

A 60 GHz Development & Test Platform for Wireless Systems Research

Christopher Yarp

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-17

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-17.html>

May 1, 2018



Copyright © 2018, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

The author would like to thank Professor John Wawrzynek and Professor Robert Brodersen for their help and guidance throughout the project and the subsequent report. The author would also like to thank the students at the Berkeley Wireless Research Center (BWRC) who shared their perspectives and knowledge.

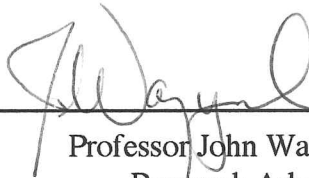
A 60 GHz Development & Test Platform for Wireless Systems Research
by Christopher William Yarp

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

Committee:



Professor John Wawrzynek
Research Advisor

May 10, 2017

(Date)



Professor Robert W. Brodersen
Second Reader

5/10/17

(Date)

A 60 GHz Development & Test Platform for Wireless Systems Research

Christopher William Yarp

5 May 2017

Abstract

Wireless networking has become one of the predominant last-hop networking technologies of choice for consumer and enterprise applications. While this popularity has led to increased investment in wireless research and wireless deployment, it has also led to increased contention for shared spectral resources as well as increasing expectations from end users. Increased use of millimeter wave (mmWave) spectrum for communications is one compelling technique to address both the increased demands of end users as well as congestion in lower frequency bands. However, mmWave provides its own set of benefits and challenges when constructing networks, leaving many open research questions. To evaluate these questions, a testing and development platform that is both flexible and capable of real time operation with acceptable data rates is required. The creation of such a development platform is the focus of this project. The resulting hardware platform is based around a Field Programmable Gate Array (FPGA) and an accompanying baseband design that resides completely on the FPGA. The baseband design exhibited good performance in simulated tests and performance evaluations with the entire platform demonstrating a reliable 60 GHz link in a series of physical tests.

Contents

1	Introduction	3
1.1	Goals & Scope	5
1.2	Baseband Specification	6
1.3	Tools	7
2	System Architecture	8
2.1	Hardware Platform	8
2.2	SoC Top Level Design	9
2.3	RF Frontend	12
2.3.1	Features	12
3	Tx Baseband Architecture	13
3.1	BPSK Modulator	13
3.2	Square Root Raised Cosine Transmit Filter	14
4	Rx Baseband Architecture	19
4.1	Square Root Raised Cosine Receive Filter	19
4.2	AGC	21
4.3	Timing Recovery	23
4.3.1	Timing Error Estimator	26
4.3.2	Loop Filter	30
4.3.3	Timing Correction Accumulator & Symbol Clock Generator	30
4.3.4	Fractional Delay Filter	31
4.4	Clock Crossing	32
4.5	Carrier Recovery	32
4.5.1	Phase Error Estimator	33
4.5.2	Loop Filter	35
4.5.3	Phase Accumulator	35
4.5.4	Phase Ambiguity	36
4.6	Golay Corrector & Preamble FSM	36
4.7	BPSK Demodulator & Data FSM	40
5	FPGA Optimizations and Implementation Results	41
5.1	Timing	41
5.2	Area	42

5.3	Synthesis and Implementation Settings	44
5.4	Floorplanning	44
5.5	CSR False Paths	45
5.6	Integrated Logic Analyzer	46
5.7	Resource Utilization	46
6	Testing & Evaluation	52
6.1	Testing & Evaluation During Development	52
6.2	Performance Characterization - AWGN Channel	53
6.3	Example Frame Reception with AWGN Channel	55
6.3.1	Without Carrier or Timing Frequency Offset	55
6.3.2	With Carrier and Timing Frequency Offset	58
6.4	Physical Testing	60
7	Extending the Baseband to Support Higher Order Modulation Schemes	61
8	Future Work	64
9	Conclusion	65
	Appendices	71
A	QAM Modulation	71
B	QAM Demodulation	74
C	Carrier Phase and Frequency Offset (CFO)	77

Chapter 1

Introduction

Wireless communication has become one of the most popular last-hop network technologies due in part to its ease of deployment, ability to provide reasonable quality connectivity in many scenarios, and near ubiquity in modern devices. While this popularity has spurred recent development of wireless technology, it has also resulted in increased contention for limited, shared, resources. At the same time contention for spectral resources has increased, the desires and expectations for future wireless systems continues to increase. Data rate, which has always been one of the primary focuses of improvement, is expected to increase further as requirements continue to grow [1]. Applications like 4K video and VR are also expected to increase bandwidth requirements significantly compared to today [2]. Limited resources are also being shared among an increasing number of end devices. In addition to the data rate expectations, there has been renewed interest in reducing latency and increasing the reliability of wireless communications. The confluence of all of these goals can be seen in the road maps for proposed wireless technologies such as 5G [3] [4].

There are many challenges presented by these combinations of goals. One of the core issues is that optimizing for two of the above goals tends to degrade the performance of the third. For instance, optimizing for data rate and latency tend to decrease reliability. Another challenge is that improvements on these goals tends to increase usage of the spectrum. Increasing data rate tends to require higher bandwidth. Decreased latency relies on the channel being open for a transmission shortly after it is ready, reducing efficiency. Reliability tends to require increased redundancy.

One proposed method of addressing these problems is to use additional spectrum. The challenge with increased spectrum use is that spectrum allocation is statically handled by the FCC in the United States and by similar bodies elsewhere. Most lower frequency bands have already been allocated. While there are a few bands that allow unlicensed transmissions (such as the ISM bands in 2.4 and 5 GHz) these bands tend to be oversubscribed. While lower frequencies are mostly taken, there is a large section of bandwidth around 60 GHz approved for unlicensed use by the FCC. This band originally included 7 GHz of bandwidth for unlicensed use from 57 to 64 GHz [5] but was later extended to include another 7 GHz from 64 to 71 GHz [6].

Moving to 60 GHz (which is in a portion of the spectrum commonly referred to as millimeter wave or mmWave) presents its own distinct set of challenges and benefits. Unlike lower frequency radios, mmWave radios can take advantage of small form factor antenna

arrays capable of providing large gain through directional beam patterns¹. Antenna arrays operate by shifting the phase of the signal at each element to form different antenna patterns, some of which are directional. By introducing the ability to vary these phase shifts, electronically steerable antenna arrays can change their antenna pattern without mechanical parts. The addition of amplitude control to each of the antenna elements further increases the flexibility of the antenna to produce directional patterns with less power outside the primary directional beam. Most equally spaced linear planar antenna arrays are built with vertical and horizontal antenna spacing less than the carrier wavelength to avoid grating lobes [9]. Because mmWave signals have wavelengths on the order of millimeters, antenna arrays with a large number of elements can fit in an area small enough to be integrated with portable consumer electronics such as cell phones.

The potential for small form factor antenna arrays with many elements opens the possibility for increased use of spatial diversity in mmWave wireless communications. Traditionally, device form-factors have restricted the use of antenna arrays with many antenna elements. WiFi end devices are often limited to less than four antennas with access points capable of more [10]. While this allows for some use of spatial diversity, the limited number of antenna elements impact the degree of improvement. By contrast, a 12 element Tx and 12 element Rx antenna array can fit in a 4×4 cm² area. However, one key restriction for most mmWave arrays is that there is usually a limited number (one in many cases) of data streams provided to the RF frontend of the array. The phase shifting and aptitude adjustments for each antenna element are performed in the analog domain and are not directly accessible to the baseband. By contrast, lower frequency arrays can often afford separate phase coherent data streams for each antenna element.

While there are many methods of adjusting the phase coefficients of an antenna array, one prominent method is to set the coefficients such that directional high gain beams are formed. In addition to increasing the gain of desired signals, directionality also reduces the gain of potentially undesired signals outside of the main beam. At the transmit side, this means that interference caused at radios outside of the main beam is reduced. At the receive side, this means that interfering transmissions from radios outside of the main beam are attenuated². While this effect has the potential to decrease interference to other users and

¹Common questions concerning antenna requirements for mmWave radios come from the evaluation of Frii's transmission equation. Frii's transmission equation states that $P_R = \frac{P_T G_T A_R}{4\pi d^2}$, $A_R = \frac{G_R \lambda^2}{4\pi}$ where P_R is the receive power, P_T is the transmit power, G_T is the transmit antenna gain, A_R is the effective receive antenna area, G_R is the receive antenna gain, λ is the carrier wavelength, and d is the separation between the transmitter and receiver [7]. If the system is operating in the Equivalent Isotropic Radiated Power (EIRP) limited regime, $P_T G_T$ is constant. To maintain the same receive power as a low frequency radio, the receive effective antenna area must be kept constant. If the transmit power is limited rather than the EIRP (G_T held constant) and it is assumed that the transmit and receive antennas are identical, the effective antenna areas required to maintain the same receive power decrease. This is due to the relationship between the effective antenna area (aperture) (A_e), directivity (D), and gain G where $G = \eta_n D$ and $D = \frac{4\pi}{\lambda^2} A_e$ [8]. η_n is the aperture efficiency of the antenna and is the ratio of the effective aperture size to the physical aperture size [8]. Assuming the antenna efficiency remains consistent, maintaining the equivalent antenna area when shifting to higher frequencies results in higher antenna gain.

²These statements are simplifications that apply mainly to the transmit antenna array. The receive antenna array is more nuanced because the Low Noise Amplifiers (LNAs) at each element can be driven into saturation by a strong signal. Even if the phase shifters are set to cancel out a strong interferer by placing a null in interferer's direction, the saturation of the LNAs will prevent the proper reception of the desired

provide increased gain to desired users, it also means that the radio may have insufficient gain to successfully decode signals when the array antenna pattern is close to omnidirectional. While initially considered a disadvantage of wireless systems, the broadcast nature of the medium has been utilized in current wireless standards. Omnidirectional Tx and Rx allows radios to hear AP beacons and helps facilitate association with a network. Listening omnidirectional allows terminals to receive frames destined for them without requiring prior scheduling or coordination of array phase shift coefficients. Directionality can be more easily introduced into the Access Point (AP) / Station (STA) or Base Station (BS) / User Equipment (UE) star topology since each STA / UE only communicates with its associated AP / BS³. However, directionality complicates the development of mesh and ad-hoc networks as radios need to coordinate when and where to direct their electronically steered beams in order to transfer frames at an acceptable rate and reliability.

The benefits and challenges introduced by high element count antenna arrays along with different propagation characteristics in mmWave frequencies leave many open research questions. Many of these questions reside in the link and network layers as well as with more general systems research. How should directionality be exploited in mesh networks? How should the challenges introduced by antenna arrays be addressed in next generation communications systems? While questions like this can be studied with simulation, a physical prototyping and test platform is needed to investigate these questions with high confidence under real world conditions.

1.1 Goals & Scope

The goal of this project is to create an initial version of a 60 GHz development and test platform for evaluating systems level research questions. There are three basic components of a wireless communications system: the baseband IP, the platform on which the baseband runs, and the RF frontend that translates between the baseband and the desired carrier. For 60 GHz, the RF frontend is particularly challenging to produce. Fortunately, there are a few commercially available RF frontends. It was determined that a RF frontend by Sibeam (part of Lattice Semiconductor) would be sufficient for most systems level research. However, when antenna array functionality is not required, an RF front-end from VubIQ can be used. The VubIQ front-end allows standard WR15 waveguide connections and is compatible with standard horn antennas.

There are three typical solutions for the baseband and platform: use an existing baseband implemented on an ASIC, implement a baseband in software and use a SDR platform, or implement a baseband on an FPGA. Each of these solutions has its own distinct set of advantages and disadvantages. Using an existing ASIC requires the least amount of design work but restricts experimentation to what is revealed by the chip's configuration registers. It is sometimes necessary to operate the ASIC in unintended modes of operation to attain the desired behavior, a task that can require substantial amounts of reverse engineering. An alternative is to develop the baseband in software and to use an existing SDR platform. The platform abstracts away many of the details about the Analog to Digital Converter

signal. This effect places stronger requirements on LNA linearity and dynamic range than may be expected.

³AP/STA is used in IEEE 802.11 documents while BS/UE are used in 3GPP documents.

(ADC) and Digital to Analog Converter (DAC) but is limited by the computational power of the host computer and the communication link between the host computer and the SDR hardware platform. While this method can take advantage of the generally faster software development time, it faces limitations on supportable bandwidth and latency. Also, while initial prototyping development can be fast, producing a high-performance implementation often requires careful considerations of the host platform and increased development time. A carefully designed FPGA implementation can often outperform a software implementation in both the data rate achieved and in latency. This performance can be accomplished by leveraging the fine grain parallelism provided by a pipelined design operating in parallel on hardware. However, while FPGAs are capable of being reconfigured, the design process has much more in common with ASIC hardware design than software design. The FPGA design process often requires additional design and implementation time relative to software design processes. For this project, an FPGA based implementation was chosen due to the large volume of baseband sample data and the potential for lower latency operation. In projects where a host system is used for higher level control, the FPGA design offloads the baseband processing from the host system, freeing resources for high level control. The baseband was developed to be as modular as possible to allow for modifications as experiment requirements change over time.

1.2 Baseband Specification

The baseband specification is motivated by its intended use as a component of several mmWave research projects. While most commercial basebands prioritize performance and efficiency, systems researchers often desire the ability to modify certain functions of the underlying baseband to test a new concept. This desire led the prioritization of modularity and simplicity of design over performance to assist in the easy adoption and modification of the baseband by different mmWave research projects with varied objectives and areas of expertise. The sampling rate was chosen to both match the rate of medium range ADC/DAC cards as well as to be within a reasonable clock range for an FPGA design. A breakdown of the specification is given below:

- 250 MS/s Sample Rate
- 4x Oversampling (62.5 MS/s Symbol Rate)
- BPSK Modulation (62.5 Mbps) with Options for Future Expansion to Higher Order Modulation Schemes⁴
- Control PHY Preamble from 802.11ad
- Fixed Frame Size

⁴See chapter 8 for more details on supporting higher order modulation schemes.

1.3 Tools

The baseband was designed and implemented in Mathworks Matlab/Simulink platform with Finite State Machines (FSMs) being implemented in Mathworks Stateflow. Mathworks HDL Coder was used to translate the Simulink flow-graphs to synthesis ready VHDL. A Xilinx ZC706 development platform was used as the base of the system with a FMC151 card by 4DSP used to provide ADC and DAC functionality. Xilinx Vivado was used to integrate the generated VHDL from Simulink with a modified version of the 4DSP ADC/DAC IP to allow streaming operation. Xilinx SDK was used to write software for the ARM processor on the Zynq FPGA. The code was a merger of the configuration code provided by 4DSP with custom functions to exercise the baseband IP.

Chapter 2

System Architecture

There are several popular methods for constructing communications platforms: using an existing ASIC solution, implementing a software baseband and utilizing an SDR platform, or implementing the baseband on an FPGA that interfaces with data converters. As was discussed in section 1.1, there are advantages and disadvantages to each scheme. Due to the volume of data delivered and the potential for low, more predictable, latency, an FPGA based approach was taken.

2.1 Hardware Platform

There are several FPGA based hardware platforms available that are capable of supporting baseband research. These platforms range from fully integrated solutions primarily intended for SDR usage to more general purpose platforms. While it is tempting to use a platform intended for SDR applications, there are some drawbacks for this project. Many SDR platforms are designed with the assumption that most users will implement the baseband in software and rely on the FPGA platform for simple glue logic between the host PC, data converters, and RF frontend. Part of the value proposition is that the details on how the platform operates are abstracted away from the baseband designer, allowing them to focus on their specific task. While this abstraction provides a productivity boost to SDR developers, it has made integration of custom blocks on the FPGA a relatively niche use case that is often less documented. Fortunately, this has begun to change recently with projects like Ettus Research's RFNoC which aim to help streamline the process. However, these projects are still in the early stages of development and were not widespread at the time the hardware platform for this project was chosen. It was also the case that, until recently, many SDR platforms did not contain FPGAs that were sized to allow extensive custom block insertion. This is due in part to the cost of large, high-performance, FPGAs and also in part to the use of the FPGA as glue logic for many SDR platforms. Finally, some SDR platforms expect to provide the full hardware solution including the RF frontend which complicate the use of 60 GHz RF frontends¹.

