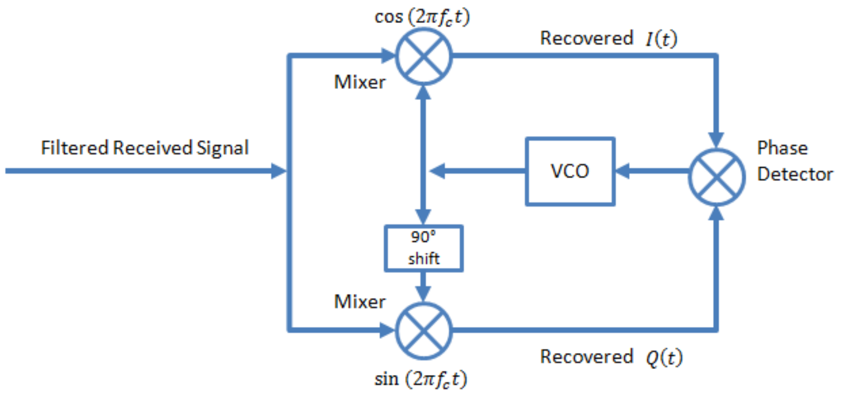

Figure 10. Costas Loop Block Diagram.

The local voltage-controlled oscillator (VCO) at the receiver is synthesized as two LUTs, containing the functions $\cos(2\pi f_c t)$ and $\sin(2\pi f_c t)$. The received signal is first demodulated in two paths: one which mixes $r(t)$ with a cosine, the other which mixes with a sine. The demodulated outputs are then fed to a phase detector—an element which generates a voltage proportional to the phase difference between the in-phase and quadrature-phase inputs. This voltage serves as the control signal for precise adjustment of the VCO frequency. This process operates recursively, allowing the Costas Loop match the local oscillator frequency to that of the carrier.

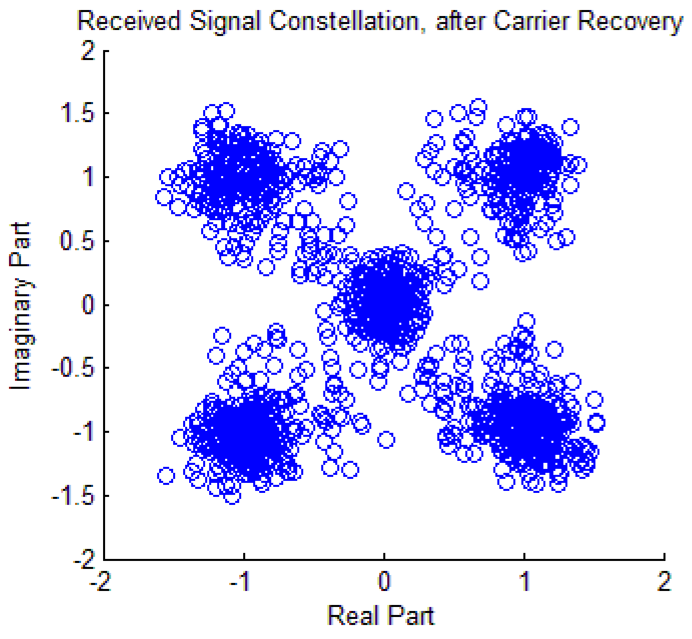

Figure 11. Signal Constellation, after Costas Loop. Due to carrier recovery, the signal constellation has been significantly cleaned up, with no apparent signs of rotation.

A seen from Figure 10 above, there is no more rotation in the signal constellation after the Costas Loop has been applied; this is because the tuning of the receive oscillator to the carrier frequency has removed the frequency offset from the demodulated baseband signal. Because the frequency offset has been corrected for, the receive signal has successfully undergone QPSK demodulation into the message signals $I(t)$ and $Q(t)$.

