EE 358 Final Project

Austin Rothschild, Eugene Min, and Ishan Seendripu

Stanford University, Department of Electrical Engineering

March 2025

Abstract

In this report, we describe the operation of a custom SDR based wireless communication system and the associated techniques implemented in order to perform symbol, frame, and frequency synchronization. We discuss how each of these blocks work in the digital baseband of this communication system, and the functionality of the template implementations provided to us. Furthermore, we define a target Figure of Merit (FoM) dependent on digital hardware area, symbol rate, and BER performance. We explain our reasoning and design of various optimizations in order to enhance our FoM. The primary design change involves a custom implementation of a decision-directed carrier recovery PLL used to increase carrier frequency offset (CFO) compensation performance. We also discuss other optimization attempts that were made, and general debugging efforts in order to arrive at a functional design. Measured performance of the optimized design is highlighted in Section 3. Performance limitations are justified and other potential points of improvement are suggested.

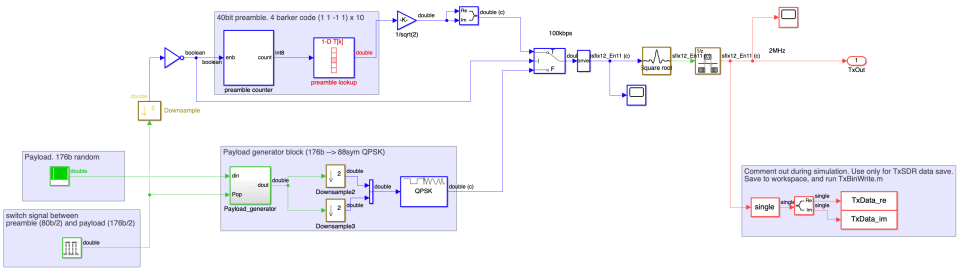
Contents

1 Overview

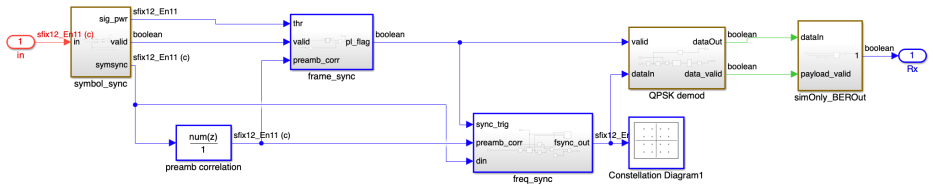
For the final project of EE 358, we implement and investigate optimizations for **symbol**, **frame**, and **frequency synchronization**. We first prototype our design in Simulink, and then demo the functionality in an over-the-air (OTA) setup running synthesized HDL implementations of the different synchronization methods. Finally, we explore design optimizations on the various synchronization blocks in order to minimize a specified design target Figure-of-Merit (FoM) which incorporates trade-offs among the symbol rate, HDL area, and BER.

1.1 System Architecture

In the default wireless link, we transmit packets at a configurable symbol rate and receive them with architecture shown below in Fig. 1. After some changes to the initial delay values in various blocks, this system works. However, it only works well for low carrier frequency offsets for reasons that will be explained.



(a)



(b)

Figure 1: Default TX (top) and RX (bot) Simulink blocks. Primary design changes are made to the RX synchronization blocks.

The preamble consists of a length 4 Barker code repeated 10 times in order for the receiver to perform frame synchronization. The autocorrelation of the received signal with the Barker code is subsequently used to calculate the frequency/phase offset of the packet. The data payload is QPSK modulated with 88 bits for each I and Q channels. The packet structure is shown in Fig. 2.

Since the preamble sequence is known, this is a data-aided approach. Data-aided approaches are popular in modern wireless systems since they are accurate and have good performance. The drawback is that the spectral efficiency reduces since resources are allocated to send non-information bearing training data. Thus, training sequences are ideally short so throughput can be maximized.

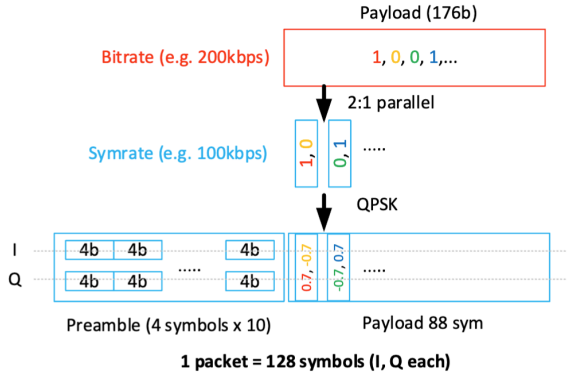


Figure 2: TX Packet Structure: 40 symbol preamble sequence followed by a 128 symbol payload.

The transmitter block sets the data and symbol rates to be used in the link (default is 100 ksym/s), and consists of a packet generator which is passed through a root-raised cosine filter for transmit pulse shaping. Pulse shaping with a roll-off factor β can be used to time-limit the TX signal and help minimize adjacent symbol/channel interference. The tradeoff is that there is additional spectral leakage, but this is not a problem for this system as we are not constrained by out-of-band emissions. Additionally, TX pulse shaping can reduce ISI due to frequency offsets that occur when sampling signals. The TX pulse shaping filter is an *interpolating filter* since it shapes the square baseband pulses into a smoother symbol. The pulse shaping filter of the TX block should be matched in the subsequent RX block in order to maximize received SNR and satisfy the Nyquist criterion for zero ISI [1].

1.2 Symbol Synchronization

The symbol synchronization block ensures that the receiver correctly detects the start and timing of each symbol received in a digital signal. It aligns sampling instants with the optimal downsampling point, which is ideally the center of the current symbol.

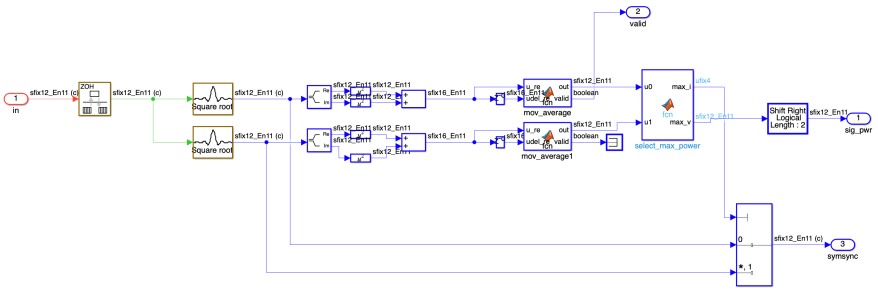


Figure 3: Symbol synchronization block implementing two delay lines at different decimation offsets. The received signal is passed through each path and the maximum correlation power path is selected as the optimal downsampling instant. The correlation power is also used as the frame detection threshold level.

The matched filter at the RX is a *downsampling filter* which is used to select which time offset to choose for symbol synchronization. The TX and RX pulse shaping filters should both be identical, and their product is thus a raised-cosine filter. The symbol synchronization blocks implement multiple delay-line paths with different decimation offsets (corresponding to symbol delays) and looks at the correlation power in each path with the received signal as follows:

$$\mathrm{E}_b \left[\left| \int y(t) \sum_n b^*[n] g^*_{TX}(t - nT - \tau) dt \right|^2 \right] \tag{1}$$

The path with the largest power is chosen, and that delay value is used for the fine-grained symbol synchronization offset and provides the optimal downsampling point. Knowing the optimal symbol offset to sample will lead to the lowest inter-symbol interference (ISI). This is equivalent to finding the sampling point on an eye-diagram where the eye opening is maximum and ISI from the neighbors is low.

The symbol synchronization block has non-trivial silicon overhead due to the need for numerous correlators and multiplier elements. Thus, the number of taps, delay lines, and FIR implementation should be chosen with care in order to minimize the FoM. More delay lines allow for a greater degree of freedom with respect to offsets to select from at the expense of FPGA area. For our implementation, we use a two delay line RX matched filter bank with each having 33 taps ($2 \cdot L + 1$ where $L = 16$ is the FIR filter span).

The characteristics of the pulse shaping filter also affect the performance of the symbol timing recovery. Having a larger excess bandwidth factor β provides more timing information and also reduce spectral overlap with neighboring symbols [2]. Besides pulse-shaping, equalization techniques can be used to mitigate or completely remove ISI, however that is not considered in this project.

1.3 Frame Synchronization

The default implementation of frame synchronization uses the preamble to determine where the payload data starts. This is implemented by correlating the signal coming out of the RX matched filter with a known preamble sequence. The known preamble sequence is used as taps of an FIR filter, scaled by the preamble length of 40 symbols. A popular code to use for this is known as the Barker code [3]. The frame sync block also takes in average power of the RX signal to act as a threshold level for the peak-detector within the frame sync block. The frame-sync block works by looking at the autocorrelation of the received signal with the known Barker code and observing the autocorrelation peak.