¹There are exceptions such as the BasicTx and BasicRx boards from Ettus Research which allow direct access to the DAC and ADC on certain USRP platforms. However, these cards are limited to the sample rate of the USRP's on-board ADCs and DACs.

In light of these observations and with the goal of selecting a flexible hardware platform, a Xilinx ZC706 Development Kit was chosen as the base FPGA board for the project. As a general purpose Xilinx Development Kit, the ZC706 does not include any RF ready ADCs² or DACs. However, the ZC706 board was designed to be expandable with two FGPA Mezzanine Card (FMC) connectors, one High Pin Count (HPC) and one Low Pin Count (LPC) [11]. There are several vendors producing FMC cards for a variety of applications including data conversion (ADC/DAC). Most of these vendors supply IP blocks along with the card to expedite integration with the customer’s project. An advantage of using FMC cards to provide the ADC/DAC functionality is that different cards with different capabilities can be acquired depending on the requirements of the project. It also provides a path for developing more specialized FPGA based hardware platforms using custom FMC cards as a stage in the prototyping process. In addition to the FMC header, a SFP+ connector allows the platform to be extended with high speed wired networking [11]. The ZC706 also includes several convenient peripherals including a gigabit Ethernet port, SD Card slot, USB to UART adapter, and USB to JTAG adapter [11]. At the core of the ZC706 is a high end Zynq7000 FPGA which includes an on-chip ARM Cortex A9 processing System coupled with a Kintex7 equivalent FPGA fabric [11] [12]. This provides the flexibility to run certain functions on the ARM processor which is able to communicate with IP blocks instantiated in the programmable logic portion of the FPGA via high speed AXI buses.

There are several off-the-shelf ADC/DAC FMC cards that could have been used to provide the Analog/Digital interface required to interface the digital blocks on the FPGA to the analog baseband connections on the RF frontend. Cards vary in the number of channels, the sampling rate, and the resolution of the on-board ADCs and DACs. The FMC151 card from 4DSP met the specifications of the proposed baseband and was selected to provide the ADC/DAC functionality to the platform. The FMC151 includes a dual channel, 16 bit, DAC operating at 800 MS/s and a dual channel, 14 bit, ADC operating at 250 MS/s. On-chip interpolating filters allow the DAC to ingest data at the same sample rate as the ADC and still operate close to its maximum sample rate. To expedite adoption of the FMC151 into designs, 4DSP provided a reference FPGA design as well as a C# program running on a PC to facilitate initialization of the FMC151 as well as sample data exchange between the host PC and the FPGA. While the reference design and software provide a starting point, several modifications to the 4DSP IP and software were required before the baseband blocks could be integrated. The two channels from the DAC are fed to the Tx I and Q inputs of the RF frontend while the Rx I and Q outputs of the RF frontend are fed to the two channels of the ADC.

An image of the base hardware platform, excluding the RF frontend, is shown in Figure 2.1.1.

2.2 SoC Top Level Design

The top level SoC design, depicted in Figure 2.2.1, provides the framework for how the different components of this research platform are integrated together. While the baseband

²The ZC706 does include a low speed 1Ms/s ADC (XADC) that is primarily used for monitoring sensors and voltages.

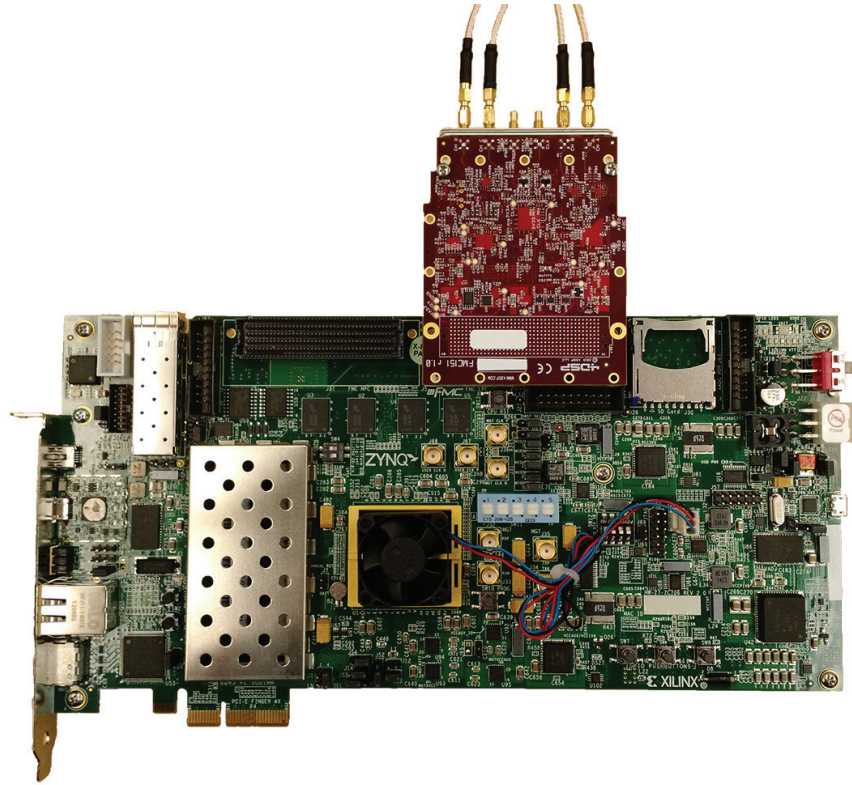


Figure 2.1.1: Xilinx ZC706 (Green, Bottom) and Connected 4DSP FMC151 (Red, Top)

Tx and Rx blocks are the central focus of this project, they cannot operate alone. At a basic level, the Tx block must be provided with data to transmit and the Rx block must have a location to deliver received data to. To operate as a real-time radio, the Tx component must be able to stream data to the DAC. Likewise, the ADC should be streaming data into the Rx block to be processed. Finally, the Tx and Rx blocks have a set of configurable parameters which need to be set properly for correct operation. The top level design provides mechanisms to fulfill each of these requirements.

The data input to the Tx block and the data output of the Rx block both closely resemble AXI Streaming interfaces. Through the use of a small shims, the Tx and Rx blocks can be interfaced with other AXI Streaming blocks. This allows for wide flexibility in how data can be passed between the Tx and Rx IP. For example, an Ethernet interface and some custom logic could be used to stream data directly to and from the Tx and Rx blocks respectively. Alternatively, Xilinx provided AXI Memory Mapped to Streaming FIFOs can be used to allow the ARM processor (which uses Memory Mapped IO) to send packets to the Tx block and to receive packets from the Rx block. The latter approach was taken at this stage of the project as it allowed flexible processing of the transmitted and received data on the ARM processor.

The 4DSP FMC151 IP block provides the interface between the ADC and DAC sitting on the FMC151 and other blocks residing on the FPGA. The FMC151 IP was modified to permit streaming operation in place of the burst mode operation that was implemented. FIFOs are used to buffer data between the data converters and the baseband blocks.

Like most ADCs and DACs, the ones present on the FMC151 provide several Control Status Registers (CSRs) that must be set in a specific sequence to prepare the data converters for proper operation. 4DSP provided a set of IP blocks that are used to communicate with the ADC and DAC along with other ICs present on the FMC151. A modified version of the 4DSP provided C# application running on a host PC provides the sequence of register reads and writes required to setup the card. These commands are exchanged over the Ethernet interface with the ARM processor which runs a bare metal ARM application. This application uses the Xilinx Lightweight IP (LwIP) library to create a sever which listens for these commands and relays them to the appropriate configuration IP on the FPGA.

Configuration of the Tx and Rx blocks is accomplished by the inclusion of several Control Status Register (CSR) blocks. These blocks contain registers which can be set and read using the memory mapped AXI interface that is accessible by the ARM processor. The outputs of these registers are connected to the configuration ports of the Tx and Rx baseband blocks. The provided 4DSP bare metal ARM application was modified to include the baseband initialization as well as to generate and parse data frames exchanged with the Tx and Rx baseband blocks respectively. Telemetry is sent to the host PC via UART.

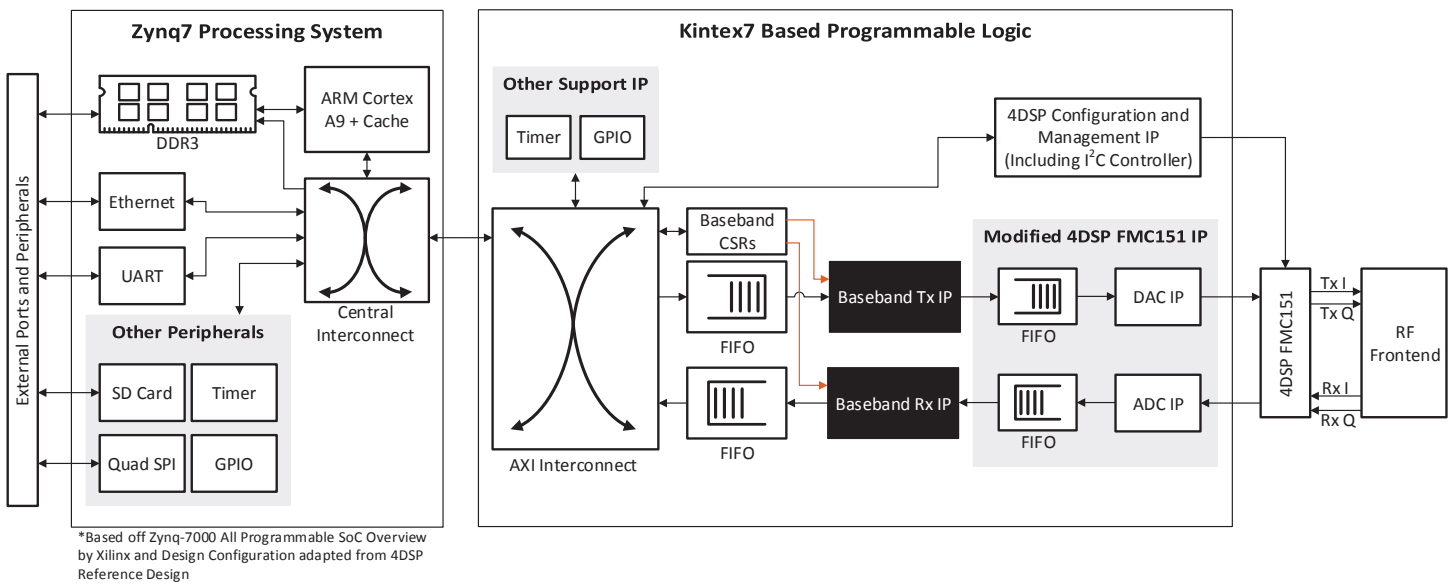


Figure 2.2.1: System Architecture

2.3 RF Frontend

Designing RF frontends that operate at 60 GHz has historically been a very difficult task. Fortunately, recent advances have made mmWave RF front-ends economically viable for consumer applications. However, due to the nascent state of mmWave solutions, relatively few commercial mmWave RF front-ends are available. RF frontend chips from Sibeam and RF frontend modules from VubIQ were selected for this project due to their feature sets and availability. Due to the added complexity of integrating with the Sibeam chips, the project initially focused on integration with the VubIQ RF frontends. Integration with the Sibeam chips is planned for future iterations and is discussed in chapter 8. While the design of the RF frontend is not the focus of this project, its implementation has influenced several aspects of the system design. As such, some basic information about the RF frontend operation is detailed here. A simplified architectural diagram of the RF frontend omitting some of the optional features and configurable components is shown in Figure 2.3.1. The operation of the QAM modulator and demodulator are discussed in appendix A and appendix B respectively.

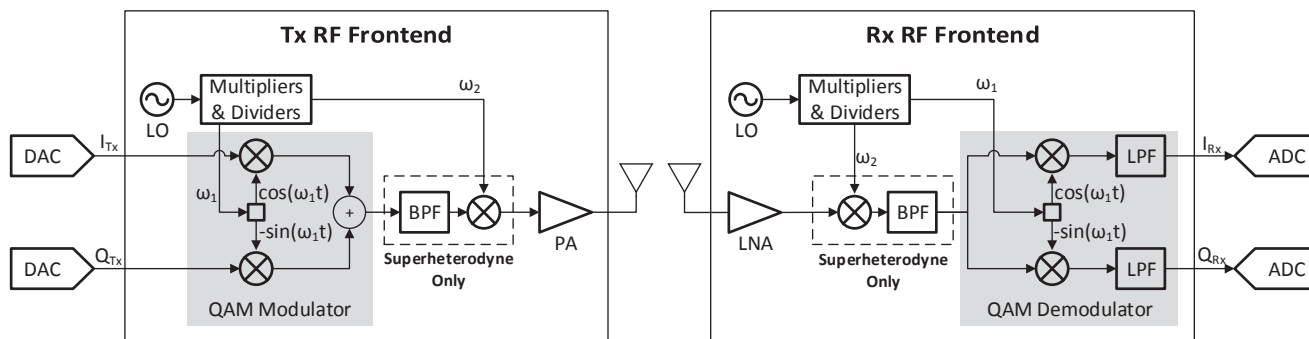


Figure 2.3.1: Simplified RF Frontend Architecture

2.3.1 Features

The VubIQ solution uses a “double conversion superheterodyne architecture with a sliding I” in both the Tx and Rx module [13] [14]. The first stage of the Tx module consists of a QAM modulator that takes in baseband I and Q and produces an Intermediate Frequency (IF) signal³ The IF signal is then filtered and attenuated before being passed to another mixer to be shifted to the desired carrier frequency [13]. This signal is then sent through a power amplifier (PA) before passing to the antenna [13]. The receiver first amplifies the signal from the antenna with a low noise amplifier (LNA). This signal is then downconverted to IF where it is filtered and attenuated before passing through through a QAM demodulator⁴. The resulting I and Q signals are then passed through a final filtering and attenuation stage before being delivered to the baseband outputs [14].

³The QAM modulator has an optional FM mode which can be used to frequency modulate the I and Q signals with two other signals. This mode is unused in this project.

⁴The receiver contains optional AM and FM detectors that are unused in this project.

Chapter 3

Tx Baseband Architecture

The Tx Baseband IP, outlined in Figure 3.0.1, is responsible for converting a stream of binary data into a stream of samples to be sent to the DAC. It begins by mapping binary digits to fixed point numbers on the complex plane. These samples are then up-sampled by 4x and run through a square root raised cosine filter which acts both as a pulse shaping filter as well as in interpolating filter. The resulting samples are then scaled by the desired gain and shifted to counter any DC offset at the DAC.