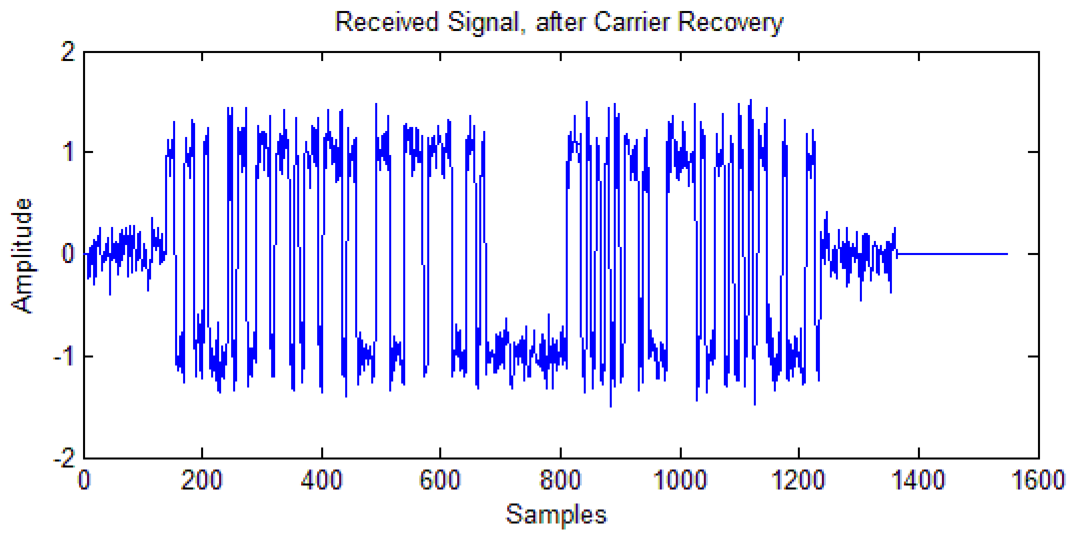

Figure 12. Received Signal after passing through Costas Loop for Carrier Recovery.

2.5.2 Timing Recovery

The next step is to sample the $I(t)$ and $Q(t)$ waveforms at the symbol rate and decide which symbols were transmitted. But before sampling the waveform, timing recovery is employed to determine the optimal times for sampling. The received waveform is down sampled using the timing offset estimate, to provide an even cleaner signal constellation, which is ready for symbol decisions.

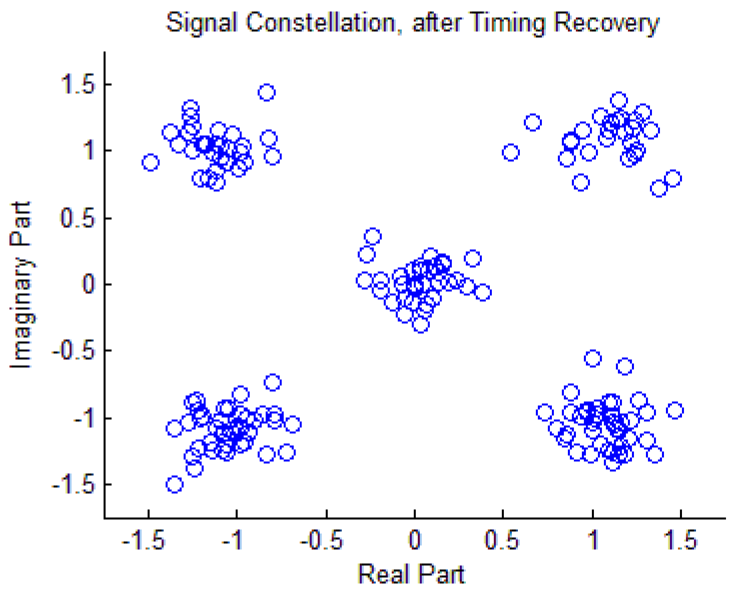

Figure 13. Signal Constellation after timing recovery. The symbols have been optimally sampled, as evidenced by the more concise nature of the constellation.