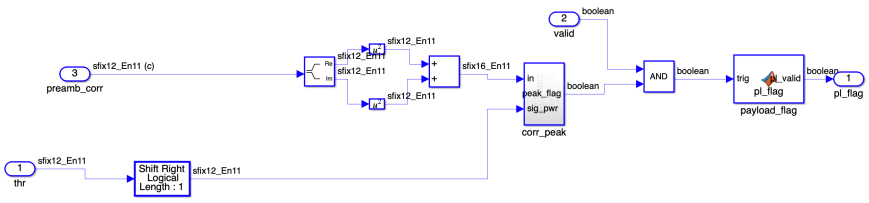
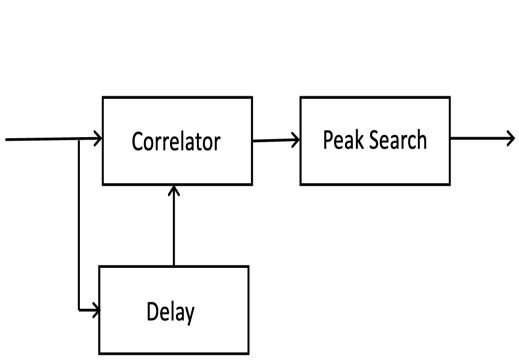
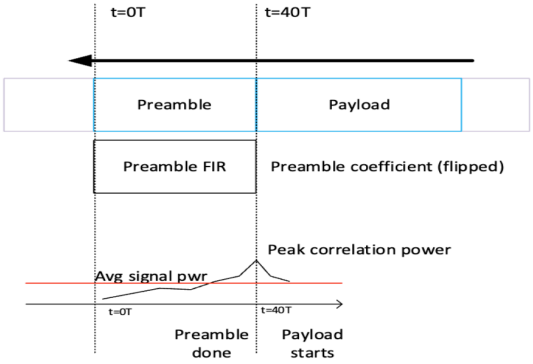


Figure 5: Frame synchronization subsystem. The received signal power is passed to the correlation peak detector along with the auto-correlated signal. The correlation peak detector finds peaks by comparing sampling offsets to each other and checking if a threshold has been exceeded. We describe this more in Section 2.3.



(a) Correlation of the received signal with the preamble Barker finds new received frames and indicates if a payload is valid.



(b) The peak autocorrelation of the received signal with the preamble is used to mark valid payload instances.

Figure 4: Frame Synchronization Overview

The unique structure of the Barker code makes it so that the autocorrelation is maximized when there is no offset, and low when there is offset. Thus, the peak should only occur at the right frame offset. A valid autocorrelation peak is used as the offset point to update the coarse carrier frequency offset, and indicate that a new payload is valid in the current received packet. Different lengths of Barker codes control the relative autocorrelation side-lobes with offset samples, as well as the non-offset autocorrelation peak.

A Barker code of length N is simply a sequence of $+1$ and -1 where the sidelobe autocorrelation does not exceed 1 in absolute value. The autocorrelation of the Barker sequence satisfies:

$$R(k) = \sum_{i=1}^{N-k} b_i \cdot b_{i+k} \leq 1, \quad \forall k \neq 0 \tag{2}$$

Longer sequences have stronger autocorrelation and lower sidelobe level ratios. There are 9 known Barker codes, and the commonly utilized ones are listed in Table. 1.

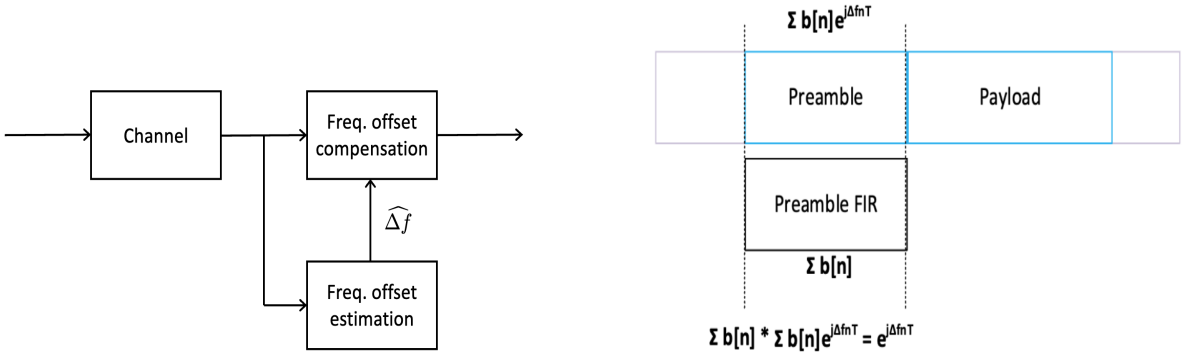
Length N	Barker Code Sequence
2	$+1, -1$
3	$+1, +1, -1$
4	$+1, +1, -1, +1$
5	$+1, +1, +1, -1, +1$
7	$+1, +1, +1, -1, -1, +1, -1$
11	$+1, +1, +1, -1, -1, -1, +1, -1, -1, +1, -1$

Table 1: Barker Code Sequences for Different Lengths.

In terms of design optimization for the frame sync block, the length of the Barker code and the subsequent FIR filter and peak detector play a role in the BER accuracy as well as design area. Additionally, since we elect to implement joint frame and frequency synchronization using the preamble sequence, the size of the Barker code is a tradeoff between the coarse frequency estimate desired for frequency synchronization and the autocorrelation magnitude/sidelobe ratio for frame synchronization. Often, a mixed approach is used.

1.4 Frequency Synchronization

To do carrier frequency synchronization, we use a data-aided approach to obtain a coarse CFO estimate. Since the receiver is using a known preamble sequence, it can directly estimate the carrier frequency offset (CFO) due TX/RX local-oscillator mismatch or Doppler shift by accumulating the phase rotations from the cross-correlation block used in frame-sync over time. Note that data-aided techniques somewhat depend on having a small CFO so that timing can be acquired first from an over-sampled and slowly rotating constellation which has not been compensated.



(a) (b)

Figure 6: Frequency synchronization overview of a data-aided approach which obtains a corase CFO update every new packet from the preamble autocorrelation.

The frame-sync block computes the square of the output to estimate the power. However, if the square is not taken, the cross-correlation output represents the average phase offset during the preamble sequence. If $b[n]$ is the known preamble Barker code sequence, and the received preamble sequence is

given as $r[n] = b[n]e^{j2\pi\Delta fnT}$, then their discrete autocorrelation is:

$$\begin{aligned} R[k] &= \sum_{n=1}^{N} b[n]r^*[n-k] \\ &= \sum_{n=1}^{N} b[n]b[n-k] \cdot e^{-j2\pi\Delta f(n-k)T} \\ &= e^{-j2\pi\Delta fkT} \sum_{n=1}^{N} b[n]b[n-k] \\ &= e^{-j2\pi\Delta fkT} R_b[k] \end{aligned} \tag{3}$$

At a lag of $k = 0$, $R[0] = N$ is just the length of the Barker sequence. At a non-zero lag k, the phase shift is **averaged**. Since $R_b[k]$ is known and real, the phase is completely determined by the phase shift of the preamble symbols. This average phase offset is recomputed for each new packet that is received, as the frame synchronizer block provides the timing trigger for sampling the average phase offset and is only sampled when the preamble is aligned. Furthermore, if we make the assumption that the offset is not large during the packet and remains relatively constant over the duration of the payload period, then this can be used at the frequency compensation de-rotator simply by taking the complex conjugate of Eqn. (3) and multiplying the payload by this symbol. Note that in order for this to work well, we should use sequences with a high degree of repetition. Thus, the Barker code length should either remain short, or there should be a dedicated synchronization sequence sent during the preamble that is separate from the frame synchronization sequence. For this design, we elect to use a 4-bit Barker sequence.

The current frequency offset compensation method applies a coarse estimate based on the present packet's preamble. However, the frequency offset may drift over the duration of the payload length and this leads to a slight rotation of the constellation which can lead to decision boundary crossings of the received signal and thus symbol errors. Empirically, we observed that the carrier frequency compensator drifts over the payload period, and is especially pronounced at the end of the packet. This severely limits the achievable BER and should be corrected to recover performance. We can mitigate this by applying a fine frequency correction which tracks the change over the whole packet. This is carried out by a decision directed feedback loop described in Section 2.1.

1.5 Default System

The above synchronization blocks were provided to us in a boiler plate fashion. The provided system had various issues with the delay and rate conversions that were dealt with initially. Specifically, the carrier frequency synchronization block did not have the proper delay set to account for the peak detector subsystem. Additionally, the QPSK demodulator was 1 sample too early on the payload valid, causing the BER to always be incorrect at the beginning of the frame. After resolving these issues, we could observe 0% BER in ideal channel conditions in Simulink. From here, we began investigating which blocks to improve in order to optimize our FoM. This is the subject of section 2.

2 Design Optimizations

Here we discuss the various design optimizations we made to enhance our FoM, as well as make the wireless link more robust.

2.1 Decision Directed Carrier Recovery

The decision directed carrier recovery constitutes the majority of our design changes for this project. Here, we describe the overview of the system, motivation for making this design change, implementation details, and validated performance improvement.

2.1.1 Overview

The primary design optimization we made centered around improving the frequency synchronization block described in Section 1.4. At the start of the carrier recovery sequence, we achieve full carrier synchronization by the preamble based compensation described previously. As the payload duration progresses, the residual CFO can cause symbol errors if the true CFO is not small relative to the payload duration. A way to mitigate this in data-aided designs is to apply decision-directed carrier recovery. In this method, the hard-decision outputs of a symbol decision device (QPSK demodulator) is fed to a comparison circuit which computes the phase error between the received signal and the decoded symbol. This phase error term can be used to discipline an oscillator as part of a PLL, or directly applied as a derotaton to the coarsely compensated payload. This has the advantage of being a symbol-by-symbol compensation, as opposed to packet-by-packet, so the resolution is much higher. The general block diagram of our approach is shown below in Fig. 10.