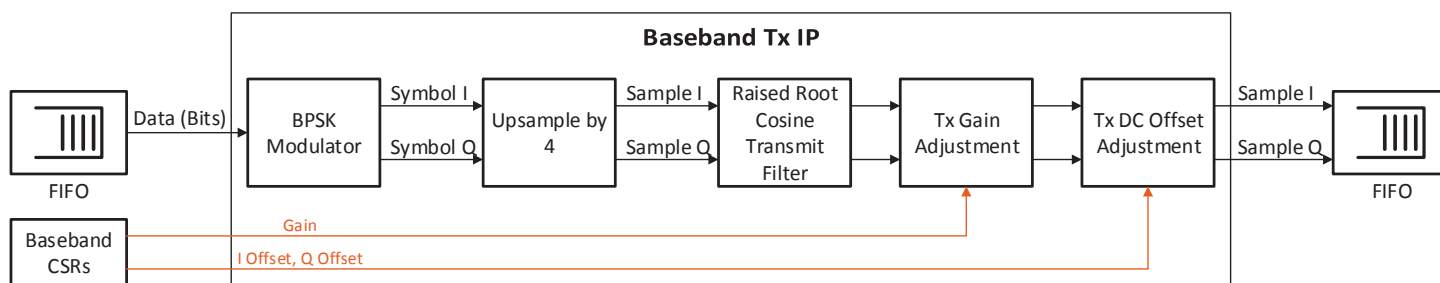


Figure 3.0.1: Baseband Tx Architecture

3.1 BPSK Modulator

The general purpose of digital modulators is to transform digital information into waveforms that can be transmitted. There are a wide range of methods to perform this transformation including (but not limited to) [7]:

- Frequency Shift Keying (FSK): maps digital values onto tones at specific frequencies
- Phase Shift Keying (PSK): maps digital values to different phases of a signal
- Amplitude Shift Keying (ASK): maps digital values to different amplitudes of a signal

The selection of the modulation scheme depends in part on the capabilities of the RF front-end.

Many RF frontends include analog QAM modulators and demodulators. These QAM modulators provide a useful abstraction to baseband radio designers by splitting the baseband signal into two components: an in-phase component (I) and a quadrature component (Q). These two analog signals are often viewed as components of a complex signal with the I channel considered the real component and the Q channel considered the imaginary component. A point on imaginary plane can be interpreted as a phasor with a given phase ($\arg(I+Qj)$) and magnitude ($\sqrt{I^2 + Q^2}$). Because of this, QAM modulators allow baseband designers to easily use modulation schemes that are hybrids of phase and amplitude modulation. Details about the operation of the QAM modulator and demodulator are discussed in appendix A and appendix B respectively.

When presented with a QAM modulator and demodulator, it is common to describe the transformation of digital information into waveforms as a mapping of digital values onto points on the complex plane. These points are commonly referred to as *constellation points*. Data values mapped to these points are commonly referred to as *symbols*. As noted above, each point can be viewed as a phase with a specific amplitude and phase. Binary Phase Shift Keying is one such mapping where signal binary digits are mapped to one of two points on the complex plane. These points are symmetric about the origin and are often placed on the real axis. The constellation plot for BPSK modulator used by this project is shown in Figure 3.1.1. The decision about which binary digit maps to which point is mostly arbitrary for BPSK but has more impact in higher order constellations with more points.

In order to transmit a digital message, it is first broken into individual bits. These bits are then converted to symbols by mapping the binary bits onto points on the complex plane. These symbols are then streamed, in sequence, to the latter modules of the Tx baseband. Because the modulation operation is a static mapping, its hardware implementation is a simple lookup table. This makes the modulator one of the simplest components in the baseband.

3.2 Square Root Raised Cosine Transmit Filter

The principle purpose of the square root raised cosine transmit filter is to act as a pulse shaping filter. The signal streaming out of the modulator can be viewed as a sequence of pulses with each pulse having a specific complex value given by I and Q. If each pulse is passed to the DAC as a single sample, the resulting waveform would consist of a series of sharp pulses with no separation between them. In general, abrupt changes in time have wide spectral footprints. In reality, there are bandwidth limitations placed on the transmitted signal. Some of these limitations come from regulatory bodies like the FCCs while others come from the physical limits of the devices used in the radio. Band limiting causes symbols to spread in the time domain and to overlap with adjacent symbols. This overlap of spread pulses is called Intersymbol Interference (ISI) [15]. The amount of overlap depends on the symbol rate as well as the bandwidth of the band-limiting channel. While any overlap is undesirable, at symbol rates approaching or exceeding the bandwidth of the channel the resulting overlap can be so severe that individual pulses may not be distinguishable from each-other [7].

To limit the effects of intersymbol interference, pulse shaping can be applied to the signal

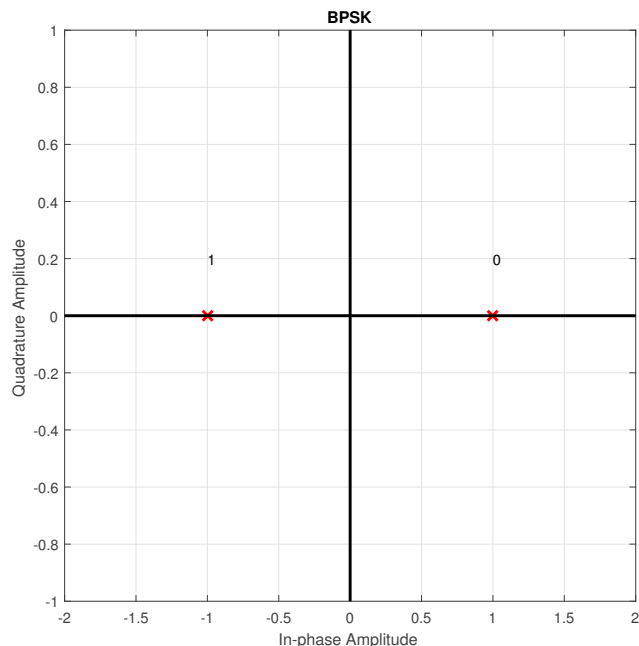


Figure 3.1.1: BPSK Constellation Plot

before it is transmitted. Ideally, zero inter-symbol interference is desired. Nyquist has shown that a necessary and sufficient condition for a signal pulse, $x(t)$ with a symbol period of T to have zero ISI is [15] [7]:

$$x(nT) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} \quad (3.1)$$

Intuitively, this means that a symbol pulse should have unity gain at its center but should have zero magnitude at the center of other pulses. For this to be accomplished the symbol period, T , must satisfy the condition that $1/T \leq W/2$ where W is the bandwidth of the channel [7]. This means that the signal must be oversampled by at least a factor of 2 for the filter to be realizable in the digital domain. If the DAC operates at the Nyquist rate with $1/T = W/2$, the only filter that accomplishes this is the ideal low pass filter (LPF) for $W/2$: $\text{sinc}(\pi t/T)$ [7]. An ideal sinc filter for an oversampling rate of 4x (corners at $W/4$) is shown in Figure 3.2.1. There are several problems with the ideal LPF filter. One problem is that the ideal filter is non-causal IIR which must be truncated and shifted for practical implementation. Another problem is that the peaks fall off at a relatively slow rate of $T/(\pi t)$ [7] [16]. If there is a timing error in the receiver, it may not sample at the ideal point where the current symbol has unity gain and other symbols have a gain of zero. This results in ISI being present at the receiver [7] [16]. The slower this convergence to zero, the worse this effect is.

For $1/T < W/2$, other filters can be used. One of the most commonly used transmit filters is the raised cosine filter [7]. The raised cosine filter relaxes the requirements of the ideal LPF by allowing its frequency response to extend outside the desired range. The frequency

response of the raised cosine filter for a rolloff factor $0 \leq \beta \leq 1$ is [7]:

$$X(f) = \begin{cases} T & , 0 \leq |f| \leq \frac{1-\beta}{2T} \\ \frac{T}{2} \left(1 + \cos \left[\frac{\pi T}{\beta} \left(|f| - \frac{1-\beta}{2T} \right) \right] \right) & , \frac{1-\beta}{2T} \leq |f| \leq \frac{1+\beta}{2T} \\ 0 & , |f| > \frac{1+\beta}{2T} \end{cases} \quad (3.2)$$

The raised cosine filter's smoother frequency response makes it easier to approximate with practical filters [7]. An example impulse response of a raised cosine filter with 4x oversampling is shown in Figure 3.2.2. Note that every fourth sample has an amplitude of zero and that the peak roll-off is substantially faster than the ideal LPF.

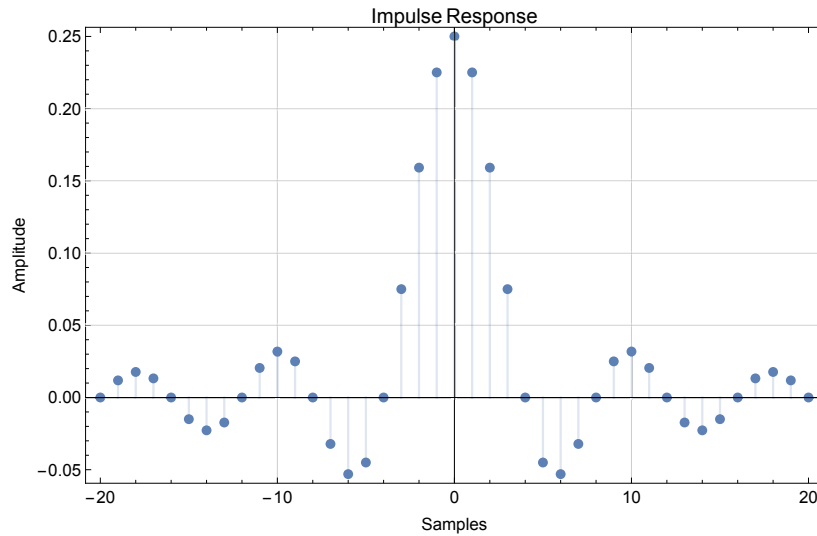


Figure 3.2.1: Truncated Ideal Low Pass Filter for 4x Oversampling

Even though the raised cosine filter limits ISI in the transmitted signal, it is not the pulse shaping filter used in this design. This is due to the fact that both the transmitter and receiver include matched filters. Ideally, the combination of the two filter should minimize ISI. A common method to achieve this is to take the square root of the raised cosine filter and to use that as the pulse shaping filter [17]. More specifically, $G_{Tx}(f) = \sqrt{|G_{RC}(f)|}e^{-j2\pi ft_0}$ where t_0 is a delay to allow the filter to be realizable [7]. The receive filter is then matched to this transmit filter. The frequency response of the matched receive filter is the complex conjugate of the frequency response of the transmit filter ($G_{Rx}(f) = G_{Tx}^*(f)$). Since the transmit and receive filters can be viewed as cascaded LTI filters, their joint frequency response is $G_{Tx}(f)G_{Rx}(f) = G_{RC}(f)$. [7]. This filter is commonly referred to as the *root raised cosine filter*. An example response is shown in Figure 3.2.3. Note that no samples have an amplitude of zero.

The radio design used in this project makes use of matched filtering in both the transmitter and receiver. Due to this separation, the root raised cosine was used as the pulse shaping filter so that the combined effect of the transmit and receive filters approximate the response of a raised cosine filter. An oversampling factor of 4x was primarily selected to ease timing recovery at the receiver. However, this oversampling factor has the added benefit of allowing

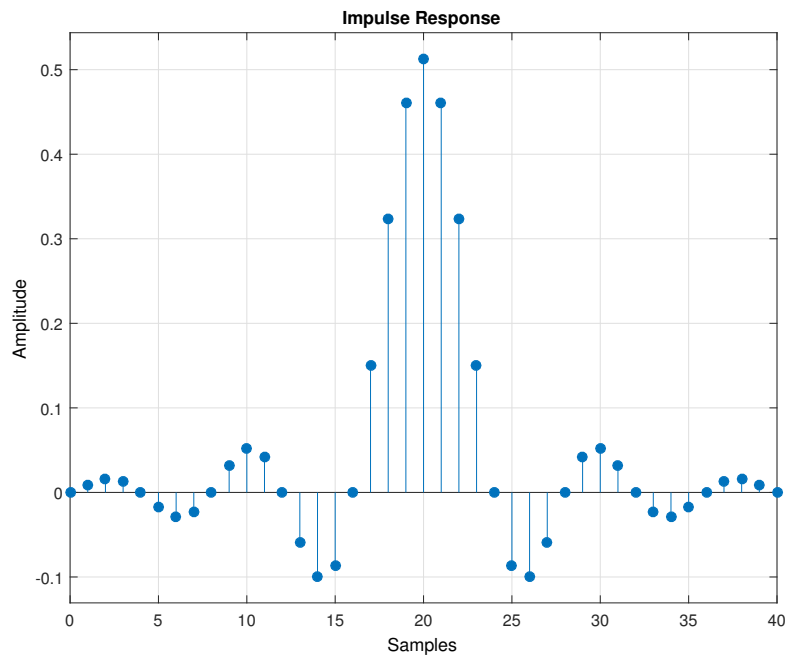


Figure 3.2.2: Raised Cosine Filter over 10 Symbols with 4x Oversampling and $\beta = 0.2$

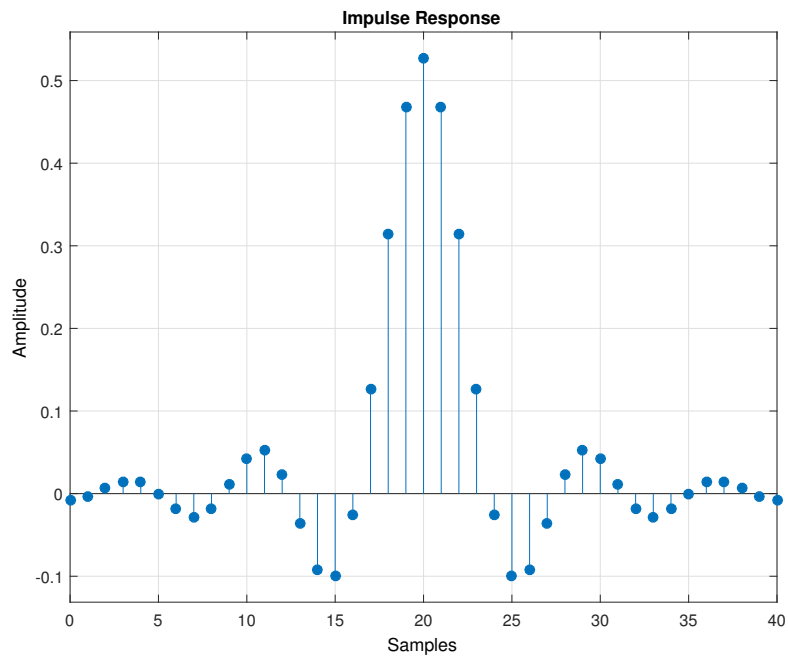


Figure 3.2.3: Root Raised Cosine Filter over 10 Symbols with 4x Oversampling and $\beta = 0.2$

the implementation of the root raised cosine filter with a rolloff $\beta > 0$. To achieve the 4x oversampling with the pulse shaping, the symbol stream coming from the BPSK modulator is up-sampled by a factor of 4 by inserting 3 zeros in between each sample. The resulting stream is then passed through the raised cosine filter to be shaped. The raised cosine filter effectively accomplishes the task of the interpolating low pass filter which is commonly used when up-sampling to interpolate the values of the inserted zeros. To calculate the coefficients used in the root raised cosine filter, the corresponding Simulink block [18] was used.