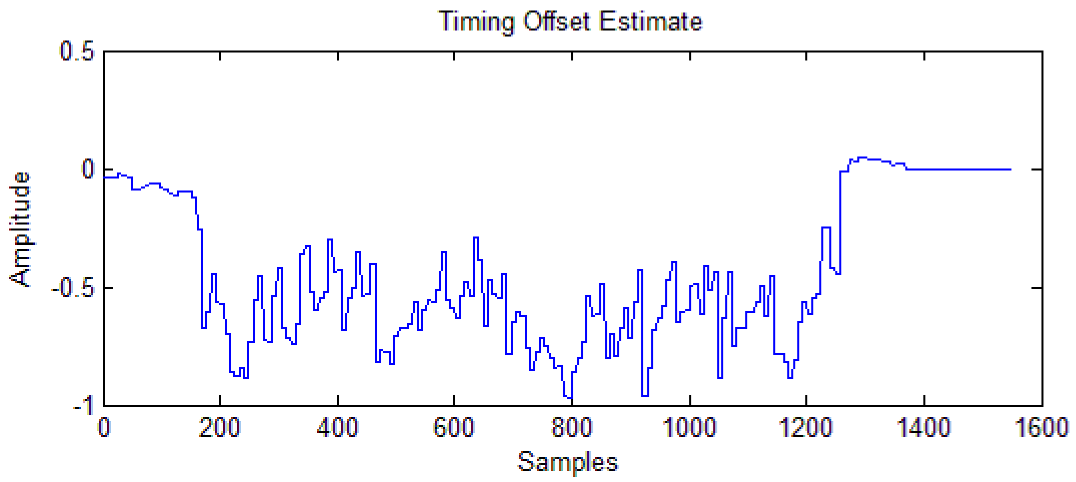

Figure 14. Timing Offset Estimate from noise introduced in the AWGN channel. Notice how the zero padding done in the channel to the first and last 100-bits simulates a time delay.

2.5.3 Correlator

Cross-correlation between the training sequence and the processed signal is then performed to find the packet data within the received waveform. A glance at the magnitude of the cross-correlation function immediately reveals where the packet data lies within the received signal.

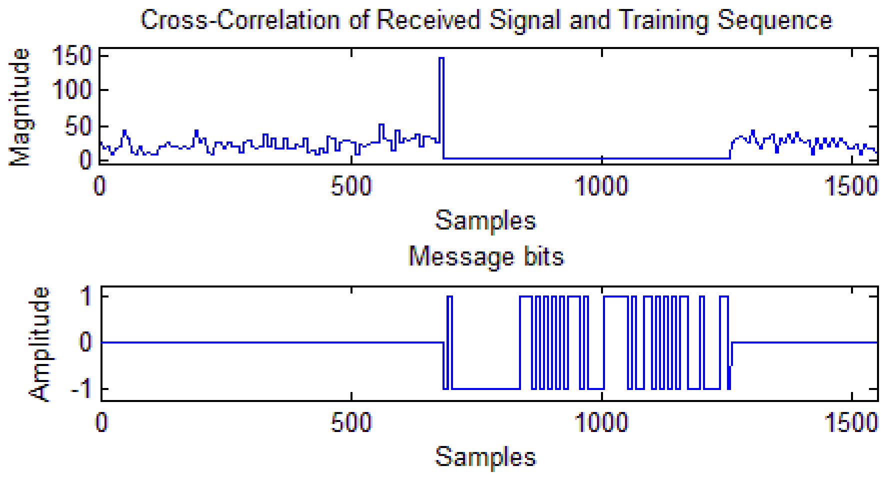

Figure 15. Magnitude of Cross-Correlation function between Received Signal and Training Sequence. Good correlation during training sequence, a peak in correlation at the start of the message bits, and no correlation between the training sequence and message bits. From the cross-correlation of the two signals, it is clear where the packet is in the received signal.

The cross-correlation of the received signal and training sequence compares the data from the two, and returns the magnitude of the correlation between the two signals. This is used to determine where the packet sits within the received signal. As seen from Figure 14, there is a peak in the cross-correlation that indicates the start of the packet, and then a flat 0 throughout the packet, where it is clear that the packet is being transmitted.

2.5.4 Decoding

As the cross-correlation figures out where the packet data is being sent, the symbols in the packet are mapped back into bits. The translation from symbols to bits is based mostly on the symbol's place in the signal constellation. The symbols are decoded as follows:

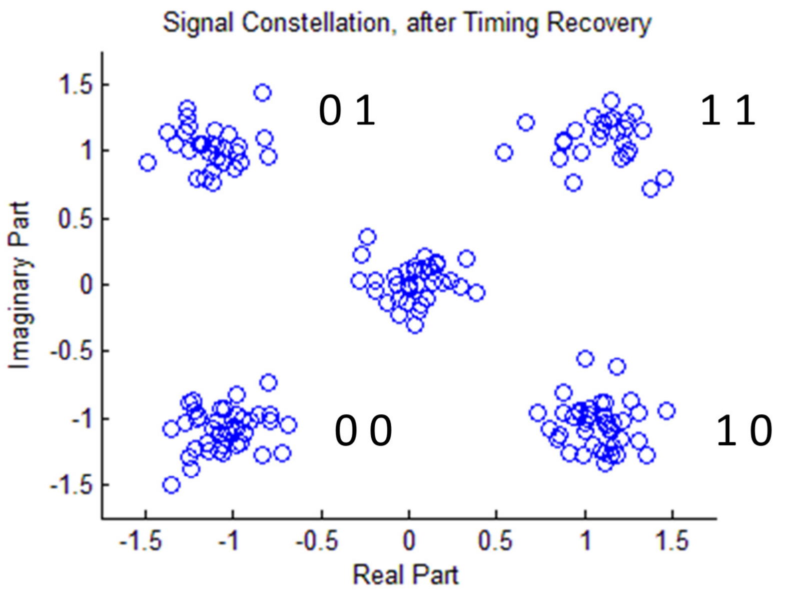

Figure 16. Mapping of QPSK symbols to bits from signal constellation of the symbols.

3 Methodology for Creating a Wireless Transceiver Core in MATLAB HDL Coder

I'll fill this part in

3.1 Algorithm Implementation Example – Costas Loop Core

This section will focus on **how** to script the MATLAB function for usage with the HDL Coder tool. The Costas Loop, used in the receiver for carrier recovery, will be used for this demonstration, since it is a reasonably complex algorithm which touches upon some key development features of the tool.

Since the function is being used for code generation, a `%#codegen` header must be placed at the start of the function. The function declaration with the inputs/outputs of the function (corresponding to the input/output signals to the core) follows the header.

```
%#codegen
function z = qpsk_rx_foc(y, mu, finish_rx)
```

The function describes the operations in each clock cycle, so it is worth mentioning that the function processes data on a sample-by-sample basis. The analog-to-digital converter (ADC) samples the received analog waveform to digitalize the signal, and then passes these samples on into the receiver core. The input `y` to the Costas Loop represents a sample of the received waveform. As for the other inputs/outputs, the gain coefficient `mu` is a parameter set in the test bench, and `finish_rx` is the reset signal for the core.

Next, there are specific protocols used to declare variables, such that they are appropriately mapped into memory by HDL Coder.

To store a variable in **RAM**, the variable must be declared persistent. This is done by

1. Definition of a variable as persistent, for example:

   ```
   persistent phi
   ```

   lets the tool to map the variable to RAM rather than registers, allowing the value of the variable to persist across multiple calls to the core.
2. The persistent variable(s) must be initialized to a custom value using an `isempty` statement, for example:

   ```
   if isempty(phi)
       phi = 0;
   end
   ```

   Without specific assignment, the persistent variable is initialized to an empty matrix. It is recommended that all persistent variables be initialized with an `isempty` statement.

To store a variable in **ROM** (as one would do for a look-up table, LUT), specific steps must be taken. These steps will be illustrated through the Costas Loop's storage of trigonometric functions, sin and cos.

1. Write a function to generate and store the values of the LUT. Typically, the function similar to "make_trig_lut" is written for this purpose. Some contents of this function are displayed below:

   ```
   function make_trig_lut

   % Generate LUT values
   ii = (0:(2^12-1))/2^12;
   c = cos(2*pi*ii);
   s = sin(2*pi*ii);

   % Create cosine LUT
   fid = fopen('COS.m','w+');
   fprintf(fid,'function y = COS\n');
   fprintf(fid,'%%#codegen\n');
   fprintf(fid,'y = [\n');
   fprintf(fid,'%14.12f\n',c);
   fprintf(fid,'];\n');
   fclose(fid);
   ```

   The script first generates the numeric values (cos and sin) to be stored as LUTs, then creates one function to store each LUT, "COS" and "SIN". A glimpse at the first few lines of the generated "COS.m" file shows the numerical implementation of the discrete COS function:

   ```
   function y = COS
   %#codegen
   y = [
   1.000000000000
   0.999998823452
   0.999995293810
   0.999989411082
   ```

A call to this function (`make_trig_lut;`) in the test bench initializes the LUT. LUTs are often used when a set of values are too complex to be generated on the fly (such as sine and cosine functions, training sequences, or digital filter coefficients).