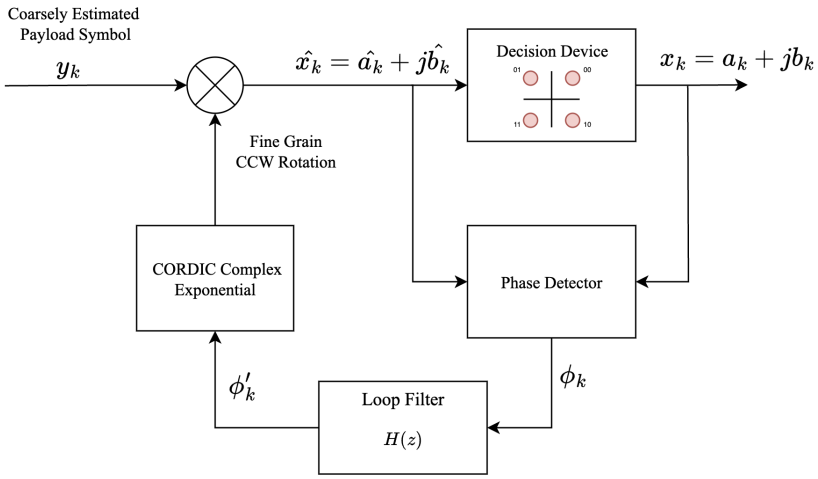


Figure 7: Decision directed carrier compensation block diagram. The coarsely estimated symbol occurring in the payload packet is y_k is multiplied counter-clockwise by a complex exponential generated by a decision directed phase error computation. The symbol decisions are used to track the phase drift over the course of the payload and update the de-rotation.

2.1.2 Motivation

The coarsely estimated payload symbol y_k has already been de-rotated by the coarse CFO estimate, which is the phase of the autocorrelation signal described in Section 1.4. The default compensation signal is shown below in Fig. 8. The system works by using the sinusoid of Eqn. (3) to act as a frequency compensator for the current packet. Thus, this method depends on the frequency offset being relatively constant over the duration of a packet. Observe that for a 50 Hz carrier frequency offset, the phase towards the end of the payload can be quite different from the present value, as indicated by the steep phase changes between some packets.

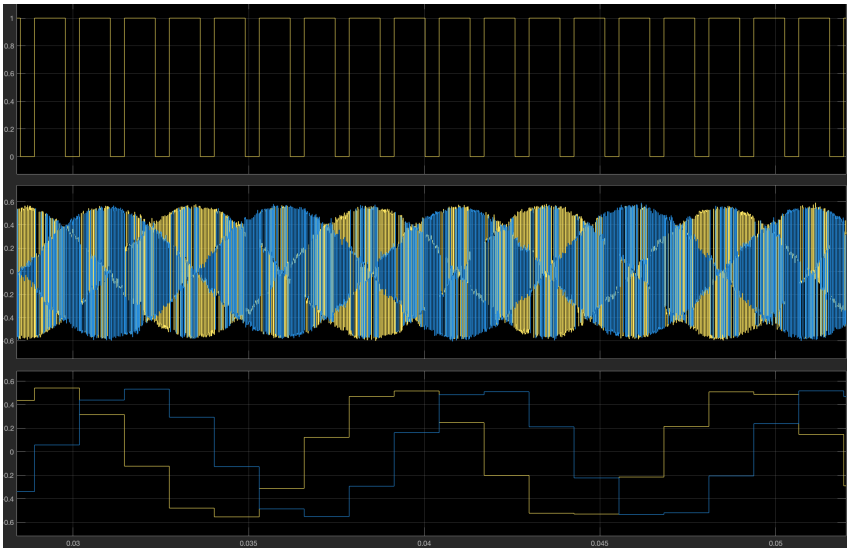


Figure 8: Top: sync_trig signal, indicating the start of a new valid payload. Middle: the RX output of the down-sampled RX match filter, spanning multiple packets. Bottom: the coarse CFO estimate for the RX signal data. Note that the CFO estimate is constant over each payload duration and updated once per packet. Simulated as CFO: 50 Hz.

For low CFO values, this system works fine. However, we know from previous labs and our own measurements that the CFO between the transmitter and receiver is actually much higher than 50 Hz, and thus the CFO drift causes a significant drift at the tail-end of each packet. This causes bit-errors to accumulate and lead to higher errors floors due to the end of the payload not being correctly demodulated. We validate this by showing what happens as the simulated CFO increases in Fig. 9.

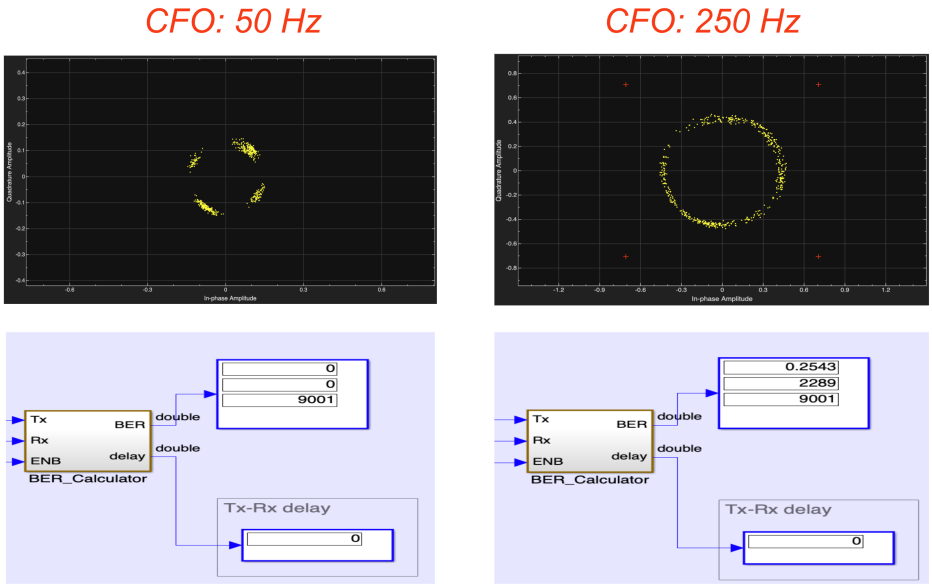


Figure 9: Effects of increasing CFO using only coarsely compensated frequency offset correction. High CFO leads the end of the payload to drift and cause incorrect demodulation decisions to be made, leading to higher BER.

In practice, the SDRs can have a CFO in the range of hundreds of hertz, so the drift towards the

middle/end of each payload from the initial compensation can be large. We observed this in our SDRs, and elected to focus on fixing this issue in order to get a good BER performance.

2.1.3 Phase Detector

The output of the QPSK decision device is a symbol in the complex plane represented by x_k. The actual received signal will be some signal in one of the QPSK quadrants $\hat{x}_k$ which has some phase offset ϕ_k as seen in Fig. 10.

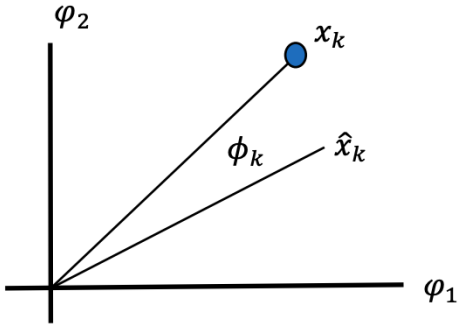


Figure 10: Decision Directed Phase Error [4].

The decision directed carrier recovery loop uses the phase difference ϕ_k to correct the residual offset. The decision device is often known as a slicer. If the SNR is high and phase offset is small, the slicer will make correct decisions and accurate phase estimates with high probability for symbols at the beginning of the packet. In low SNR conditions, incorrect symbol decisions can lead to error propagation which can cause the correction algorithm to diverge. The phase estimator extracts the error term by looking at the ratio of the input to the decision device $\hat{x_k} = \hat{a_k} + j\hat{b_k}$ relative to the output $x_k = a_k + jb_k$.

$$\begin{aligned} \frac{x_k}{\hat{x}_k} &= \frac{|x_k|}{|\hat{x}_k|} \cdot e^{\jmath\phi_k} \\ &= \frac{a_k + \jmath b_k}{\hat{a}_k + \jmath\hat{b}_k} \\ &= \frac{\left(a_k\hat{a}_k + b_k\hat{b}_k\right) + \jmath\left(\hat{a}_k b_k - a_k\hat{b}_k\right)}{\hat{a}_k^2 + \hat{b}_k^2} \end{aligned} \tag{4}$$

From this, we can get an expression for the phase error as:

$$\begin{aligned} \phi_k &= \arctan\frac{\hat{a}_k b_k - a_k\hat{b}_k}{a_k\hat{a}_k + b_k\hat{b}_k} \\ &\approx \arctan\frac{1}{\mathcal{E}_{\boldsymbol{x}}}\left(\hat{a}_k b_k - a_k\hat{b}_k\right) \\ &\approx \frac{1}{\mathcal{E}_{\boldsymbol{x}}}\left(\hat{a}_k b_k - a_k\hat{b}_k\right) \\ &\propto \left(\hat{a}_k b_k - a_k\hat{b}_k\right) \end{aligned} \tag{5}$$

where we have made use of the small angle approximation for arctan since the initial estimates will be coarsely compensated. This is referred to as a cross-product phase detector [5]. This method works well for high SNR scenarios [6], and is used after sending a known training pattern (in our case, the preamble). Alternatively, a LUT for the inverse tangent function can be used to get a more precise phase-error signal, but this is more expensive in hardware.