Chapter 4

Rx Baseband Architecture

The Rx baseband architecture, outlined in Figure 4.0.1, is broadly responsible for detecting and decoding data frames from samples received from the ADC. Compared to the Tx block, the Rx block has several more components as it must correct for the various RF impairments present in order to successfully decode a frame. The received samples are first shifted to correct for any DC offset at the ADC and are scaled if appropriate. The sample then pass through the square root raised cosine receive filter which is the matching counterpart to the square root raised cosine transmit filter in the Tx block. An AGC loop then estimates the received signal power and supplies a scaling factor to bring the received samples to the expected scale. In future versions, the correction factor could be used to set the gain of an analog amplifier in the receive chain of the RF front-end. The samples are then fed to the timing recovery block which selects the optimum sampling point for the signal and signals it using a “strobe” line. This line is used to drive the “push” control signal of a clock crossing FIFO. The result is that one sample per symbol period is enqueued onto the FIFO with the enqueued sample being taken at the optimal sampling point. The entries in the FIFO are now referred to as *symbols* rather than samples. The remaining portion of the Rx baseband is able to operate at a rate only slightly faster than the symbol rate. The symbols then pass from the FIFO through the carrier recovery block which corrects for the carrier phase and frequency offset. These corrected symbols then pass through a Golay correlator. The thresholded magnitudes of the correlator outputs are used by an FSM to determine when a frame is being received and when the preamble is over. In parallel with the correlation, the symbols are passed through a demodulator which maps points on the complex plane to binary digits. A smaller FSM is used to provide a “valid” flag for the duration that the demodulated data is valid. For this design, the payload has a fixed length resulting in the valid flag remaining high for a fixed number of symbols after the preamble.

4.1 Square Root Raised Cosine Receive Filter

The square root raised cosine receive filter is the matched filter to the raised cosine transmit filter present in the Tx baseband. As was discussed in section 3.2, the raised cosine filter minimizes inter-symbol, interference (ISI). Ideally, the joint response of the Tx pulse shaping filter and the Rx matched filter should approximate raised cosine filter. This is

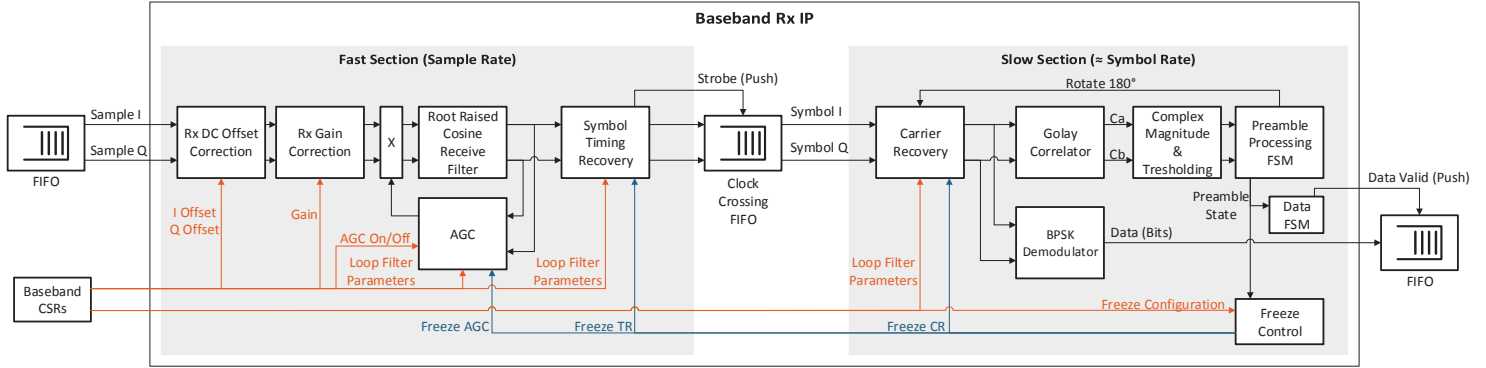


Figure 4.0.1: Baseband Rx Architecture

accomplished by splitting the raised cosine filter into a square root raised cosine filter at the transmitter and a matched filter at the receiver.

In addition to the ISI minimization that the transmit filter and receive root raised cosine filters jointly accomplish, matched filtering also provides additional benefits. Signals propagating through a wireless channel are subject to additive noise and interference. Interference and noise that is not filtered by the analog receive filters will be present in the digital domain after being sampled by the ADC. Ideally, the effects of noise and interference should be minimized to so that the desired signal can be reliably decoded. Matched filtering is a common method used to minimize the effects of noise.

Matched filtering is often introduced in the context of optimal receivers for signals passing through an Additive white Gaussian Noise (AWGN) channel. Proakis takes this approach in several of his texts [7] [15]. Decoding of various signals is often initially described by taking the correlation of the incoming signal with the pulse shapes of the possible symbols defined by the chosen modulation scheme [15]. These correlators can be replaced with matched filters whose impulse responses are the time reversed versions of the pulse shapes [7]. A key property of matched filtering is that it maximizes the SNR of a signal that passed through a AWGN channel [7].

Matched filtering can also be viewed in the frequency domain. The frequency response of the matched filter is $H_{matched}(f) = H_{orig}^*(f)e^{-j2\pi fT}$ [7]. The magnitude of the frequency response matched filter matches that of the original filter, $|H_{matched}(f)| = |H_{orig}(f)|$. Recall that white noise has a flat frequency response. Intuitively, it makes sense to filter the incoming signal with a filter whose magnitude response matches that of the pulse shaping filter. Matching the amplitude response in the receive filter avoids attenuating frequencies that are part of the shaped pulse and attenuates frequencies that are not present in the shaped pulses. Since AWGN noise has a flat frequency response, this allows some components of the AWGN noise to be filtered out while maintaining frequencies that are used by the shaped pulse. Proakis provides an in depth derivation of the frequency domain view of matched filtering and shows that it has equivalent results to the time domain analysis [7]. The raised cosine receive filter Simulink block was used to calculate the matching filter coefficients for this design [19]. The frequency response of the transmit filter to which this filter is matched is shown in Figure 4.1.1. Note that the corner frequency occurs at a normalized frequency

of $1/4$ which corresponds to the oversampling rate.

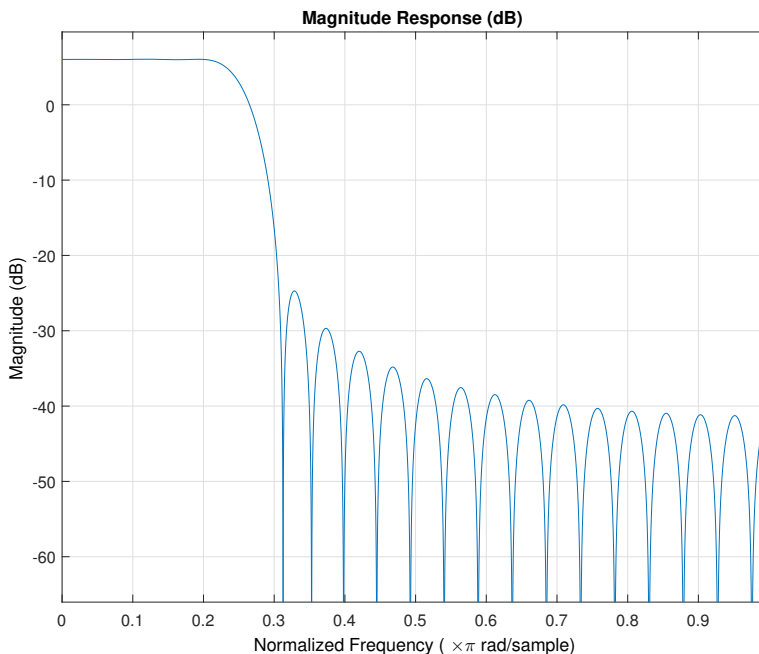


Figure 4.1.1: Root Raised Cosine Filter over 10 Symbols with 4x Oversampling and $\beta = 0.2$

4.2 AGC

Blocks in the receiver chain are generally designed with the expectation that the input waveform is scaled to a specific magnitude. Some blocks require this scaling to make proper decisions while other blocks, such as synchronizers, may have coefficients that are dependent on signal power [20]. M-QAM modulation schemes in particular require proper signal scaling for reliable decoding [20]. Although M-QAM is not currently implemented in the design, it is planned for future iterations. In addition, many blocks require the scaling due to dynamic range concerns. The basic data types used throughout the DSP portion of this design are fixed-point. Fixed-point number representations use fixed numbers of bits to represent the integer and fractional parts of a number. This is in contrast to floating-point numbers which are more similar to scientific notation with a fixed number of bits used to represent the significand and a fixed number of bit used to represent the exponent [21]. While there are several well known numerical analytic issues when using floating point numbers, they have wider dynamic range than fixed point numbers with the same bit width. Floating point arithmetic is generally more computationally expensive than fixed point arithmetic and requires more resources to implement effectively on an embedded platform or FPGA. There has been some recent momentum in integrating hard floating point blocks into FPGAs with Intel/Altera including hard single precision floating point blocks in the Aria 10 and Stratix 10 FPGAs [22] [23]. However, this technology is still emerging and was unavailable at the

time this design was created. Because blocks were designed with fixed point data types, they operate best when the input data is scaled appropriately.

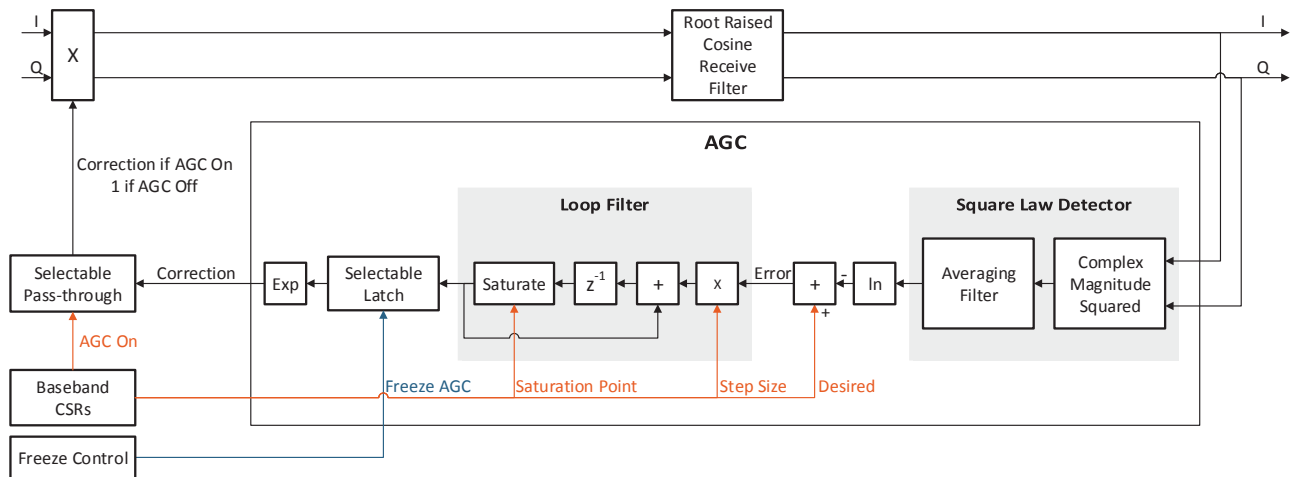


Figure 4.2.1: AGC Architecture

The Automatic Gain Control (AGC) block is responsible for adjusting the receiver gain to attain a desired receiver power level. At its core, the AGC is a control system that estimates the power level of the input signal, compares it to a desired value, and produces a correction factor that is used to either drive an external variable gain amplifier or is used to scale the signal digitally. The AGC used in this design was adapted from the Simulink AGC block [24] [25]. The original Simulink AGC block is a logarithmic loop AGC [25]. The signal passes through a square law detector which produces a value that is proportional to power. The log of this value is taken which produces a value proportional to the power in dB. This is compared against the desired value in dB to produce an error term which is used in the loop filter. The natural exponent of the loop filter result is used as the correction factor. A diagram of the AGC design is shown in Figure 4.2.1.

The original Simulink AGC design utilized a feedforward structure which took the estimate of the signal power level before the correcting multiplication [25]. While this structure has some desirable properties such as faster convergence time and improved stability, it has the disadvantage of requiring wider dynamic range [26]. The feedback structure not only reduces the dynamic range requirements but is also compatible with schemes that adjust an analog variable gain amplifier (VGA) before the AGC. While an external VGA is not easily accessed from the baseband with the VubiQ radios, it is a feature that is expected in other RF frontends. As a result, the implemented AGC block follows a feedback structure.

The logarithm is an important aspect of this AGC design. An AGC operating on signal power alone has a convergence time that is inversely proportional to the input level of the signal [26] [20]. This is undesirable as the convergence time of the AGC would depend on the level of the received signal and would be slow for weak signals. The logarithmic AGC, by contrast, has a convergence time that is independent of the input signal level [26] [20]. This simplifies specifying the length of the preamble as the AGC settling time is constant.

The AGC includes configurable threshold in the loop filter to prevent inputs from being driven outside a usable range. For low noise scenarios where no signal is being received, an unbounded AGC can apply excessive gain that multiplies the input signal so that the high order bits are lost. The AGC detector may not be able to detect an increase in signal power at this point. Saturation can be used to limit the AGC gain so that no high order bits are lost.

There are some cases when a user may want to set gain manually. To accomplish this, a selectable pass-through can be disabled to block the AGC correction from reaching the complex multiply. Likewise, there are cases when a user may want the AGC level to remain frozen after the preamble. A *freeze* line is provided to optionally latch the AGC correction value for this purpose.

4.3 Timing Recovery

There are generally two separate sets of oscillators present in a radio system. One set produces the carrier frequency and is used to mix the baseband signal up to the carrier at the transmitter and to downconvert from the carrier back to baseband at the receiver. The second set of oscillators control the sampling clocks for the DAC at the transmitter and the ADC at the receiver. In practice, there is some variance between identically specified oscillators that cause them to operate at slightly different frequencies and phases [27]. Correcting for these oscillator differences is exceptionally important for reliable reception of coherent modulation schemes. Correction for the mixer oscillator is handled by the Carrier Recovery block (section 4.5). This section details the correction for the sampling oscillators driving the DAC at the transmit side and the ADC at the receive side.

The core problem addressed by timing recovery is to determine when the received waveform should be sampled to properly detect a symbol [7]. Recall from section 3.2 that the transmitted signal can be viewed as a pulse train with pulses shaped using the root raised cosine transmit filter. After passing through the root raised cosine receive filter and the AGC, the signal can be viewed as a normalized pulse train with a raised cosine pulse shape. Ideally, the waveform should be sampled at the point that maximizes the likelihood of proper decoding. Intuitively, the signal should be sampled where the output of the matched filter is maximized on average. For most schemes, this should be at the center of the pulse as the matched filter can be viewed as a correlation of the received signal against the pulse shape. The fact that the raised cosine filter response produces nodes at the center of adjacent symbols to mitigate ISI further reinforces this notion that symbols should be sampled at the center of their respective pulses. An alternative explanation is that the signal should be sampled at the widest opening of the eye diagram produced at the output of the matched filter at the receiver [20]. Failure to correct for sampling clock errors can significantly degrade the ability of the receiver to properly decode the signal. If it is assumed that the sampling clocks in the transmitter and receiver are synchronized in frequency but not phase, the waveform would be sampled at the same suboptimal sampling point for each symbol. An example eye diagram is shown in Figure 4.3.1. Looking at this eye diagram, the variance in sampled amplitude gets larger as the sampling point gets farther from the optimal sampling point. This produces artificially low SNR for symbols with low magnitude at the

sampling point. On an I/Q plot of the received signal, this would appear as a widening of the cluster of received constellation points. An example of this effect is shown in Figure 4.3.2. If it is no longer assumed that the transmit and receive sample clocks are synchronized in frequency, the receiver will no longer sample at the same point in the symbol waveform. This will result in the received constellation exhibiting time varying spread. It also means that symbols may either be duplicated or skipped in the case of a faster or slower receiver sampling clock, respectively. This can be potentially disastrous to proper decoding as later blocks typically assume that they will be passed one value per symbol. Duplication or loss of symbols results in the symbol stream becoming offset from what is expected. This offset can cause cascading bit errors in the decoded signal. This is in contrast to a simple “bit flip” error which is isolated to individual symbols.