2. After initializing the LUTs in the test bench, define the LUT in the function. Set the variable to contain the LUT equal to the generated functions containing the values. The variable containing the LUT should not be declared persistent. This definition should be performed as shown:

```
%ROMs *not* declared persistent
lSin = SIN;
lCos = COS;
```

Variables not intended for RAM or ROM should be declared without the "persistent" definition, and will be mapped into registers.

After variable declaration, the reset signal `finish_rx` for the phase offset `phi` is addressed:

```
if finish_rx == 1
    phi = 0;
end
```

`finish_rx` is 0 while a packet is being processed and 1 otherwise.

The next step is to focus on algorithm implementation. For the Costas Loop, first the local voltage-controlled oscillator (VCO) signal must be created. The Costas Loop operates through feedback, where the oscillation frequency of the VCO is determined by the control signal: the phase offset, phi. The oscillator is numerically generated as a series of complex numbers, whose real and imaginary parts are determined by the phi index of the cosine and sine LUTs (phi serves as the read index of the LUT).

```
% Create the VCO signal
if phi < 0
    phi = phi + 1;
end
if phi >= 1
    phi = 0;
end
phi12 = round(phi*2^12)+1;
if phi12 >= 2^12
    phi12 = 0;
end
f_i = lCos(phi12+1);
f_q = lSin(phi12+1);
f = complex(f_i,f_q);
```

Then, filtered input sample is mixed (multiplied) with the current value of the local oscillator, to demodulate the OTA signal from the carrier frequency to baseband. Because the frequency of the local oscillator has been matched exactly to the received carrier frequency (with feedback from the phase offset carefully tuning the oscillator frequency), frequency offsets at baseband are removed.

```
% Mix received sample, y, with local oscillator, f.
z = y*f;
z_i = real(z);
z_q = imag(z);
```

The baseband sample is separated into its real and imaginary parts, and fed as inputs to the phase detector. A voltage proportional to the phase difference between the real and imaginary parts is generated, and used to update the phase offset.

```
% Phase Detector: Form error term to drive VCO generation
tf = z_i*sign(z_q);
bf = z_q*sign(z_i);
e = tf-bf;

% Update phase offset
phi = phi + mu*e;
```

4 Overview of the Designs in the QPSK Tutorial Project

This section will give a high level overview of the project, to show users how to create complex cores. The test bench will be used to illustrate how signals flow through the cores, and the inputs/outputs of each core to will be discussed to provide an overall sense of how each core operates. In our projects, the transmitter and receiver cores share a test bench. The test bench begins by emulating the packet creation performed by the embedded processor in the FPGA.

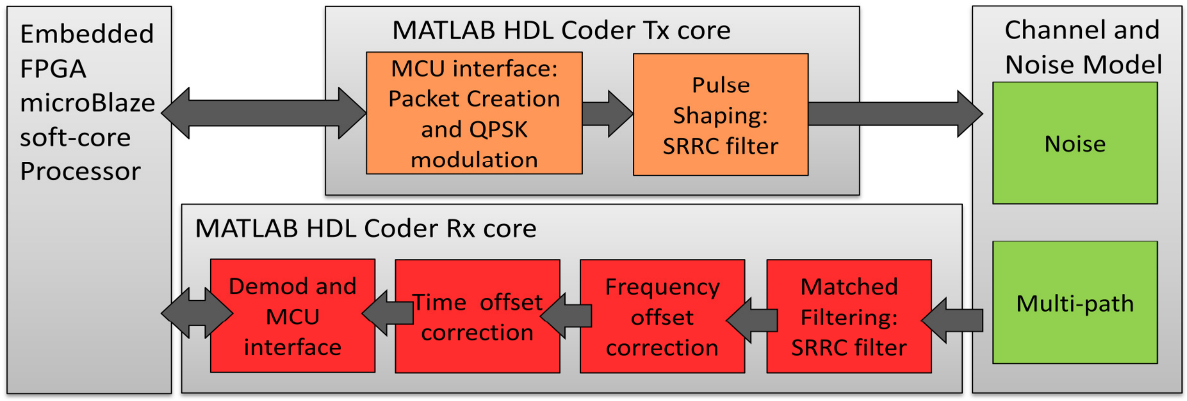
Embedded FPGA microBlaze soft-core Processor
MATLAB HDL Coder Tx core
MCU interface: Packet Creation and QPSK modulation
Pulse Shaping: SRRC filter
Channel and Noise Model
Noise
Multi-path
MATLAB HDL Coder Rx core
Demod and MCU interface
Time offset correction
Frequency offset correction
Matched Filtering: SRRC filter

Figure 17. High level overview of the transmit/receive cores' functionality, and integration the cores within an FPGA.