In our RX hardware model, we inserted a new sub-system into the QPSK demodulator block as shown in Fig. 11

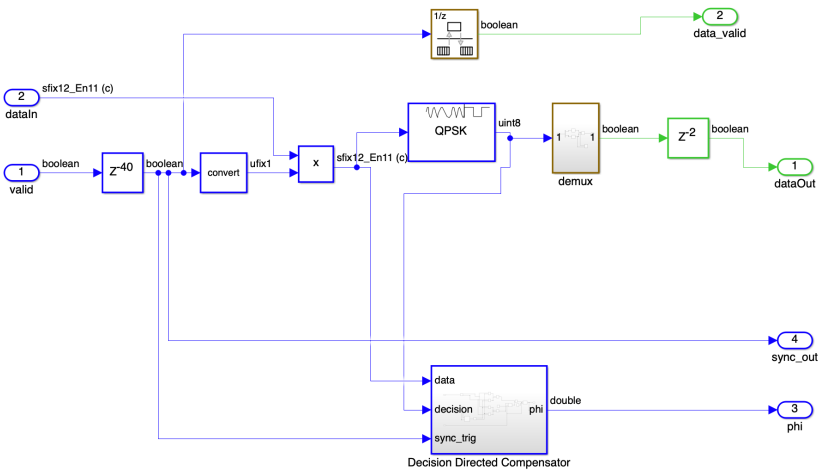


Figure 11: Decision directed subsystem in Simulink. The subsystem takes in the current symbol, the current symbol estimate, and the valid signal (sync_delay) which has been delayed by 40 to indicate the start of the payload. This block only operates on payload symbols.

Going inside the new subsystem, we see the implementation of a phase detector able to carry out the computation in Eqn. (5). This block takes as input the current payload symbol and extracts the real and imaginary components. The estimated payload at the output of the QPSK modulator is de-multiplexed in order to separate the real and imaginary bit streams. The QPSK system uses a Gray-mapped encoding (see Fig. 10 for the mapping). We then implement a complex 2D look-up table that maps the bit-stream to the corresponding complex valued output. This is shown over a single payload interval in Fig. 13.

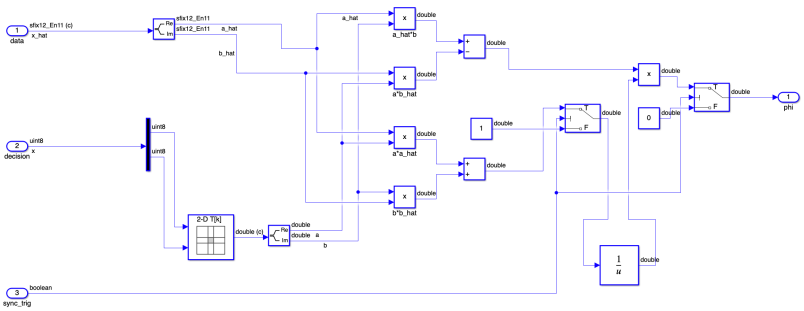


Figure 12: Implementation of the arctan phase detector, which computes Eq. (5).

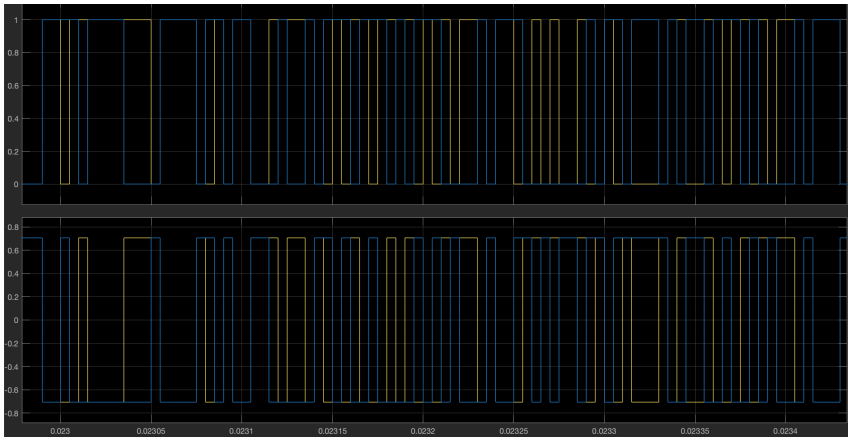


Figure 13: Output mapping of the estimated symbol complex LUT. Top: QPSK gray-mapped hard decisions. Bot: unit-energy complex representations of the symbols.

After the LUT mapping, the relevant signed product is computed between x and $\hat{x}$ using the real and imaginary components. The signal is then divided by the energy. Note that division is a very expensive computation in general. We initially were carrying out a full hardware division operation, but found we were close to over area and also failing setup/hold-time limits when attempting to generate the HDL. As such, we found an efficient method able to minimize both the area and latency significantly in [7]. Instead of division, we use an HDL optimized Newtwon-Raphson approximation of the reciprocal of the energy component, and then multiply this by the signed difference. This is a much more hardware-efficient approach, which allowed us to pass the HDL timing checks.

We see the output phase error signal over a few different packets shown below in Fig. 14. Note that the phase error is only computed during the payload interval. Outside of this, the input symbol stream has been zeroed out. To avoid division by zero errors, we use a switch to set this to 1 during these periods. Lastly, before outputting the phase error signal, we reset the error back to zero during preamble intervals and send this to the loop filter.

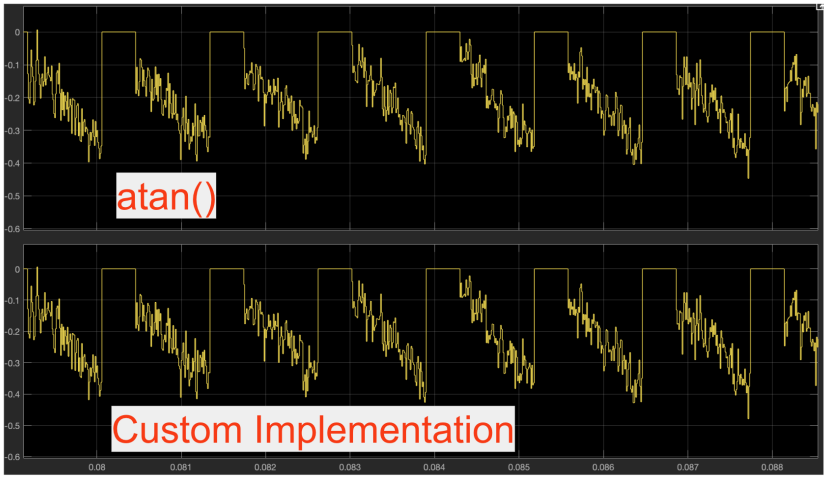


Figure 14: The unfiltered phase error signal (bottom) in comparison to an explicit atan() computation. We see a very close correspondence.

2.1.4 Feedback Loop

After computing the phase error for each valid payload symbol, we pass this to another custom Simulink block used to generate the carrier compensation signal. This is shown in Fig. 15. This block takes as input the true sync_trig signal, sync_delay (which is a 40 sample delayed version of sync_trig), the coarsely estimated preamble compensation sinusoid, and the phase error.

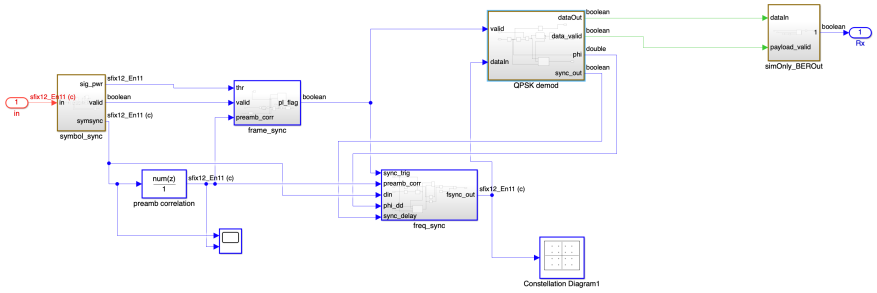


Figure 15: Simulink model updates showing the new signals and subsystems implemented in order to do decision directed carrier recovery.

The output is the signal used to apply the frequency compensation of the data signal. We see in Fig. 16 that we have still only have one derotation multiplier block to compensate the data signal. This is because we multiplex each new coarse CFO signal with the sinusoid generated from the phase error accumulation.

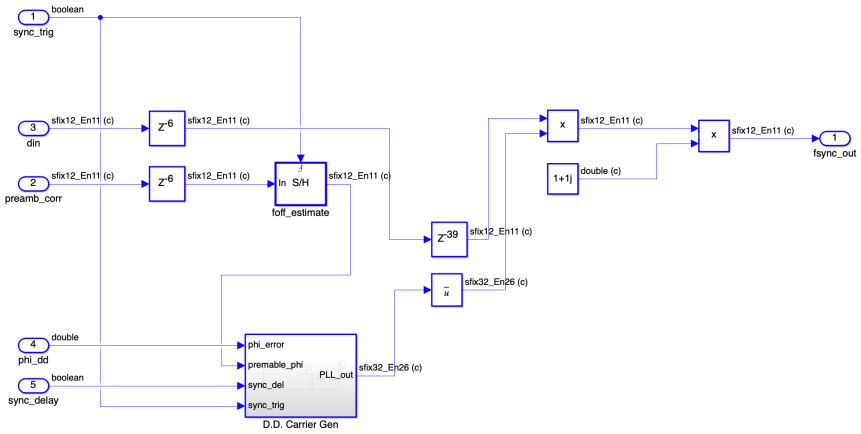


Figure 16

Going inside the D.D. carrier gen block as shown in Fig. 17, we see the loop filter, HDL LUT carrier generation block, and a switch to multiplex between the coarse CFO estimate and the decision directed sinusoid. This new sinusoid is generated from an HDL optimized complex LUT having 64 bits of precision. The block introduces a unit delay element in order to meet delay requirements for HDL synthesis, since this is a feedback loop and cannot apply these changes instantaneously.