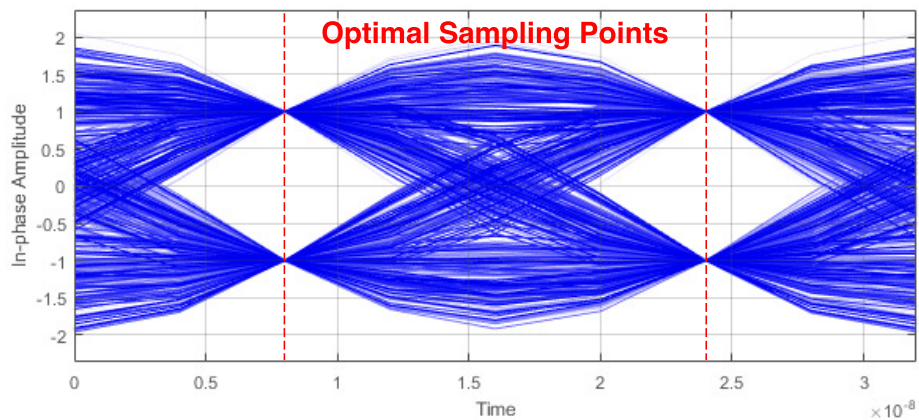
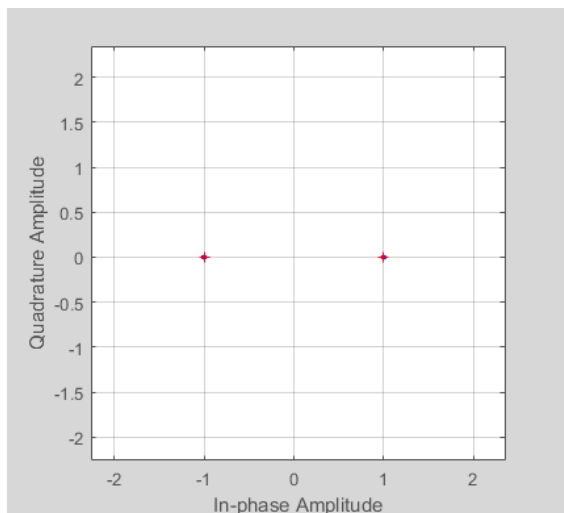
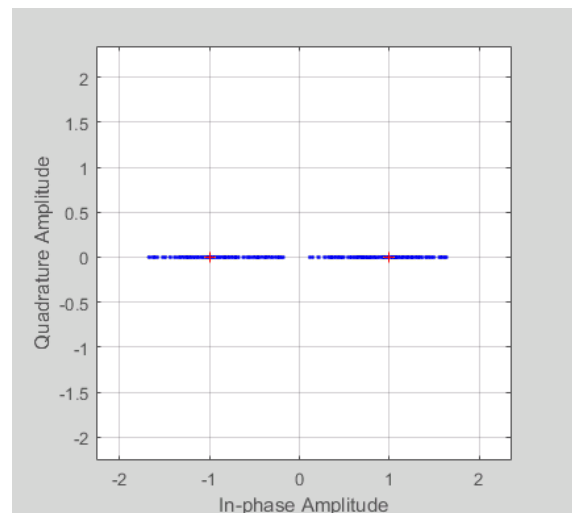


Figure 4.3.1: Eye Diagram of Post Match Filtered Signal with Ideal Channel



(a) Ideal Sampling



(b) Fixed Sampling Error (1 Sample)

Figure 4.3.2: Constellation Diagrams of Matched Filtered Signal with Ideal Channel

The process to synchronize the receiver sampling clock to the transmitter sampling clock

is often referred to as *timing recovery* [7]. There are many different methods to perform timing recovery, some of which are covered in texts like [16] and [20]. In general, these schemes typically involve making an estimate of the timing phase error in the received signal and then applying some correction [16]. The timing recovery block can either be arranged in a feed-forward or a feedback topology [16]. The feedback topology uses the timing error estimate to adjust the parameters of a correction block. The output of this correction is what the timing error estimator uses as input. The feedback topology has the desirable property that it can adapt to time varying changes, provided that these changes are sufficiently slow [16].

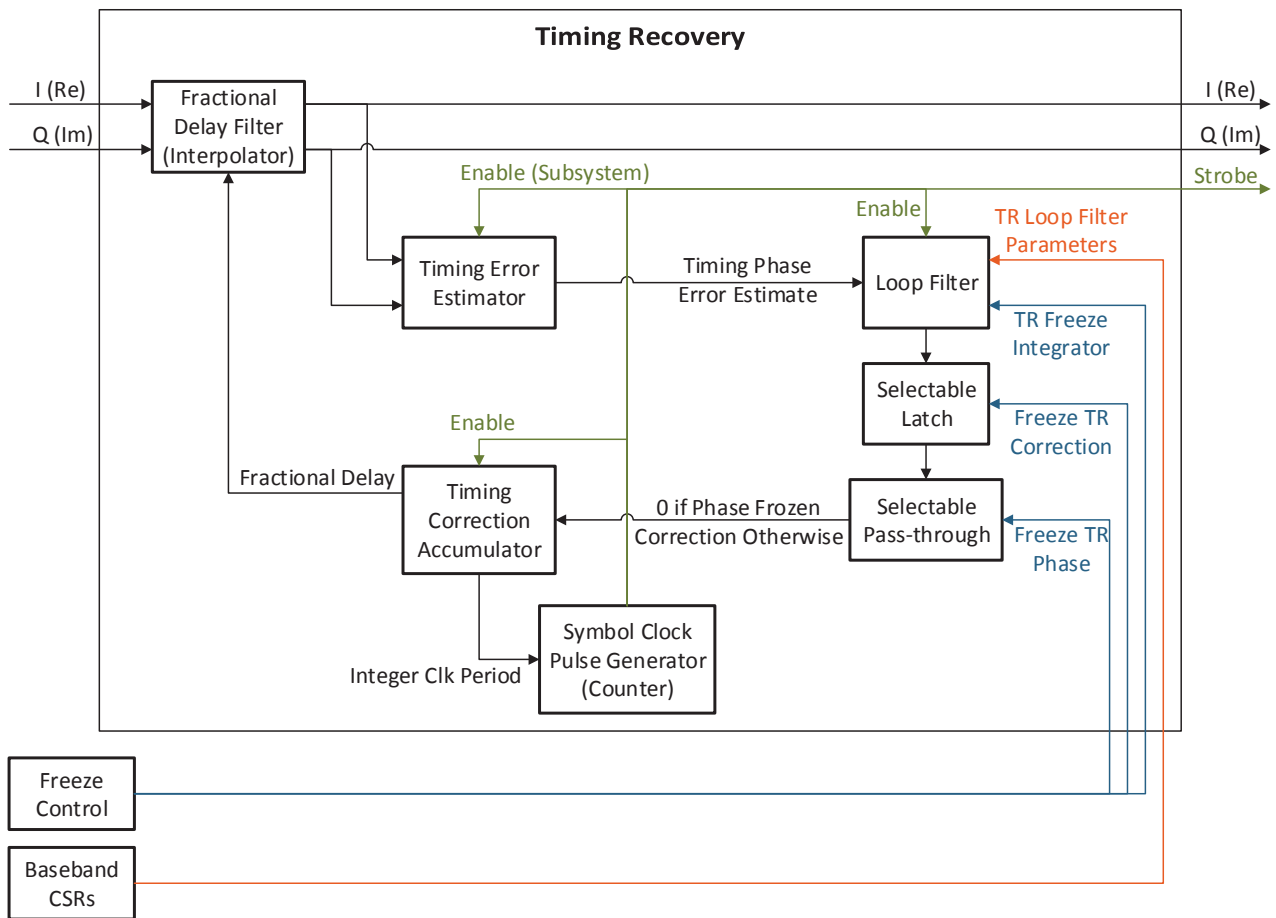


Figure 4.3.3: Timing Recovery Architecture

The timing recovery block for this project is arranged in a feedback topology with the top level architecture shown in Figure 4.3.3. Its general structure is based on a design in [16] which was also adapted as a Simulink example [28]. The implementation for this project relies primarily on [16] which includes a description of the overall control loop along with descriptions of the phase error estimator and fractional delay filter. The signals at the input to the block has already passed through the root raised cosine receive filter (matched filter) and has been normalized by the AGC. This signal is first passed through a fractional

delay filter which interpolates values at a fixed position between sampling points¹. This fractionally delayed signal is fed to the timing error estimator block. A correction value is then calculated by running this error signal through a loop filter. The correction value is then used to derive the proper fractional delay which is passed to the fractional delay filter as well as the clock period for the next symbol. This period is nominally four clock cycles but can grow or shrink to accommodate a frequency offset in the sampling clock or a phase offset greater than one sample. The symbol clock is used as an enable (trigger) signal for several blocks in the timing recovery loop. This is primarily due to the algorithm used for phase detection operating once per symbol period. The trigger signal is used to keep the timing error estimator along with other blocks aligned to the recovered symbol clock. In order to start the timing recovery block, the clock period is initially assumed to be four samples per symbol, the oversampling rate chosen for this design.

4.3.1 Timing Error Estimator

The core of the timing recovery block is the Timing Phase Estimator which is responsible for estimating the phase offset of the sampling clock from the ideal sampling point. There are many potential methods for deriving the timing error estimate with [16] and [20] detailing many potential schemes. The timing estimator used in this project has its basis in Maximum Likelihood (ML) estimation. The details of the ML derivation leverage several concepts from Bayesian statistics and can be quite nuanced. A brief summary of the general ML framework and derivation is provided below, followed by more DSP centric explanation.

Likelihood (ML) Perspective

The objective of an optimal receiver is one which minimizes decoding errors [16]. One way to model this objective is to view the receiver as an estimator of the transmitted data. For a data sequence a and a received signal r , an optimal receiver returns a sequence $\hat{a} = \operatorname{argmax}_a p(a|r)$ [16]. If it is assumed that there is an equal probability across data sequences and the noise is not a function of the data sequence, the maximum a posteriori estimate can be reduced to a maximum likelihood estimate: $\hat{a} = \operatorname{argmax}_a p(r|a)$ [16]. However, this function neglects synchronization parameters such as the timing offset and carrier phase offset. A pure maximum a posteriori estimator eliminates these unwanted parameters via a weighted average [16]. However, the pure optimal receiver is impractical to construct and approximations to it are made in practice. In general, approximations involve the joint estimation of the synchronization parameters (timing and carrier offsets) and the data sequence [16]. One method to perform this joint estimation is to first estimate the synchronization parameters and then to use those estimates to derive the estimate of the transmitted data sequence [16]. This partitioning is significant as it allows synchronization (timing recovery and carrier recovery) to be viewed as a separate operation which precedes decoding. The estimation of symbol timing error, ϵ , and carrier phase offset, θ , for each

¹Interpolating a value at a specific point between two sampling points can be viewed as introducing a fractional delay (less than one sample). Matlab/Simulink use the term “fractional delay filter” while Meyer primarily uses “interpolating filter”. “Fractional delay filter” is used here to emphasize the role of the filter in the design.

possible data sequence a is $\hat{\epsilon}_a, \hat{\theta}_a = \operatorname{argmax}_{\epsilon, \theta} p(r|a, \epsilon, \theta)$ [16]. Note that there is still a dependence on the transmitted data stream in this estimate. Different methods are used to address this dependence with Data Aided (DA) using a known data sequence, Decision Directed (DD) assuming decoded data is correct, and Non-Data Aided (NDA) which removed the data dependence by averaging [16].

An NDA approach was used for the implemented timing error estimator in part due to its flexibility. Meyr in [16] derives a NDA timing estimator that is both non-coherent (does not rely on carrier phase being corrected) and works with both PSK and QAM based modulation schemes. The estimator can be expressed as:

$$\hat{\epsilon} = \operatorname{argmax}_{\epsilon} \sum_{l=-L}^L |z(lT + \epsilon T)|^2 \quad (4.1)$$

where $z(t)$ is the received signal after matched filtering, T is the symbol period, and $[-L, L]$ is the observation window [16].

It is possible to implement this algorithm by discretizing ϵ , calculating the objective function for each value of ϵ , and selecting the ϵ that maximized the objective. However, there is an alternate method to perform the estimation in Equation 4.1.

The method, sometimes called the spectral estimation method, relies on the observation that $|z(lT + \epsilon T)|^2$ is approximately cyclostationary² given a sufficiently large observation window [16]. Meyer takes the Fourier series representation of the process [16]:

$$|z(lT + \epsilon T)|^2 = \sum_{n=-\infty}^{\infty} c_n^{(l)} e^{j(2\pi/T)nT\epsilon} \quad (4.2)$$

In this form, each segment of the waveform has its own set of Fourier coefficients with $C_n^{(l)}$ representing the n th Fourier coefficient for the l th block of the input sequence. Each of these coefficients can be considered a random variable defined by [16]:

$$C_n^{(l)} = \frac{1}{T} \int_0^T |z(lT + \epsilon T)|^2 e^{-j(2\pi/T)nT\epsilon} d(\epsilon T) \quad (4.3)$$

The objective function in Equation 4.1 can be viewed as a scaled time average of the cyclostationary process over the observation interval [16]. When taking the Fourier series expansion of the objective function, the coefficients are averages of the coefficients in Equation 4.2 across the $2L + 1$ blocks of input data [16]³:

$$\sum_{l=-L}^L |z(lT + \epsilon T)|^2 = c_0 + \sum_{n=1}^{\infty} 2\Re [c_n e^{j2\pi n\epsilon}] \quad (4.4a)$$

$$c_n = \sum_{l=-L}^L C_n^{(l)} \quad (4.4b)$$

²A cyclostationary process is a stochastic process with time varying averages that are periodic [7].

³The signal is real valued so c_k and c_{-k} can be collapsed into single terms as shown

It should be noted that an alternate approach to reach this point can be derived from [7] by taking the DFT of the time-averaged autocorrelation of the input signal over the period T .

With proper matched filtering, only c_0 and c_1 have nonzero values [16] yielding a final objective of:

$$\sum_{l=-L}^L |z(lT + \epsilon T)|^2 = c_0 + 2|c_1| \cos(2\pi\epsilon + \arg(c_1)) \quad (4.5)$$

Furthermore, Meyer has shown in [16] that c_0 and $|c_1|$ are independent of ϵ . The objective function Equation 4.5 (equivalently Equation 4.1) is maximized when [16]:

$$\hat{\epsilon} = -\frac{1}{2\pi} \arg(c_1). \quad (4.6)$$

Meyer notes that the bandwidth of $|z(lT + \epsilon T)|^2$ is limited to double the bandwidth of $z(lT + \epsilon T)$ [16]. So long as the sampling rate accommodates this bandwidth, c_1 can be calculated by time averaging DFTs of the input signal:

$$c_1 = \sum_{l=-L}^L \left[\frac{1}{M_s} \sum_{k=0}^{M_s-1} (|z([lM_s + k]T_s)|^2 e^{-j(2\pi/M_s)k}) \right] \quad (4.7)$$

where T is the symbol rate, T_s is the sampling rate, and $M_s = T/T_s$ is the oversampling rate [16]. In the case of 4x oversampling, the complex exponential only takes the values $\{1, -1, j, -j\}$ which leads to an elegantly simple implementation depicted in Figure 4.3.4 [16]. In fact, this simplicity has helped make this estimator one of the more popular NDA timing error estimation techniques [16] [20].