4.1 Transmitter Core

After packet formation, the transmitter core is called upon to transmit the data. This core is contained in the function, `qpsk_tx`:

```
function [x_out, re_byte_out, tx_done_out,d_b2s] = ...
    qpsk_tx(data_in, empty_in, clear_fifo_in, tx_en_in)
```

The `clear_fifo_in` is a control signal used to clear the internal first-in-first-out (fifo) buffer before accepting a new packet. Once the buffer is cleared, the `empty_in` signal allows the core to wait a couple clock cycles before loading the buffer. The packet data is then serialized into bytes and loaded into the fifo through the `data_in` input. As soon as the processor has loaded all packet data into the fifo, the `tx_en_in` enable signal goes high to start the transmission.

The core begins transmission by translating the bytes to symbols. It first demultiplexes the combined data sequence into its odd and even bits to form the in-phase $I(t)$ and out-of-phase or quadrature $Q(t)$

components of the signal, and adds a training sequence to the front of each. Then, $I(t)$ is mixed with a cosine and $Q(t)$ is mixed with a sine. The sinusoids are generated by a local oscillator at the carrier frequency f_c. At last, the two components are superimposed to form the resulting QPSK signal:

$$s_n(t) = I(t)\cos(2\pi f_c t + \phi_i) - Q(t)\sin(2\pi f_c\, t + \phi_i)$$

where ϕ_i is the phase offset at the transmitter (when imperfections in oscillator frequency f_c are viewed as a phase offset from the desired frequency).

The QPSK modulated signal is oversampled undergoes pulse shaping. Pulse shaping is performed with a square-root raised cosine (SRRC) filter, to band limit and eliminate intersymbol interference (ISI) from the transmission. The resulting signal `x_out` is output from the core to the antenna for transmission.

After every byte of packet data, the core sets the `re_byte_out` signal high to grab the next byte of data. Once all bytes in the packet have been handled, the core sets the done signal `tx_done_out` high.

4.2 Channel

A physical channel is simulated in the test bench by adding additive, white, Gaussian noise (AWGN) to the over-the-air (OTA) transmitted signal. In addition, the frequency offset applied by mixers in hardware is simulated in the test bench through the channel.

4.3 Receiver Core

The OTA signal is received and fed to the receiver core on a sample-by-sample basis as the input, `r_in`. The two other inputs, `muFOC` and `muTOC` are the gain coefficients for the adaptive elements (the carrier recovery loop and timing recovery loop, respectively) in the receiver core. The receiver core first performs matched filtering with the `qpsk_srrc` function onto the received OTA signal, to maximize the SNR ratio.

Then, a Costas Loop `qpsk_rx_foc` is applied to the signal for carrier recovery. Carrier recovery aims to tune the local oscillator's frequency to the received carrier frequency, so that demodulation of the OTA signal to baseband will allow accurate symbol decisions to be made.

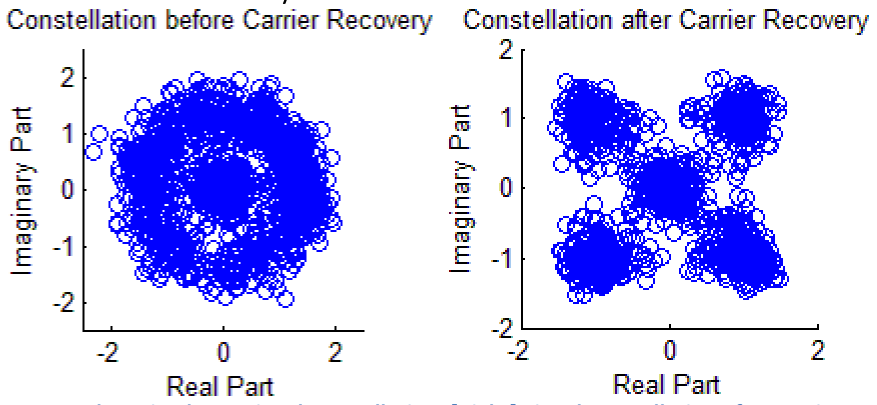

Figure 18. [Left] Unprocessed received OTA signal constellation. [Right] Signal constellation after carrier recovery. Before carrier recovery there is a rotation in the signal constellation, due the frequency offset in the signal after demodulation to baseband. With frequency offset correction, this rotation is removed from the signal constellation.