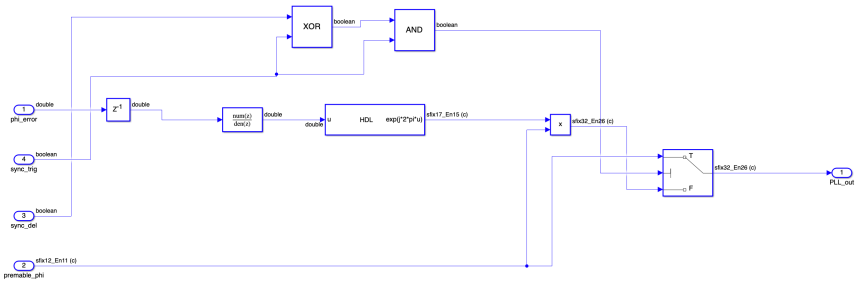


Figure 17: Loop filter frequency response.

For the loop filter, we implemented a simple first order low-pass infinite-impulse-response (IIR) filter. This was done since our phase error signal is rather noisy, and needs some smoothing applied to it before passing it to the sinusoidal LUT. An IIR was chosen since it is very low latency and does not cause large feedback delays. We initially implemented a moving average filter here, but the delay was prohibitory in HDL generation. The IIR filter is implemented with the following coefficients:

$$H(z) = \frac{-\frac{0.1}{2\pi}}{1 - 0.9z^{-1}} \tag{6}$$

where we have selected the numerator and pole location based on trial and error as a compromise between filter response and loop stability. The simulated frequency response is shown Fig. 18. The factor of 2π is used as a normalization of the error signal before passing it to the HDL optimized complex LUT used to generate the compensation sinusoid.

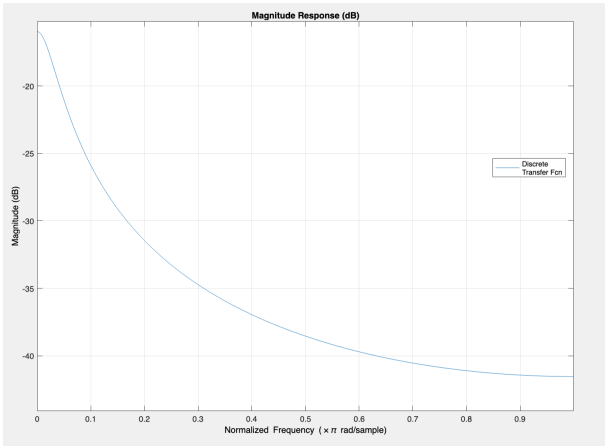


Figure 18

The relevant error signals and their conditioned responses are shown in Fig. 19. We wee the smooth response of the IIR filter output on each error signal. Note the polarity switch of the filtered signal when it is fed to the carrier generation signal. This is because we want the compensation signal to *subtract* from the current carrier offset drift, in order to provide compensation. Note that the smooth part resulting from the zeroed out phase signal during the preamble period of 40 symbols will not manifest in the final sinusoid, since a switch is used to multiplex between the coarse sinusoid and the decision directed sinusoid depending on whether a new frame has been obtained, or the signal is in the payload sequence.

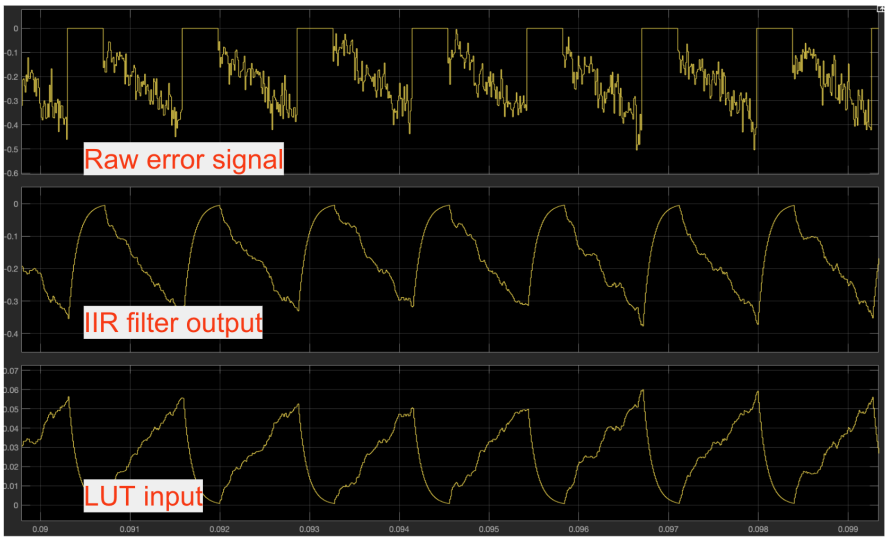


Figure 19: Phase error signals. Top: raw error from phase detector. Mid: IIR filter output. Bot: negative version of the IIR output.

The new compensation sinusoid and the relevant timing signals is shown in Fig. 20. The two trigger signals sync_trig and sync_delay are used in order to generate a pulse train to control the switch, which selects between a open loop one-shot phase update from the corase CFO, and the more finely tracked closed loop decision directed system. The switch control signal is generated as:

$$(\text{sync\_trig} \oplus \text{sync\_delay}) \wedge \text{sync\_trig} \tag{7}$$

The use of the switch ensures we capture each new CFO phase update generated from new packets, and also accumulates the CFO drift over the middle/end of the payload symbols. This interpolating effect can be observed from the bottom IQ signal in Fig. 20.

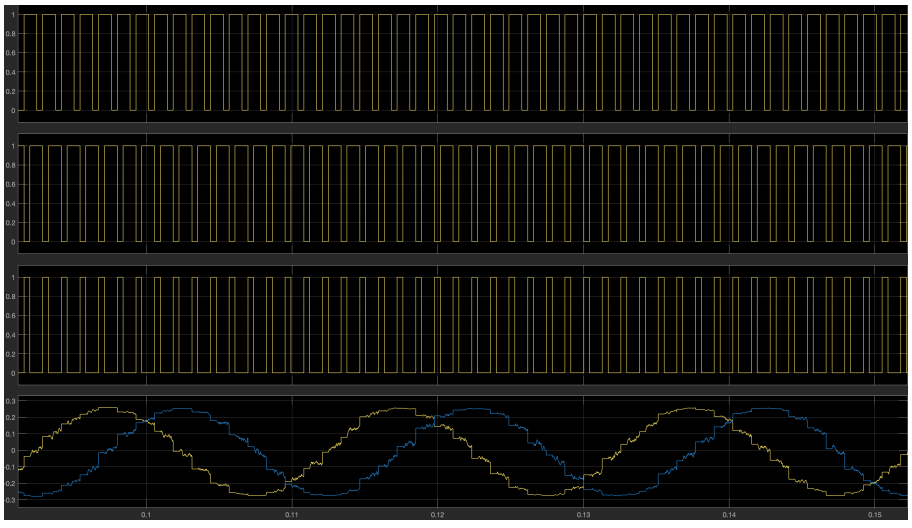


Figure 20: In order from the top: sync_trig, sync_delay, $(\text{sync\_trig} \oplus \text{sync\_delay}) \wedge \text{sync\_trig}$, compensator signal (PLL_out).

2.1.5 Simulated Measured Improvement

In Figure 21 we can see that we have essentially interpolated the phase offset between frames. However, we gain better results because we do not just do a linear interpolation, and instead we use the decisions to direct our interpolation. Even with steep phase jumps seen in a larger CFO, we are able to produce an interpolated phase correction. This saves BER performance, especially towards the middle/end of packets.

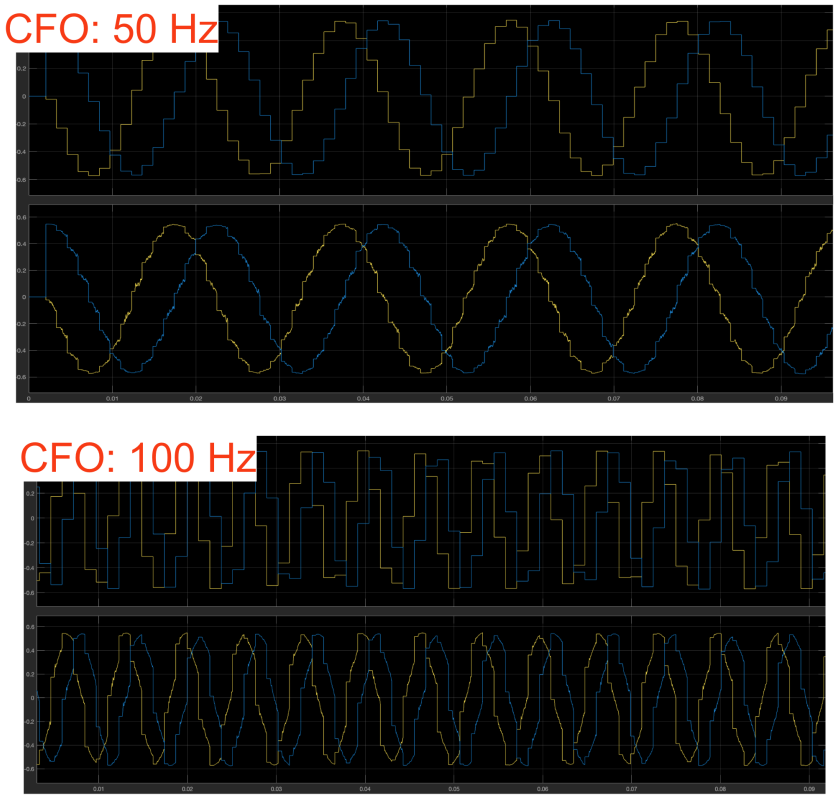


Figure 21: Output waveform of the phase compensation where the top shows the default system waveform and the bottom shows the decision directed system waveform under 50 Hz and 100 Hz CFO.