The resulting timing estimator is shown in Figure 4.3.4 with the input signal first passing through a tapped delay line. The delay line is sampled at the pulse of the symbol clock. These samples are then passed through complex magnitude squaring blocks which transform the samples into the domain used by the objective function. $c_1^{(l)}$ is then calculated from these transformed samples using an extremely simple 4 point DFT structure. Time averaging is used to calculate an estimate of c_1 . The final timing error is the result of taking the phase of c_1 using `atan2` and scaling by $1/(2\pi)$.

DSP Perspective

While the ML description above is quite rigorous, it can be difficult to develop an intuition about the algorithm. To give intuition, a less rigorous DSP centric description is provided below.

Let us assume that we have a signal modulated with a PSK or QAM modulation scheme and $z(t)$ is the received signal post matched filtering at time t . One would expect symbols to appear as peaks in $|z(t)|^2$ with dips during symbol transitions, regardless of any carrier phase offset⁴. With a symbol rate of T , it would be expected that, on average, $|z(t)|^2$ would

⁴It will be discussed later that carrier phase offset appears as a static rotation of symbols in the complex plane. This can be viewed as a multiplication by a complex exponential of unity magnitude. This multiplication does not change the magnitude of the symbol and would have no effect on $|z(t)|^2$.

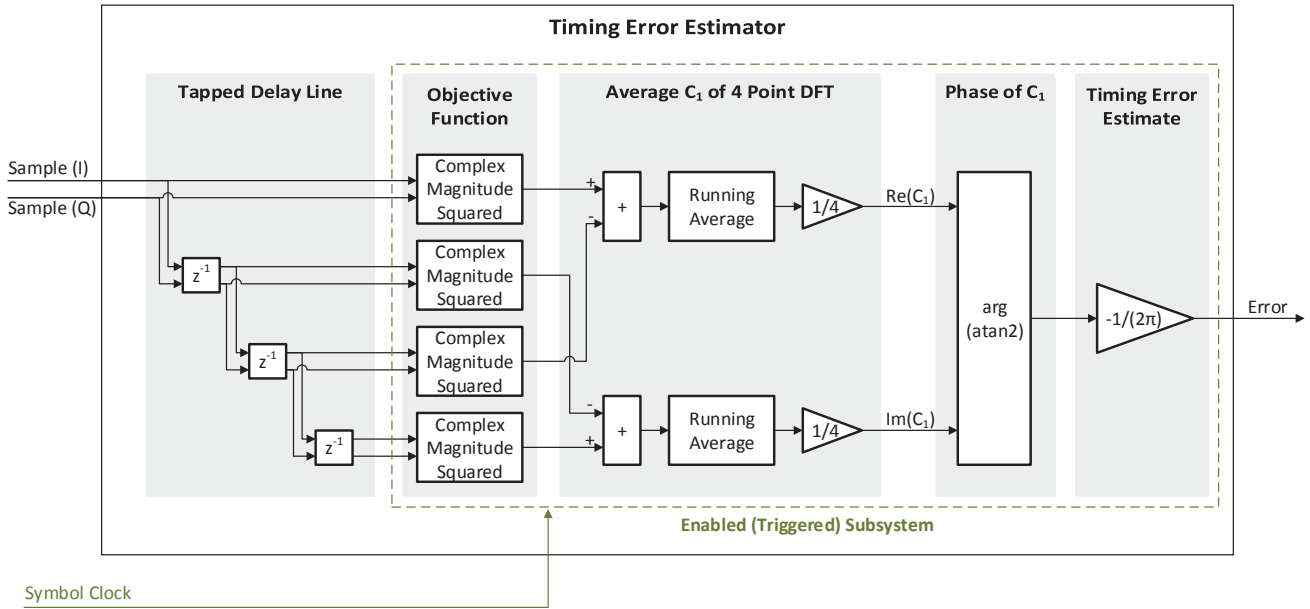


Figure 4.3.4: Timing Error Estimator Architecture

show some periodic behavior with period T due to symbol transitions. Since timing error estimation involves estimating the phase of the transmit symbol clock relative to the receiver symbol clock, the phase of this periodic behavior is of interest.

In a digital system, $|z(t)|^2$ is sampled at a sample rate $1/T_s$. It would then be expected for the sampled signal $|z(k)|^2$ to exhibit, on average, period behavior associated with symbol transitions with a period of $M_s = T/T_s$ samples. The phase and amplitude of this periodic behavior can be captured by calculating c_1 of the M_s point DFT of $|z(k)|^2$. However, it was noted above that this $1/M_s$ periodicity is not guaranteed for every period of M_s samples but is rather true on average. There are cases where no symbol transition takes place due to repeated symbols in the data sequence. An estimate of this $1/M_s$ behavior is therefore attained by averaging c_1 across many M_s sample segments. Taking the complex argument of this c_1 average yields the phase of this $1/M_s$ periodic behavior. Since this periodic behavior is caused by the transition between symbols, estimating the phase of this behavior estimates the phase of the symbol transitions. Estimating the phase of the symbol transitions is equivalent to estimating the phase offset (error) between the receive symbol clock and the transmitter symbol clock.

A key feature of this estimation method is that it does not require knowledge of the carrier phase offset. This allows carrier recovery to proceed timing recovery in the design and has important implications for the hardware implementation which are discussed in section 4.4 and chapter 5.

4.3.2 Loop Filter

The loop filter used in the timing recovery block can be viewed as a PI (proportional, integral) control system or as a single pole, single zero filter. The structure of the filter is shown in Figure 4.3.5 with each coefficient configured by control status registers (CSRs). Configurable saturation is provided to bound the output of the filter and the integrator. There are some cases where it is desirable to freeze portions of the timing recovery loop. The loop filter is capable of freezing the integrator and dropping the proportional component. It is also capable of freezing the current output of the filter. Freezing the output of the loop filter allows the timing recovery block to operate on its best estimate of the symbol clock frequency and phase offset in cases where the timing error estimate becomes unreliable.

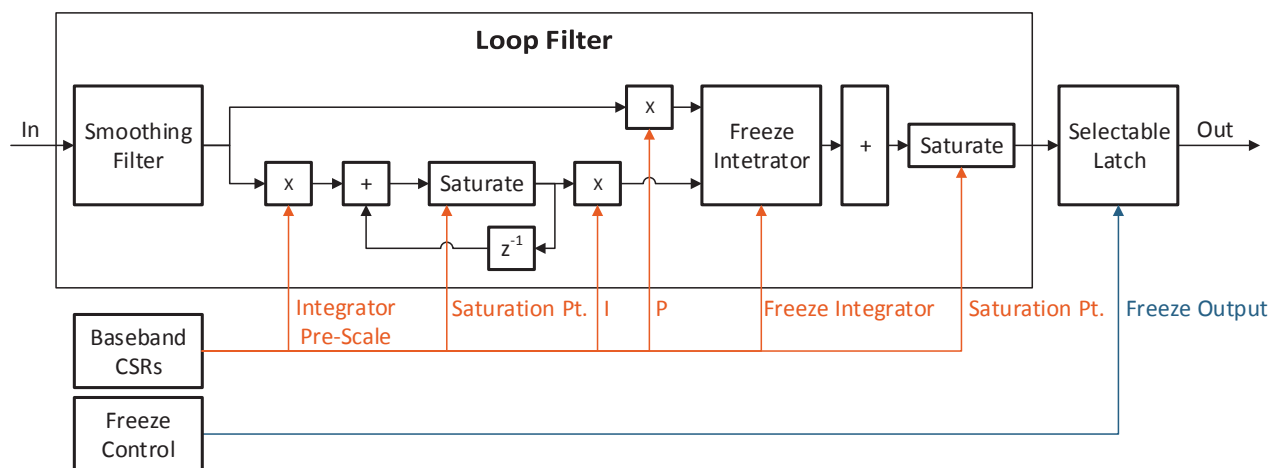


Figure 4.3.5: Loop Filter Architecture

4.3.3 Timing Correction Accumulator & Symbol Clock Generator

The ultimate role of the timing recovery block is to produce a clock pulse at the optimal point for each received symbol. This requires synchronizing the receiver's symbol clock period and phase to match that of the transmitter. The correction value from the loop filter indicates the amount of timing error and should be used to expand or contract the generated symbol clock period to bring the transmitter and receiver's clocks into alignment. One complication with this objective is that the timing recovery block is restricted to running at the sample clock rate. A clock signal must therefore be aligned to samples. This is undesirable as the timing error is likely to include fractions of samples. Restricting clock pulses to be aligned to samples would result in sub-optimal sampling of symbols, potentially leading to decoding errors.

Fortunately, the fractional component of the timing delay can be addressed by the fractional delay filter. This filter interpolates between sample points and can effectively add fractional delay to the signal. However, the fractional delay filter introduces the complexity of having two timing parameters to control: the fractional delay and the integer clock

period. The Timing Correction Accumulator is responsible for calculating these two values in order to maintain optimal sampling of the symbol. It accomplishes this by maintaining the fractional delay between symbol clock cycles. Timing error is added to this fractional delay until it overflows or underflows at which point a sample period is either subtracted or added, respectively, to the next symbol clock period. The fractional delay wraps around the interval $[0, 1)$.

The clock pulse generator is responsible for generating a pulse after the integer clock period specified by the Timing Correction Accumulator has elapsed. On generating the clock pulse, the clock generator resets and is fed a new period by the Timing Correction Accumulator.

4.3.4 Fractional Delay Filter

It is not uncommon for the optimum symbol sampling point to exist at a fixed point between two samples. Ideally, one would like to interpolate between the sample points to attain the optimum symbol sampling point. This interpolation would, in effect, add a fractional delay to the signal. Fortunately such a filter does exist. In Matlab parlance, it is often referred to as a *fractional delay filter* while communications texts such as [16] and [20] prefer to simply refer to it as an *interpolating filter*. The ideal fractional delay filter has a rather simple description as an all pass filter with linear phase. The phase is adjustable by some parameter, often denoted μ [16]. The time domain response of this filter is a sampled sinc function [16] [20]. However, the sinc function is infinite in length making this ideal interpolating filter unreliable in hardware. Fortunately, several methods have been derived to approximate it with alternatives detailed in [16] and [20]. One hardware friendly approximation is the Farrow filter which uses the results of parallel FIR filters to accomplish either a polynomial or Lagrangian interpolation [16]. For this project, a farrow structure with 4 parallel filter, each with order 4, was used. The structure of the filter is detailed in Figure 4.3.6 with the coefficients given in [16].

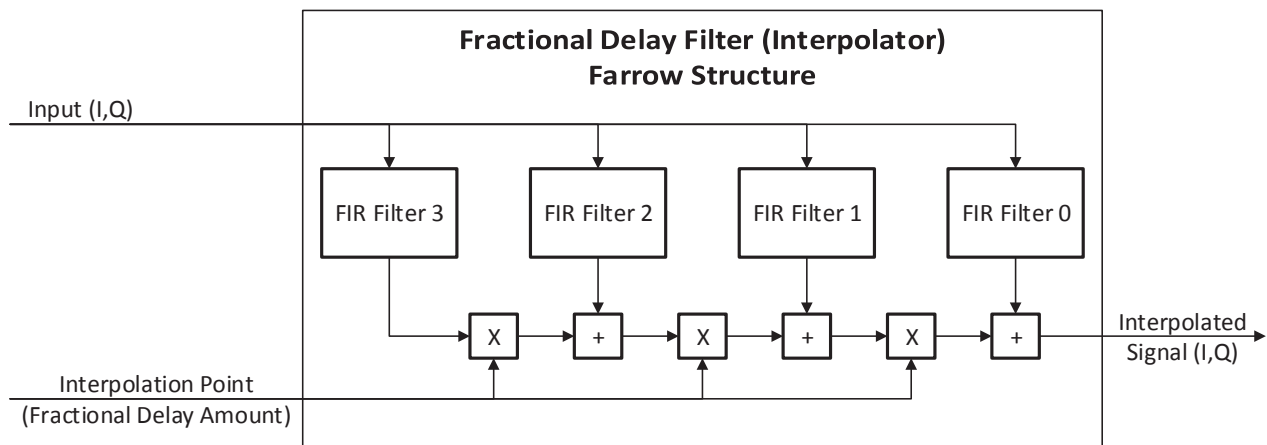


Figure 4.3.6: Fractional Delay Filter Architecture

4.4 Clock Crossing

One of the key functions of the timing recovery block is that it selects the optimal sample for each symbol. Because of this, the timing recovery block can be viewed as performing a decimation from samples to symbols with any block preceding timing recovery operating on samples and any block operating after timing recovery operating on symbols. In steady state operation, the rate of symbols delivered by the timing recovery block should be approximately 1/4 the sample rate since the baseband is over-sampled by a factor of four. This means that blocks preceding timing recovery could actually be clocked at a much slower rate than is required for blocks preceding and including timing recovery. As will be discussed in chapter 5, clocking blocks at slower rate can be extremely beneficial in hardware design. To take advantage of this property, the design is split into a fast domain and slow domain, each operating with different clocks. When partitioning a design like this, the question of how data can be passed across clock boundaries must be addressed. A clock crossing FIFO is a particularly popular method as it can absorb momentary increases in the rate of the producing block so long as the average rate of production is lower than the rate of the consuming block. This is important as the timing recovery block occasionally may need to change the integer clock period of the strobe signal to compensate for large timing phase offsets or small timing frequency offsets. The clock rate in the slow domain is 1/3 the sample rate to ensure that the FIFO drains under normal steady state operation.

The clock crossing FIFO is generated using the FIFO generator provided by Xilinx in the Vivado toolset [29]. Data is directly fed from the timing recovery block to the input of the FIFO. The timing recovery strobe signal, which is used to signal the optimal sample for a given symbol is used as the “push” line for the FIFO. The valid flag at the output of the FIFO is used as a clock enable signal for blocks in the slow domain so that they only operate on valid data from the FIFO. The ready line is set to true in all cases since the receiver does not accept back-pressure from downstream system blocks.

4.5 Carrier Recovery

The RF frontends in the transmit radio and the receive radio use two different local oscillators to upconvert a baseband signal to the desired carrier frequency and to downconvert the received signal back to baseband respectively. The basic structure of the RF frontends is presented in section 2.3 for reference. These local oscillators ideally oscillate at identical frequencies with the phase of the Rx oscillator synchronized to the phase of the received signal. In this ideal case, the two signals leaving the Rx RF frontend match the two transmitted signals fed to the Tx RF frontend. The derivation of this behavior is given in appendix A and appendix B. In practice, however, the local oscillators operate at slightly different frequencies [27]. The distance dependent delay between the Tx radio and the Rx radio also adds to phase differences usually present in the oscillators. The phase difference between the oscillators used in mixing is referred to as *carrier phase offset* while frequency offset between them is referred to as *carrier frequency offset (CFO)*. If the signal at the receiver is viewed as complex with the *I* channel being the real component and the *Q* channel being the imaginary component, carrier phase and frequency errors appear as rotations about the origin

in the complex plane. A phase offset appears as a static rotation while a frequency offset appears as a persistent rotation. The speed of the rotation is proportional to the frequency difference in the carriers of the transmitter and receiver. The underlying justification to why the phase and frequency offsets manifest as rotations in the complex plane are explained in depth in appendix C.

Both forms of rotation are undesirable and must be corrected for coherent modulation schemes [7]⁵. The general goal of the carrier recovery block (sometimes referred to as the carrier phase synchronization block) is to correct for the rotation introduced by the oscillators [20]. There are multiple methods to estimate the phase offset and to apply a correction. One correction method is to adjust the phase of the local oscillator (LO) used in the receiver when downconverting the received signal to baseband [20]. This method was not used since this project utilized RF frontends that do all the downconverting to baseband internally and do not present an easy method for modifying the phase and frequency of the LO. Since the baseband sample rate is close to the symbol rate (2x Nyquist), there was little headroom to digitally upconvert to an intermediate frequency within the frequency range of the DAC. The other method is to apply the correction at the baseband by applying a rotation to the signal in the complex plane to compensate for the rotation introduced by the carrier phase and frequency offsets [20].