The receiver core then performs timing recovery on the received signal, to determine the optimal times for sampling. The received waveform is down sampled using the timing offset estimate, to provide an even cleaner signal constellation, which is ready for symbol decisions.

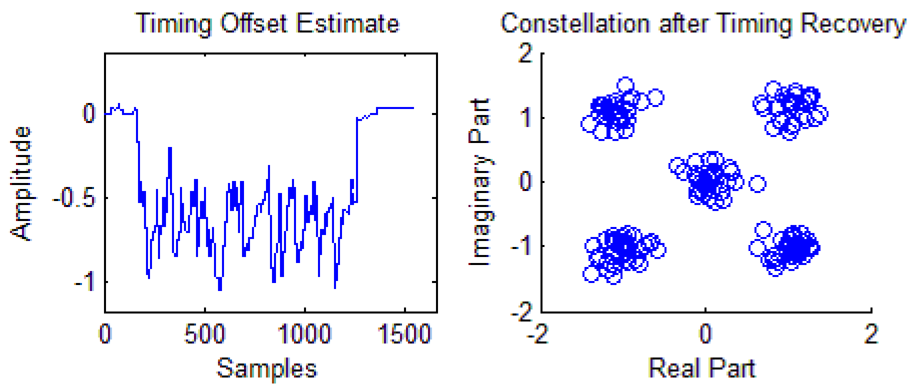

Figure 19. (Left) Timing offset estimate made by timing recovery loop in receiver core. (Right) Signal constellation after timing recovery; the timing offset estimate has allowed the received waveform to be down sampled at the optimal sampling times.

Cross-correlation between the training sequence and the processed signal is now performed to find the packet data within the received waveform. A glance at the magnitude of the cross-correlation function immediately reveals where the packet data lies within the received signal.

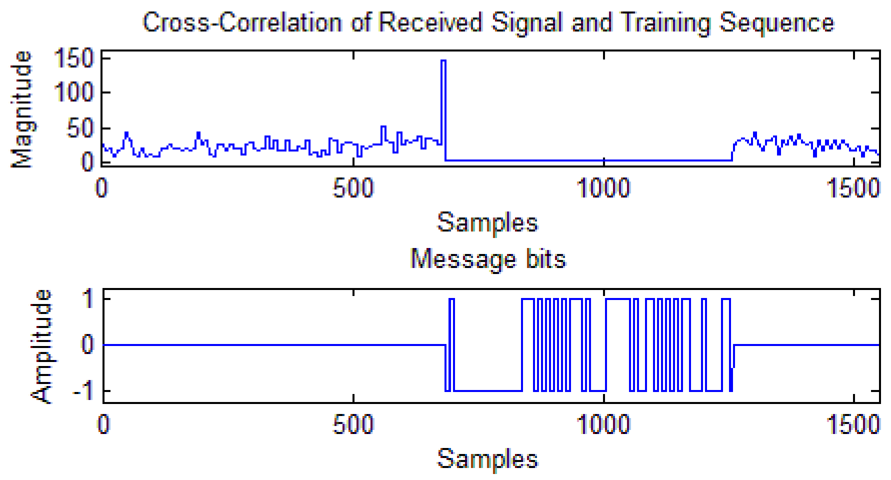

Figure20. Magnitude of Cross-Correlation between Received Signal and Training Sequence reveals quickly where the packet data lies within the received waveform.

Once the packet data is identified, the core sets the enable signal `en` high to indicate that a packet is being processed. Hard decisions are made, based on the symbol's place in the signal constellation, to convert the symbols to bits. The bits are translated into bytes and output from the core to the test bench (or embedded processor) as `byte_out`. The `finish_rx_out` signal serves as a reset for all the functions within the core.

5 Chilipepper – FMC Radio Board

We'll want to describe how to use the board and how it's used in the project. We'll save this for later.

6 References

Give citations for each algorithm