Figure 22 demonstrates from Simulink simulation of how our decision directed loop can greatly improve our constellation, resulting in a large reduction in BER. Under 100 Hz CFO, we were able to reduce our BER from 1.566% to 0%. Under 250 Hz CFO, we were able to reduce our BER from 25.43% to .88%. Thus, our custom implementation provides the improvement that that we desired, so we translated this to use in the final RTL.

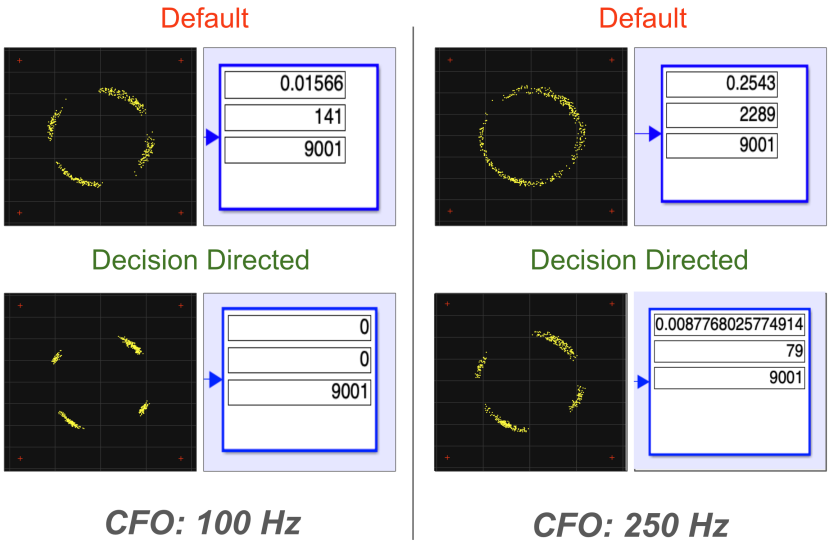


Figure 22: Comparison of simulated constellation and BER under the default and decision directed systems for 100 Hz CFO (left) and 250 Hz CFO (right).

2.2 Pulse Shaping Adjustments

One basic change we made to both our TX and RX blocks to make our system more robust was to maximize the raised cosine filter roll-off factor β [1]. Increasing the roll-off factor minimizes ISI ([8], [9]) due to faster time-domain roll-off. Any sampling offset error with a larger β will have lower ISI contribution - this can be observed at the zero crossings in Fig. 23 below. Note that this comes at the expense of frequency domain bandwidth, however, this system is not constrained by out-of-band or adjacent emissions like a commercial wireless system would be by FCC limits.

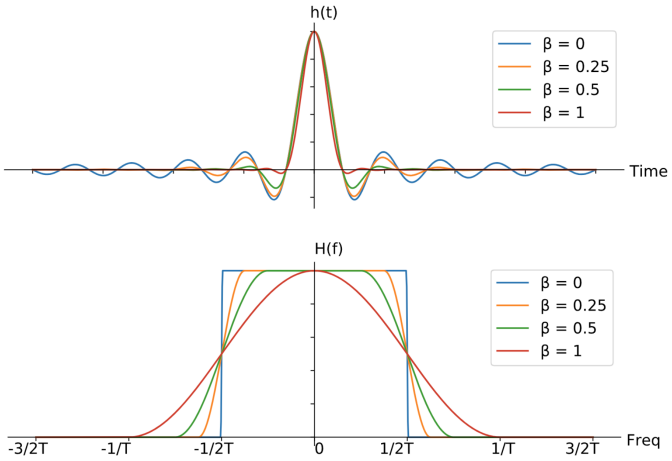


Figure 23: Root-raised cosine filter response as a function of the roll-off factor β. Top: time domain. Bottom: frequency domain.

In Fig. 24, we see that using a β has prohibitive ISI and symbol errors, while using a large β does not.

[1] The TX and RX filters are *square-root* raised cosine filters, and their frequency domain product is the *rasied* cosine filter.

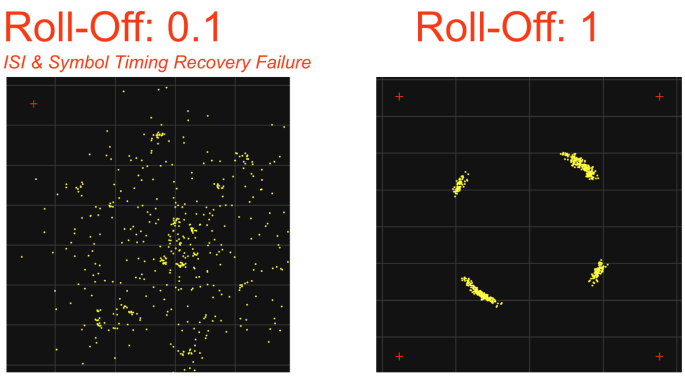
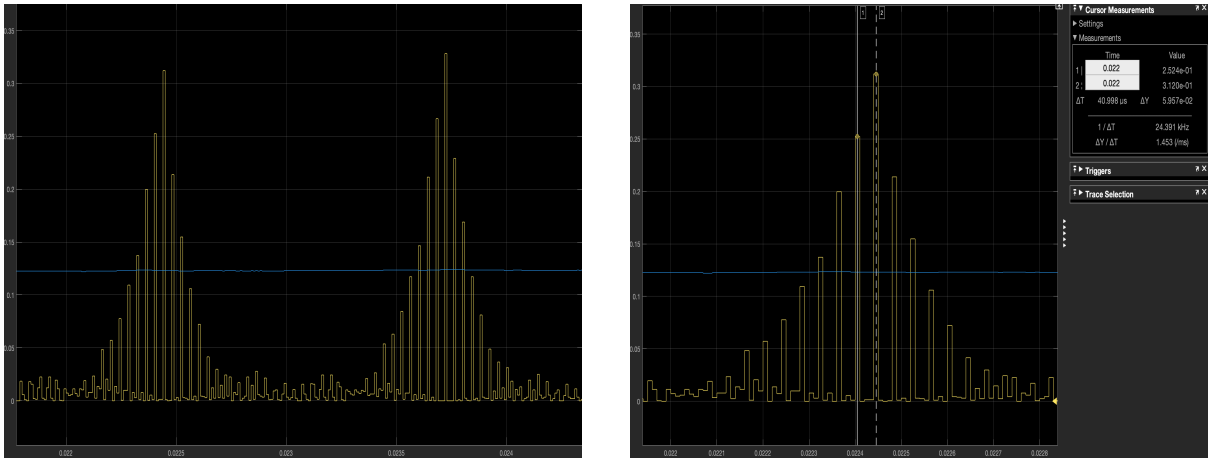


Figure 24: Simulated IQ constellation for $\beta = 0.1$ and $\beta = 1$ over the Simulink channel model. Without sufficient roll-off, symbol timing recovery has trouble synchronizing, and the resulting ISI is much higher due to not having a sharp enough time roll-off in between successive symbols.

2.3 Peak Detector Simplification

The default peak detector within the frame synchronization block has a redundant amount of checks that provide no useful information for peak detection. This is due to the structure of the Barker code used, as autocorrelation power will ideally be very low at points where the true preamble is not aligned to the right offset. At offsets of the code length, we see a peak. This is observed in Fig. 25. It is observed that these offsets only occur at $4/T_s$, therefore, it does not make sense to check at the offset points in between the peaks as the default peak detector circuit does in Fig. 26a.

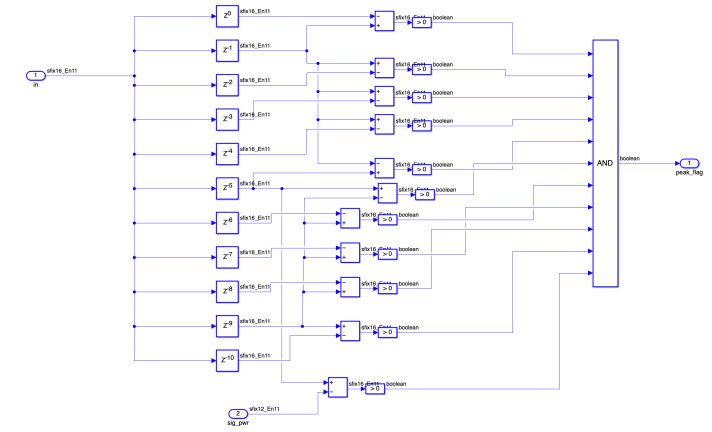


(a) Preamble autocorrelation over two packets.