There are two general classes of methods to estimate the phase offset at the receiver: decision directed and non-decision directed [7]. Decision directed methods use a decision device (typically the demodulator) to derive an estimate of the phase error based on the received data [7]. Non-decision directed methods treat the received data as random variables and use other mechanisms to derive an estimate of the phase error [7].

In general, decision directed carrier recovery schemes outperform non-decision directed schemes when the error rate in the decision is sufficiently low [7]. In addition, the decision directed recovery schemes as described in [20] and [30] are also more intuitive. The carrier recovery scheme used in this project is a decision directed scheme based on the designs in [20] and [30].

This carrier recovery block applies a complex rotation to baseband symbols to correct for the rotation caused by the carrier phase offset. To derive the correction amount, a phase error estimate is made. This error term is fed into a low bandwidth loop filter that smooths out the estimate. The result of the loop filter is then fed to a phase accumulator which adds the phase error estimate to the current correction angle. The input signal is then rotated by the correction angle by multiplying the input signal by $e^{j\theta_c}$ where θ_c is the correction angle. A high level diagram of the carrier recovery block is shown in Figure 4.5.1.

4.5.1 Phase Error Estimator

The phase error estimator is a crucial component of the carrier recovery block as the correction is derived from an estimate of the phase error. In the decision directed design used in this project, the phase error of a symbol is estimated by measuring the angle between the symbol and the closest constellation point. An example is shown in Figure 4.5.2 where

⁵There are other encoding schemes, such as differential encoding, that are non-coherent. However, the modulation schemes of interest to this project are coherent.

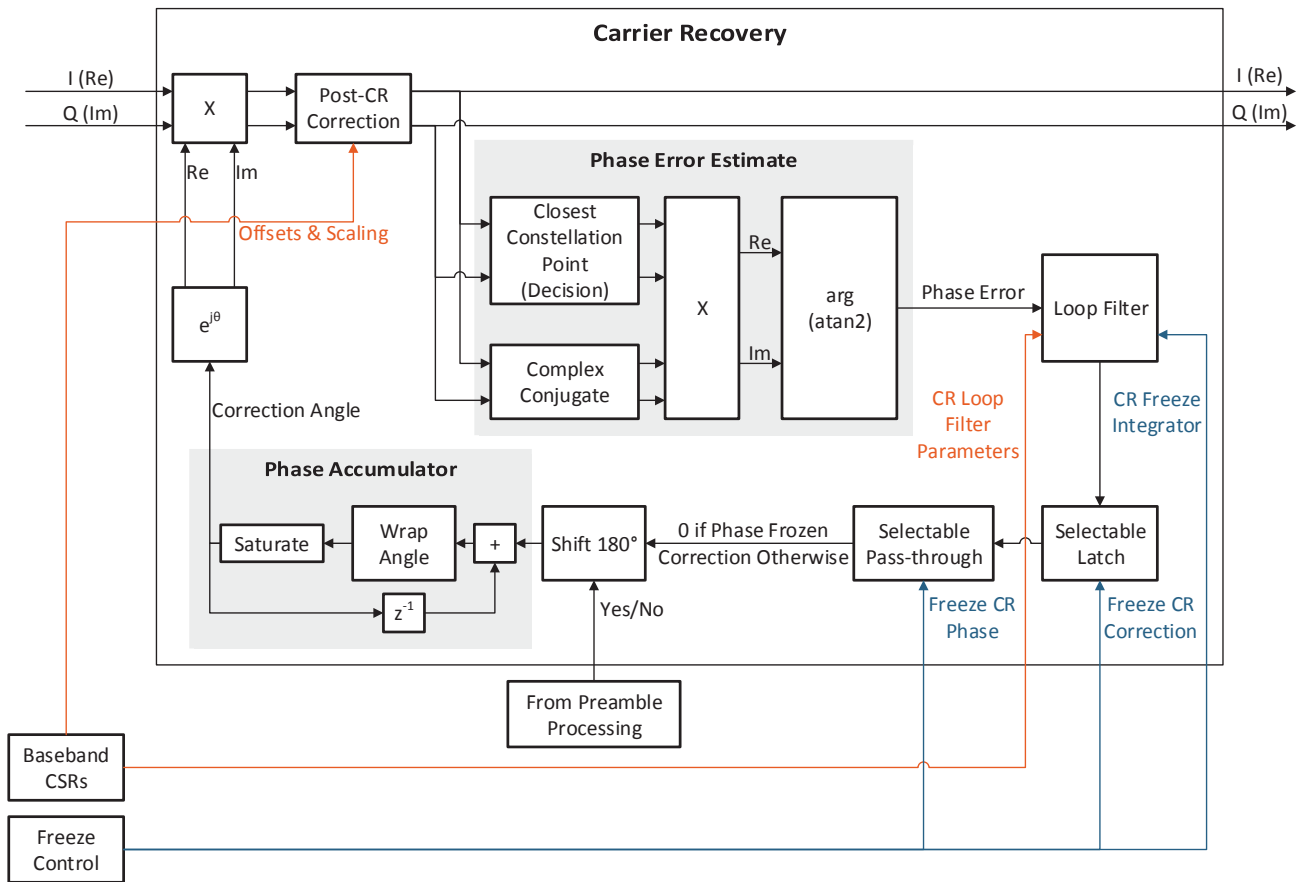


Figure 4.5.1: Carrier Recovery Architecture

the blue dot represents a received symbol and the red 'x' represents the closest constellation point. Conceptually, this can be derived by finding the angle to the closest constellation point and subtracting the angle to the received symbol point. This requires two inverse tangent operations, which is not ideal. Instead, there is an equivalent method used by [30]. A simplified explanation of this method is provided below:

$$\theta_{error} = \theta_p - \theta_a \quad (4.8)$$

$$\theta_a = \tan^{-1} \left(\frac{a_Q}{a_I} \right) \quad (4.9a)$$

$$\theta_p = \tan^{-1} \left(\frac{p_Q}{p_I} \right) \quad (4.9b)$$

$$\theta_{error} = \tan^{-1} \left(\frac{p_Q}{p_I} \right) - \tan^{-1} \left(\frac{a_Q}{a_I} \right) \quad (4.10)$$

Applying the identity $\tan^{-1} z_1 \pm \tan^{-1} z_2 = \tan^{-1} \left(\frac{z_1 \pm z_2}{1 \mp z_1 z_2} \right)$ [31]

$$\theta_{error} = \tan^{-1} \left(\frac{\frac{p_Q}{p_I} - \frac{a_Q}{a_I}}{1 + \frac{a_Q p_Q}{a_I p_I}} \right) \quad (4.11)$$

$$\theta_{error} = \tan^{-1} \left(\frac{\frac{a_I p_Q - p_I a_Q}{a_I p_I}}{\frac{a_I p_I + a_Q p_Q}{a_I p_I}} \right) \quad (4.12)$$

$$\theta_{error} = \tan^{-1} \left(\frac{a_I p_Q - p_I a_Q}{a_I p_I + a_Q p_Q} \right) \quad (4.13)$$

Note that this phase error estimate only requires one inverse tangent operation. This derivation for the left two quadrants is extended to the entire complex plane by replacing the inverse tangent operator with the four quadrant inverse tangent operation (sometimes called `atan2`). Also note that $(p_I + jp_Q)(a_I - ja_Q) = a_I p_I + a_Q p_Q + j(a_I p_Q - p_I a_Q)$. The phase error estimate can be calculated by taking $\arg[(p_I + jp_Q)(a_I - ja_Q)]$ [30], where $\arg(a + jb) = \text{atan2} \left(\frac{b}{a} \right)$. This is the strategy used in the phase estimator implemented for this project.

4.5.2 Loop Filter

The loop filter used in the carrier recovery block has an identical structure to the one used in the timing recovery block which is detailed in subsection 4.3.2. However, each loop filter has its own independent set of configuration parameters and freeze control lines.

4.5.3 Phase Accumulator

The phase accumulator is effectively a second integrator that accumulates the phase error estimates from the loop filter to produce a correction angle. This angle, translated to $e^{j\theta_{correction}}$, is constantly increasing or decreasing if there is a carrier frequency offset. However, there is a limited range of values that the phase accumulator can accommodate. Since $e^{2\pi n + \theta} = e^{\theta}$, wraparound logic is used to keep the correction angle within $(-\pi, \pi)$.

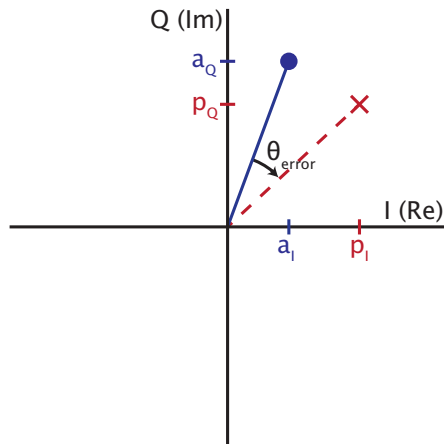


Figure 4.5.2: Example Received Signal vs. Closest Constellation Point

4.5.4 Phase Ambiguity

The carrier recovery block will lock to one of two phases if the modulation scheme is BPSK. There is ambiguity as to which of these two phases (which are 180 deg apart) is the correct phase. Picking the wrong phase would result in an incorrect mapping of received symbols to binary data. This ambiguity is resolved by the Preamble FSM block. Locking to the wrong phase results in the correlation of the preamble having the opposite sign than what is expected. The Preamble FSM can detect this and send a signal back to the carrier recovery block to add a one-time shift of 180 deg to the phase accumulator to correct the error.

4.6 Golay Corrector & Preamble FSM

The signal at the input to the correlator is a sequence of symbols that have already undergone multiple stages of estimation and correction to address various imperfections in the RF environment such as variable gain, symbol clock offsets, and carrier oscillator offsets. Each of these corrections relies on a control loop to estimate the degree of error and to apply an appropriate correction. While these loops perform their job well, unless the error is very insignificant the loops require some time to settle to the appropriate values. This necessitates the transmission of some signal before the payload data to allow the control loops to settle before it is necessary to decode the payload of the frame. This signal is often referred to as the *preamble* and is typically transmitted ahead of frame headers and the frame's actual data payload. In addition to allowing the various control loops time to settle, the preamble is typically used for two other estimations/corrections that are not handled by preceding control loops: frame synchronization and channel estimation/equalization.

While the timing recovery and carrier recovery loops used in this baseband correct for differences between the oscillators in the receiver and transmitter, their designs do not depend on a specific preamble sequence to be present. Both control loops run constantly throughout the frame and, because neither have a preamble dependence, neither estimate the absolute position of the signal within the frame. This is problematic as the exact end of the preamble

and start of the header and payload must be known for proper decoding. Misalignment of the header and payload can cause significant degradation in the bit error rate (BER) of the decoded signal due to the cascading errors it causes. It is therefore necessary to accurately estimate the end of the preamble and start of the header and payload, even in the presence of a noisy channel. The preamble sequence combined with a corresponding correlation can be used to derive this estimate. A properly designed preamble will have an autocorrelation that does not have significant peaks except at offset 0 (perfect alignment). A sufficiently long preamble should have an easily discernible peak at offset 0, even in the presence of noise. The detection of this peak can be used to flag the end of the preamble and the beginning of the header and payload portions of the frame.

The preamble used by this baseband was borrowed from 802.11ad due to its several desirable properties. 802.11ad's preamble is composed of a concatenated set of Golay sequences [32]. Golay sequences are suitable for use in a preamble as they are engineered to have autocorrelations with low false peaks (peaks at shifts other than 0) and low DC components [33]. These properties are particularly useful for frame alignment as large false peaks and DC components make it more difficult to differentiate between false peaks and the peak at shift 0.

Golay sequences can be generated using a recursive algorithm with the Kronecker delta functions used as the base [32]. In fact, Golay sequences are computed in pairs and are sometimes referred to as Golay complementary sequences [32] [33]. The autocorrelations of the two sequences have a special relationship which will be discussed later. The two sequences are typically denoted Ga_n and Gb_n where n is the length of the sequence in symbols [32] [33]. The 802.11ad preamble consists of multiple instances of the Ga_{128} and Gb_{128} sequences arranged in two logical groups referred to as the Short Training Field (STF) and Channel Estimation Field (CEF) [32]. The STF allows the receiver time to synchronize while the CEF field helps facilitate channel estimation. The 802.11ad specification [32] details several different preambles (CPHY, SC, and OFDM), all of which have the same general preamble structure but which have minor differences in the STF and CEF fields. The Control PHY (CPHY) preamble is the most robust preamble with a longer STF field than either the SC or OFDM preambles [32], likely to maximize the probability of proper reception and decoding of important control messages. Robustness was a core objective of the project and thus the 802.11ad CPHY preamble was selected as the baseband preamble. The output of the 128 A and B correlators (Ca , Cb) for the chosen preamble is shown in Figure 4.6.1.

In addition to the low false peak property, the Golay complementary sequences have an additional property that is particularly useful when performing channel estimation. The autocorrelation of the A sequence summed with the autocorrelation of the B sequence exactly cancels the false peaks [33]. A demonstration of this property is shown in Figure 4.6.2. By transmitting a sequence including Ga_{128} and Gb_{128} , the CEF allows the receiver to estimate the impulse response of the channel by summing the A correlation of Ga_{128} with the B correlation of Gb_{128} [33]. It should be noted that this feature of the Golay complementary sequences are not currently used by the baseband as the Channel Estimation and Channel Equalization blocks are not currently implemented. However, these blocks are planned for future revisions of the baseband and using the Golay complementary sequences in the preamble helps facilitate future development of channel estimation and equalization capabilities.