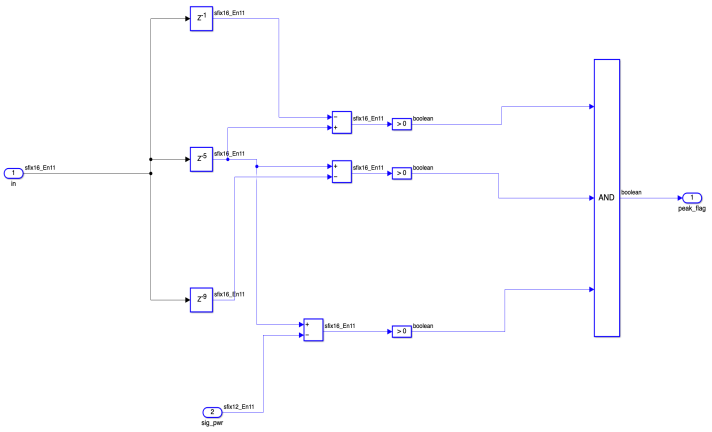
(b) Preamble autocorrelation of a single packet

Figure 25: Simulink scope of the autocorrelation power a packet (yellow) and the detection threshold (blue) passed into the the peak detector block. Peaks only occur at around 40 uS offets, which is $4/T_s$ where 4 is the length of the Barker code and Ts is the symbol period.

Given this behavior, we can simplify the logic of the peak detector circuit to that shown in Fig. 26b. We simplify the size of the and gate and also save on the intermediate comparison points. This logic only compares the center peak against the left and right peaks, along with the threshold level. If all conditions are met, a valid frame has been found and the synchronization offset is obtained.



(a)



(b)

Figure 26: Default peak detector block (top) and simplified block (bottom). The bottom block is a sufficient statistic for doing peak detection, since only offsets at the length of the Barker code need to be compared against each other and the threshold. In this case, only delays of 4 symbol periods from the center need to be compared (delay of 1 and delay 9).

2.4 Wordlength Optimization

To minimize FPGA area, we use fixed-point arithmetic instead of floating-point representation. A key factor influencing FPGA resource utilization is wordlength, which affects the number of logic gates, adders, and multipliers required. Optimizing wordlength involves balancing precision and hardware efficiency to reduce resource usage while maintaining acceptable bit error rate (BER) performance.

The optimization process begins by identifying which signals require higher precision and which can tolerate reduced wordlength without significantly impacting BER. For signals that can be truncated, we iteratively reduce their wordlength one bit at a time, monitoring BER performance after each adjustment. The process continues until a reduction causes an increase in BER, determining the minimum wordlength needed for that signal.

By applying this method across different system blocks, we identified symbol synchronization, frame synchronization, and frequency synchronization as the key areas where wordlength optimization can significantly reduce FPGA area. Below we summarize some adjustments.

- **Symbol Synchronization**: The output of the sym sync block can be reduced to **11-bit wordlength**.
- **Frame Synchronization**: In the *peak correlator*,
  - The **input** can be reduced to **8-bit wordlength** with **6-bit fraction length**.
  - The **threshold comparator adder output** can be optimized to **5-bit wordlength** with **6-bit fraction length**.
- **Frequency Synchronization**:
  - The **first multiplication block** input can be reduced to **11-bit wordlength** with **9-bit fraction length**.
  - The **second multiplication block** input can be reduced to **7-bit wordlength** with **8-bit fraction length**.

By systematically truncating precision where tolerable, we effectively reduce the total FPGA area, optimizing resource utilization without compromising system performance. We were able to bring our synthesized net FPGA area down from around 38,300 to a final area of 36,852.

3 Measured Performance

We will now discuss our measured results and subsequent FoM, IQ Constellation, and achieved BER performance.

3.1 Final FoM

Our figure of merit is defined as:

$$FoM = \frac{1}{A} \cdot f_{sym} \cdot \log_{10}(BER) \tag{8}$$

where A is the total percentage of the FPGA area used (omitting the transmitter block), f_{sym} is the highest symbol rate at the input of the RX matched filter in kHz [2], and BER is the bit error rate. We would like to make our FOM as negative as possible as a result of the BER's logarithmic dependence. Note that in our final design, we were able to successfully double the symbol rate in order to double our FoM.

We measured the performance of the baseline system given to us and also the performance of our final implementation. The results are shown in Table 2. Without our design optimizations, we found the system was non-functional and unable to perform reliable synchronization in an over-the-air setup, leading to a BER of 0.5 [3]. With our custom implementation of decision directed carrier recovery compensation, we are able to achieve a BER of 0.0057 using a transmit power amplifier gain of 40 dB, a receiver LNA gain of 40 dB, over a spacing of 1 meter. We observed lower BER performances for increased TX power due to enhanced SNR, as expected.

[2] f_{sym} here is given as the highest sample rate at the RX matched filter input. This is decimated by a factor of 2 to the base symbol rate.

[3] We attribute this to higher than expected carrier offset between the TX and RX SDR units.

Design Variant	Total Area	FPGA Area	Area Occupied %	f_{sym} [kHz]	Measured BER	FoM
Baseline	33720	9451	23.866	200	0.5	-252.265
Decision Directed, Increased Rate	36852	12583	31.775	400	0.005711	-2824.997

Table 2: Final FoM.

3.2 IQ Constellation

The measured constellation shown in Figure 27 looks to be nicely clumped into a QPSK shape. However, there is still an observed smearing of the constellation points due to Gaussian noise and phase noise. Additionally, there is a constant phase offset in our constellation, which can be attributed to residual CFO.

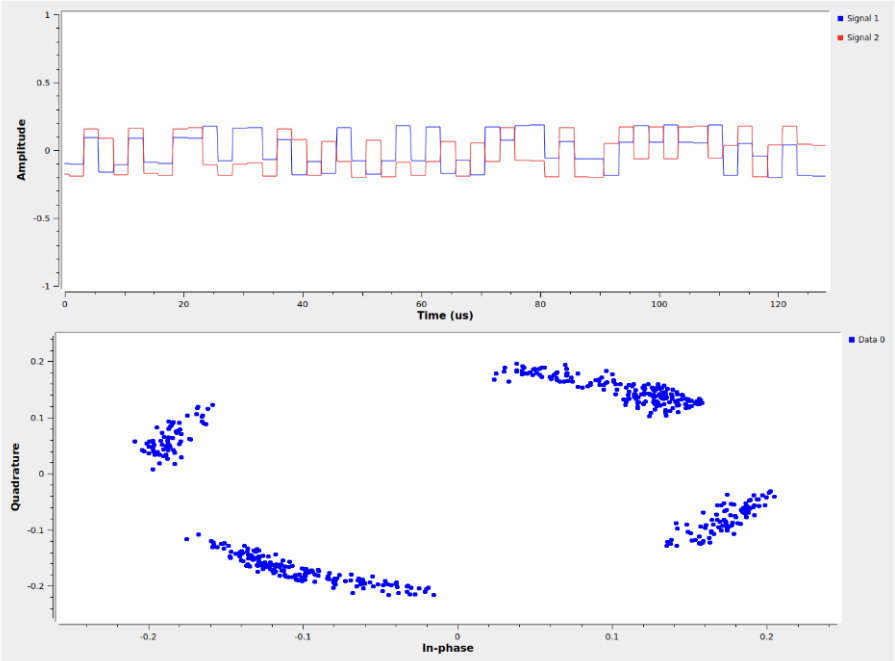
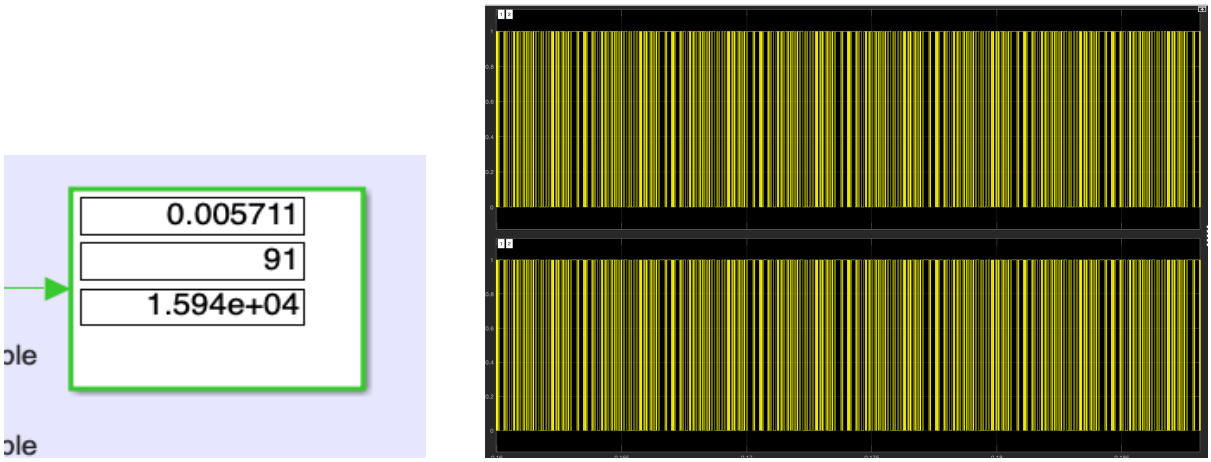


Figure 27: The IQ Constellation of our Final Design with Decision Directed Carrier Recovery at base symbol rate 200 ksym/s.

3.3 BER Performance

Here we show the final BER performance of our custom system at the output of the QPSK demodulator block in Fig. 28. Additionally, we display the bit output of our SDR compared to the actual payload bits sent and one can see that the bits match up well.



(a) (b)

Figure 28: Measured BER over $1 \cdot 10^3$ symbols (left) and comparison of true payload to demodulated payload (right).

3.4 Other Attempts

Here we describe some other design optimizations we had attempted but ultimately did not incorporate into our final design due to either no noticeable performance increases or insufficient time to finalize such changes.

3.4.1 Signed Preamble Correlation

An idea we had to minimize the synthesized FPGA area involves replacing the preamble correlator with a signed correlator. A signed correlator is a correlator that is designed to perform correlation without needing hardware multipliers, but instead taking either addition or subtractions based on the *sign* of the preamble symbol [10]. Since the Barker code sequence is just $+1$ or -1, a tapped delay line with the appropriate addition/subtraction at a given offset can implement the preamble correlation. This approach significantly reduces hardware area, since no multiplier blocks are used. The first step our signed correlator does is determine the sign of the input signal. We then put it into 39 delay blocks, which matches our Barker code having length 40. Each copy of the signal as well as the signal itself go into a LUT, and all the LUTs' outputs are put into an adder, as seen in the block diagram displayed in Figure 8. However, after testing this in our Simulink model, we realized that the signed correlator does not provide phase information which we are using for our coarse frequency offset correction at the start of each new packet. Hence, to use the signed correlator, we would need to break the preamble into a frame sync block (using a Barker code) and frequency sync block (using a repeated sequence). We did not have time to prove this out further, but it remains a low-hanging fruit to significantly save on FPGA area.

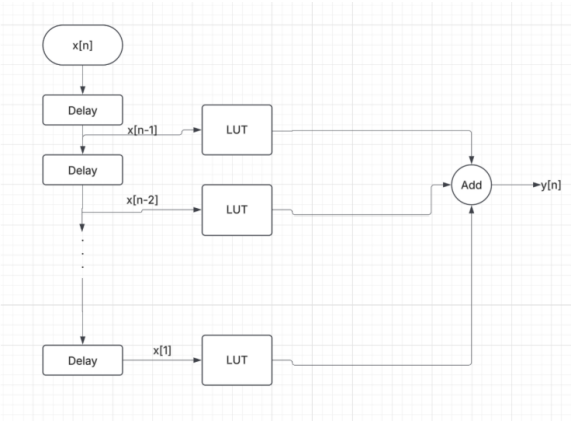
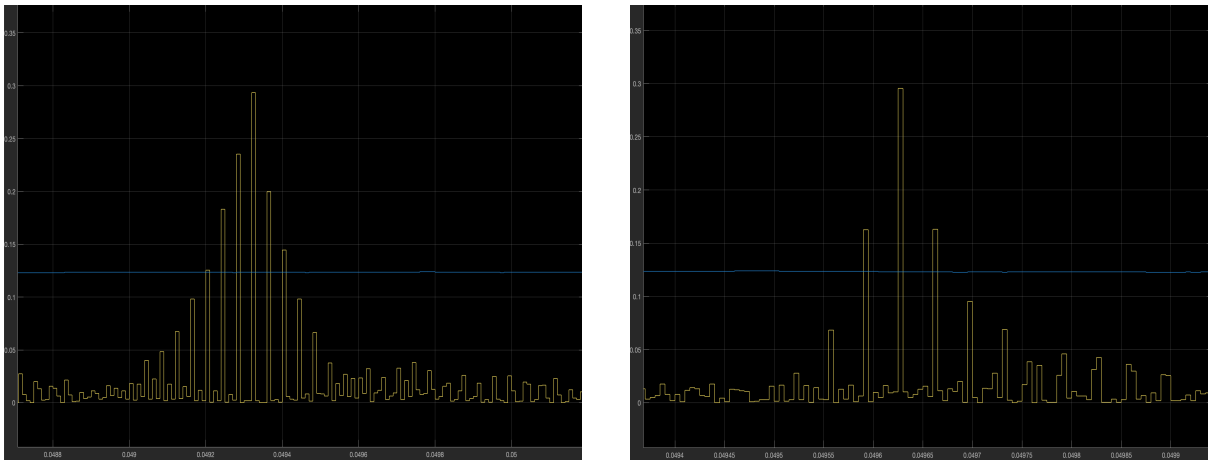


Figure 29: Signed correlator block diagram.

3.4.2 Barker Code Adjustments

Changing the Barker code length changes the absolute peak of the discrete autocorrelation with the preamble correlator. A longer Barker code reduces the cross-correlation side lobes. Both of these factors make frame detection more robust in lower SNR. We experimented both in Simulink and over-the-air testing using length 4 and 7 Barker codes (See Table. 1), and did not notice any significant improvement. However, we did observe that using length 7 Barker codes change the preamble based coarse CFO estimate. This is due to the fact that the repetition training sequence used for frequency compensation is longer, however, for carrier frequency compensation we want to use short and highly repetitive structures. As such, we ended up staying with a length 4 Barker code since we rely on the repetitive pattern of our preamble, and the observed SNR was not low enough to necessitate moving to a code with a stronger autocorrelation peak.



(a) (b)

Figure 30: Autocorrelation power for a length 4 Barker code (left) and a length 7 code (right), both normalized to the preamble symbol length. As expected, the side-lobes of the longer code are lower, and spread further away from the main lobe. This property makes detection easier in noisy environments and allows lower threshold tuning.

3.5 Design Limitations

Although our design provides good performance under the current setup, we have identified possible ways to improve our performance.

One of the major implementation challenges we encountered was the complexity of incorporating feedback. The feedback path requires minimal delay, and implementing a PLL introduces significant latency constraints. Additionally, the loop filter design can be very sensitive and dictates the overall performance of the system. We learned too late from [11] that we can avoid the inherent limitations imposed by decision directed feedback by utilizing a feedforward carrier recovery system where we simply delay the main data line and apply the decision phase error from the current symbol to itself. The main advantage of this approach is that no PLL is required and entirely avoids the complications introduced by the loop filter. Additionally, optimizing the PLL based approach could be done by enhancing our loop filter, which is currently a basic first order IIR filter. High performance systems use a critically damped, second order digital biquadratic filter. We tried this but did not have enough time to get it working well. This would lead to better phase accumulation over the payload interval, and faster settling times. Lastly, using a dedicated CORDIC NCO within the PLL could provide the most accurate frequency tracking performance, but this comes at the expense of area.

Additionally, we considered changing our preamble to optimize the length for a datarate increase, but we did not have sufficient time to fully study this and implement in our systems. We use many repeats of our Barker code, but we believe we can repeat it less and shorten our preamble significantly to decrease our overhead, which would increase our data throughput.

Moreover, we could have implemented automatic gain control (AGC) in our receiver, which automatically adjusts the gain of the received signal to maintain a consistent output level despite variations in the input signal strength. In a typical RX chain, signals from an antenna can vary widely in amplitude due to factors such as distance from the transmitter, interference, and multi-path fading. The AGC circuit detects the amplitude of the incoming signal and adjusts the gain of the amplifier accordingly to prevent distortion or saturation when the signal is too strong and to enhance weak signals for better demodulation. This ensures that the signal level fed to subsequent processing stages remains within the optimal dynamic range, improving the receiver's robustness and performance.

We also do not utilize equalization or coding techniques which would make our system more robust to ISI and bit errors. One main reason we did not pursue these options was due to our limited FPGA area and these techniques would have been prohibitively large.

4 Conclusions

Through this project, we were able to synthesize a lot of the learning we have had both in lecture in other labs throughout the course into a single system able to successfully transmit and receive data over the air. We found that after getting the initial system resolved of all the hidden bugs, that we wanted to focus on improving the phase compensation accuracy in order to lower the BER performance of the receiver, since this would give us the biggest FoM improvement. We identified the frequency synchronization block as being sub-optimal and BER limiting, and spent nearly all of our time researching and prototyping improvements to this block.

In the end, after many days and long hours of testing and iteration, we were able to refine our system into something that works well and satisfies all the complications involved in digital RTL synthesis on our SDR FPGAs. Overall, it was extremely satisfying to get everything working, and to have designed

something from scratch based on the theory we have learned in lectures and our own literature review. We now all possess a much deeper understanding of the entire baseband transmit and receive chain at a fundamental level, and have importantly seen the challenges involved in translating theory to functional hardware. Although it was difficult and time-consuming, the process was very instructive. Additionally, we are keenly aware of what blocks could be improved in future revisions.

References

[1] Wikipedia contributors, "Nyquist ISI Criterion — Wikipedia," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Nyquist-ISI-criterion

[2] S. Shahabuddin, "How excess bandwidth governs timing recovery in digital communication systems," 2024. [Online]. Available: https://wirelesspi.com/how-excess-bandwidth-governs-timing-recovery-in-digital-communication-systems/

[3] Wikipedia, "Barker codes," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Barker_code

[4] J. M. Cioffi, *Chapter 6: Fundamentals of Synchronization.* Academic Press, 2009. [Online]. Available: https://cioffi-group.stanford.edu/doc/book/chap6.pdf

[5] DSP Stack Exchange User, "High modulation index psk carrier recovery," 2024. [Online]. Available: https://dsp.stackexchange.com/questions/17297/high-modulation-index-psk-carrier-recovery

[6] J. G. Proakis, *Digital Communications*, 4th ed. McGraw-Hill, 2001.

[7] MathWorks, "Hdl reciprocal," 2024. [Online]. Available: https://www.mathworks.com/help/simulink/ref_extras/hdlreciprocal.html

[8] Wikipedia contributors, "Raised-cosine filter," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Raised-cosine-filter

[9] S. Shahabuddin, "Pulse shaping filter: Raised cosine and square root raised cosine filters," 2024. [Online]. Available: https://wirelesspi.com/pulse-shaping-filter/

[10] E. Rogozhnikov, "Performance comparison of fpga-based methods for preamble detection at interference," in *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, 2019, pp. 0152–0155.

[11] E. Ip and J. M. Kahn, "Feedforward carrier recovery for coherent optical communications," *J. Lightwave Technol.*, vol. 25, no. 9, pp. 2675–2692, Sep 2007. [Online]. Available: https://opg.optica.org/jlt/abstract.cfm?URI=jlt-25-9-2675