Another important factor when considering the Golay complementary sequences is the

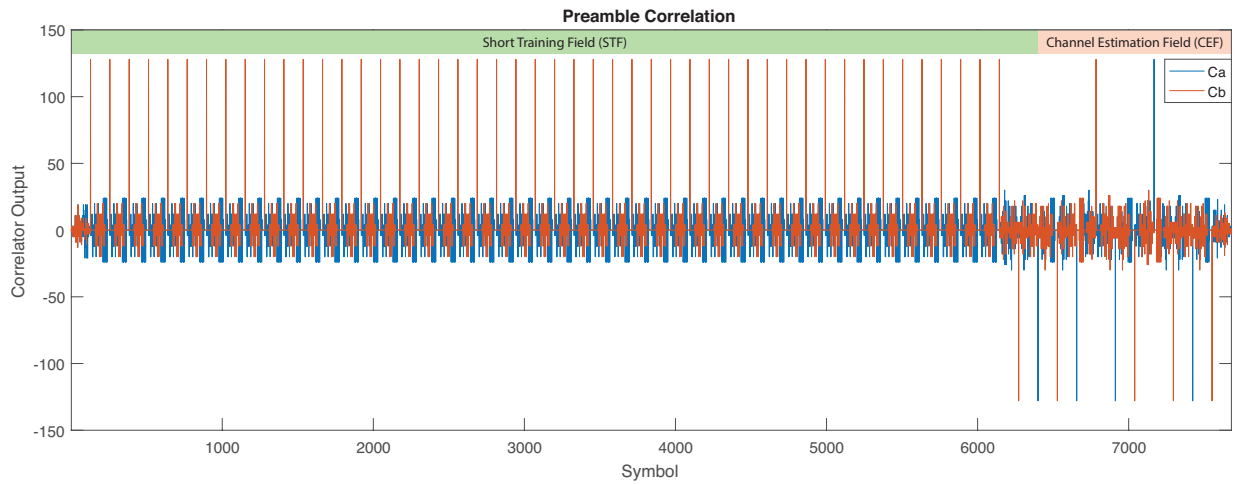


Figure 4.6.1: Correlation of 802.11ad Control PHY Preamble (Output Not Normalized and Referenced to ± 1 Symbols, Maximum=128)

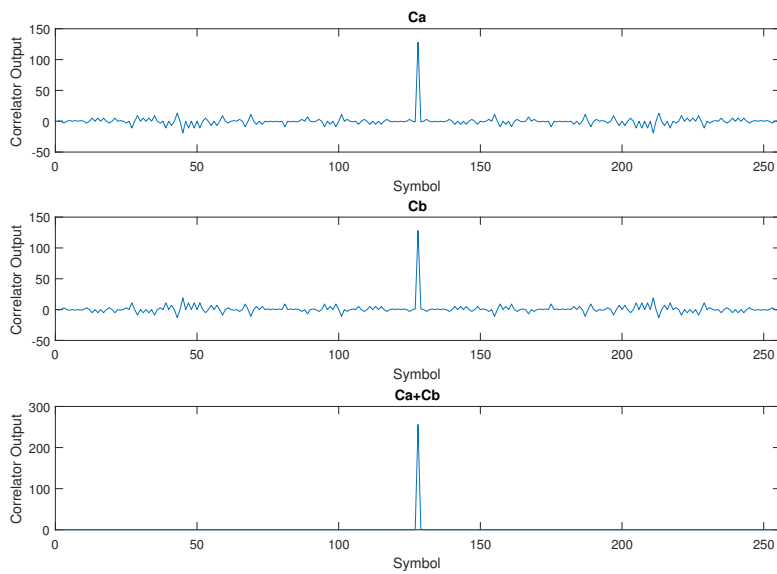


Figure 4.6.2: Correlator False Peak Cancellation Using Golay Complementary Sequences (Output Not Normalized and Referenced to ± 1 Symbols, Maximum=128)

existence of a hardware efficient correlator which is able to compute both Ca and Cb with a fraction of mathematical operators that would normally be required [33]. Cross-correlation can be implemented using an FIR filter due to the close relationship between the two operations. For the 128 symbol long Golay sequences, two 128 deep FIR filters would be required to implement the correlation. These filters could be merged to share the delay line but would still require two 128 symbol arithmetic paths to calculate both the A correlation and B correlation. A more efficient correlator structure is possible due to the mechanism by which the Golay sequences are generated. The recursive algorithm for generating the Golay complementary sequence is shown below [32]:

$$A_0(n) = \delta(n) \quad (4.14a)$$

$$B_0(n) = \delta(n) \quad (4.14b)$$

$$A_k(n) = \begin{cases} W_k A_{k-1}(n) + B_{k-1}(n - D_k) & , 0 \leq n < 2^k \\ 0 & , \text{otherwise} \end{cases} \quad (4.14c)$$

$$B_k(n) = \begin{cases} W_k A_{k-1}(n) - B_{k-1}(n - D_k) & , 0 \leq n < 2^k \\ 0 & , \text{otherwise} \end{cases} \quad (4.14d)$$

$$Ga_{128}(n) = A_7(128 - n), Ga_{64}(n) = A_6(64 - n), Ga_{32}(n) = A_5(32 - n) \quad (4.14e)$$

$$Gb_{128}(n) = B_7(128 - n), Gb_{64}(n) = B_6(64 - n), Gb_{32}(n) = B_5(32 - n) \quad (4.14f)$$

A set of delays D_k and weights W_k are supplied by the standard [32] for generating different sized Golay sequences. For Ga_{128} and Gb_{128} there are seven delay and weight values. All of the weights are either +1 or -1 and the total delay adds up to 127. Agilent Technologies detailed in an application note [33] an efficient correlation implementation that mirrors the recursive algorithm used to generate the Golay sequences. While the correlator requires 127 delays, some of these can be implemented using shift registers as only seven intermediate taps of the delay line are used. A diagram of the correlator design is shown in Figure 4.6.3.

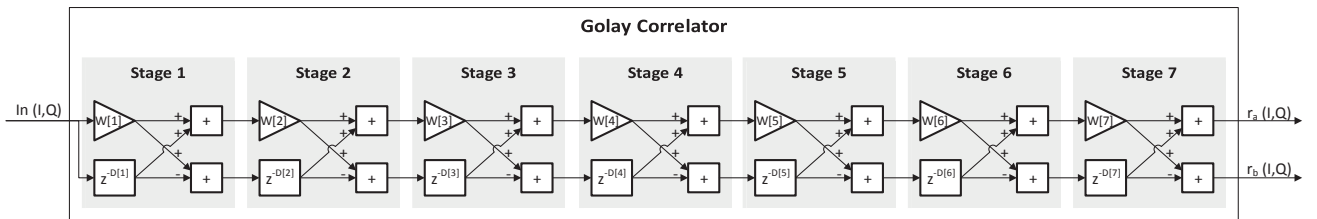


Figure 4.6.3: Efficient Golay Correlator for 128 Symbol Sequence [33]

While the Golay correlator is an important component for frame synchronization, the structure of the preamble requires additional functionality to determine which part of the frame the current symbols are from. The Preamble FSM was created to keep track of the various peaks of the A and B correlators and to compare them to the expected pattern in the preamble. Currently, peak detection is accomplished using a simple threshold operation

on the complex magnitude squared of the correlation. However, more complex schemes are possible. The preamble FSM has built-in tolerances so that small benign errors in the preamble reception can be ignored. If at some point an unexpected peak is observed or an expected peak is not observed, the FSM outputs an error flag and resets. Otherwise, the FSM outputs a flag at the completion of the STF and the completion of the CEF. The STF flag is currently unused but will be useful in later versions of the baseband to trigger channel estimation. The CEF flag is used for frame alignment and signals the end of the preamble and the start of the header and payload.

4.7 BPSK Demodulator & Data FSM

The majority of blocks in the baseband are concerned with estimating and correcting various nonidealities present in physical radio systems as well as real RF channels. The ultimate goal of these estimations and corrections is to produce a sequence of symbols which can be transformed back to data bits with as few errors as possible. With the exception of channel equalization which will be implemented in a future version of the baseband, all of the appropriate corrections have been applied to the signal present at the input of the BPSK demodulator. The role of the demodulator is to take in a stream of symbols and to map these symbols back to the corresponding bit sequence. While the objective of the demodulator is the inverse of the modulator in the transmitter, it does not simply apply the inverse operation. Noise in the channel and imperfect corrections will likely result in a received symbol having some nonzero distance from its corresponding ideal position on the I/Q plane. This introduces some degree of uncertainty with the demodulator selecting the mapping that appears most likely to correspond with a given received symbol. In many cases, this is accomplished by simply finding the ideal constellation point that the received symbol is closest to and returning the bit string mapped to that symbol. For BPSK, the operation of finding the closest ideal constellation point involves simply checking the sign of the symbol's I component. The mapping operation from the closest constellation point to the relevant bit string is trivial, making the BPSK demodulator extremely small and simple to implement.

The final task required of the receive baseband is to signal to the rest of the system when the output of the demodulator is valid. This requires knowing when the data at the output of the demodulator has transitioned from being part of the preamble to being part of the payload. This task is accomplished by the Preamble FSM described in section 4.6. However, this flag only lasts for one cycle before the Preamble FSM resets in preparation for the next frame. A valid flag which remains high for the duration of the frame's payload must be generated for downstream blocks. This task is accomplished by the Data FSM. The Data FSM uses the CEF finished flag from the preamble FSM as the signal to bring the valid flag high. Currently, the frame has a fixed payload size and the Data FSM simply counts up with each demodulated symbol. Once the payload length has been reached, the Data FSM brings the valid flag low and resets.

Chapter 5

FPGA Optimizations and Implementation Results

The target platform for this project is the Xilinx ZC706 which provides a Zynq-7000 Series FPGA. The XC7Z045 FFG900-2 contains Kintex 7 equivalent programmable logic with a hard ARM Cortex A9 processor with backing memory and IO peripherals [12]. FPGA development has its own class of challenges that are distinct from software or ASIC development. To practically realize the design outlined in section 2.2 on an FPGA, several implementation techniques were employed. This section illustrates some of the challenges associated with FPGA development as well as the mitigation techniques used to address them. Post place-and-route results are presented to show the practical viability of the design as well as areas for future design space exploration.

5.1 Timing

Timing closure is one of the most significant problems baseband designers targeting FPGAs are likely to face. For this project, the sample rate is 250 MS/s which requires the portions of the baseband operating at the sample rate to be clocked at a minimum of 250 MHz. The delay through functional units as well as long path delays due to routing can result in timing closure problems. One example particularly relevant to DSP designers centers on the DSP48E1 blocks included in the FPGA fabric which provide specialized multiply and multiply/add functionality. The DSP48E1 blocks require at least three pipeline stages to run at full rate [34] with wide multiplies requiring additional pipeline stages. Mitigation of long path delays due to routing can also be partially alleviated by the insertion of pipeline stages to break long paths into several shorter ones. As a result, pipelining is used extensively throughout the design. In feedforward sections, pipelining simply introduces end-to-end latency. In feedback sections, pipelining introduces undesirable delay in the feedback path. Viewing the feedback path as a component of a filter, the additional delays shift the filter's poles from their ideal locations. Assuming the delay is not too large, additional tuning of the loopback filters can help compensate for but not completely eliminate this effect.

Even with extensive pipelining, closing timing on a complex design can still be challenging. One of the most significant design choices for alleviating timing closure in this project

is the partitioning of the receive baseband into two clock domains. The AGC, Raised Cosine Receive Filter, and Timing Recovery all operate on samples and thus are clocked at the sample rate of 250 MHz. However, Carrier Recovery, Correlators, and FSMs operate on symbols and can therefore operate at a much slower rate. The original single clock domain design required these blocks to be selectively enabled by the timing recovery block at the optimal sampling time. A clock crossing FIFO is currently used to decouple the blocks operating at the fast sample rate from the blocks operating at the slower rate and allowed these two sets of blocks to reside in different clock domains.

The slow section of the baseband could theoretically be clocked at $1/4$ the sample rate of 250 MHz since $4\times$ oversampling is used. However, as was discussed in section 4.3, the sampling clocks in the Tx and Rx radios operate at slightly different frequencies. This could cause the average rate of the strobe signal (signaling the optimal sampling point) from the timing recovery block to potentially exceed $1/4$ the sample rate. If this occurs, the clock crossing FIFO could fill over time and eventually overflow. To avoid this, the slow portion of the baseband is clocked faster than $1/4$ the sample rate. Assuming the frequency difference between the transmitter and receiver sampling clocks is small, this allows the FIFO to drain in steady state operation. For this project, the slow section of the baseband is clocked at $1/3$ the sample rate. The relaxed timing constraints allowed the tool more flexibility in placement and routing which helped the entire design close timing. The relaxation also resulted in significant positive slack on most paths in the slow section of the baseband. This allowed many pipeline stages to be removed in blocks operating at the slower rate, reducing loop delays and resource utilization.

5.2 Area

Another common challenge when targeting FPGAs is the area requirements of a design. Like ASICs, FPGAs have limited resources that can be utilized in a design. FPGAs differ from ASICs in that a fixed number of primitive elements are placed in the FPGA programmable fabric. Moreover, the layout of these primitive elements and the available interconnect between them is fixed by the design of the FPGA itself. This results in different set of challenges when optimizing for the footprint of a design for FPGAs compared to ASICs. The composition of primitive cells used in design alternatives as well as their sensitivity to primitive element placement can be significant factors weighing into design decisions.

One illustration of this type of design decision comes from calculating complex mathematical functions such as arctan. There are many methods to implement the arctan operator, a subset of which are discussed in [35]. Both look-up table and CORDIC based approaches are valid and present tradeoffs between speed, area, and complexity. Look-up table approaches are simple to implement and typically can return results quickly at the cost of storing pre-computed solutions. These solutions can either be stored in the fabric using LUTs or can be stored in dedicated Block RAM (BRAM) cells on the FPGA. For large look-up tables, the use of BRAMs is automatically inferred. CORDIC, by contrast, relies less on precomputed values but requires multiple iterations to arrive at an accurate solution. Because CORDIC requires multiple iterations per solution, it must either be clocked faster than the input data

rate or unrolled to accommodate multiple simultaneous calculations. The clocking option may not be feasible depending on the speed of the data being passed to the CORDIC and unrolling both increases delay and uses additional resources. The trade-off between these alternatives is complex and both solutions are used in practice. Due to simplicity, low delay, and limited use of BRAM blocks throughout the rest of the design, complex mathematical functions such as arctan were implemented using the lookup table approach.

The size and composition of a design can also have an impact on timing closure and compile times. Large blocks can spread logic across a wide area of the FPGA. This can contribute to long path delays which can lead to timing closure challenges. Because long path delays are often constrained by timing closure requirements, large designs can be difficult to place and route. Tight timing requirements between multiple blocks and within a large block can cause congestion in both component placement and routing. This can lead to suboptimal routes being taken and can substantially increase compile times as the synthesis tool may need to explore many possible options before settling on one which fulfills all constraints. It is therefore desirable to reduce the size of the design wherever possible.

Substantial area improvement was attained by re-factoring large averaging filters that are used in various parts of the designs. These were originally implemented as FIR filters since averaging filters can easily be represented as FIR filters and Simulink provides a simple FIR filter block. However, averaging filters are exceptionally simple and do not require the complexity of a full FIR filter implementation. Particularly large averaging filters were replaced with a simpler design used in [36] and shown in Figure 5.2.1. This design performs an N point averaging operation by adding a scaled copy of a sample to an integrator and then subtracting it from the integrator N cycles later. This works so long as the integrator does not overflow or underflow. For a large N , the sample delay could conceivably require a substantial number of register resources. However, the LUT cells include dedicated shift registers which can compactly implement much of the required delay. This replacement resulted in substantial LUT and register savings.

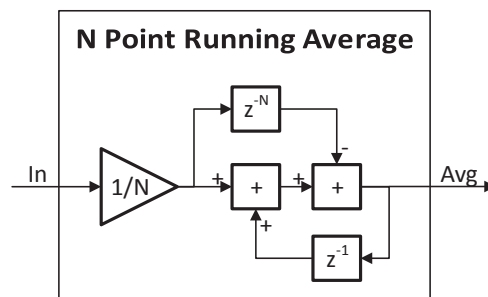


Figure 5.2.1: N Point Running Average Architecture [36]

5.3 Synthesis and Implementation Settings

Even if pipelining and other architectural techniques are employed in the FPGA design, timing closure can still be difficult to achieve. To instantiate a design on an FPGA, the HDL must first be mapped to the particular set of primitive elements present on the target FPGA platform. Typically, a logic synthesis tool such as Xilinx Vivado is used to perform this mapping. There are often multiple ways to map a design to primitive FPGA elements with trade-offs for each method. For example, multiplies can be mapped to a series of LUT and register elements or can be mapped to DSP48E1 blocks. The size of the multiply as well as other factors can influence which technique should be used in practice. In addition to making mapping choices, logic synthesis can also check for other optimization opportunities. Some of these opportunities include propagating constants across module boundaries, ingesting registers into DSP48E1 blocks to improve timing, changing the encoding of FSM state registers, and re-timing modules by moving registers across combinations logic segments to better balance delays. Synthesis tools like Vivado generally provide the user with mechanisms to specify what general strategies they would like the tool to use as well as specific parameters which the tool must respect. Vivado allows the user to specify these directives through synthesis options.

The synthesis options used by this project were based on the “Flow_PerfOptomized_high” strategy and include:

- Low Fanout Limits
- Reduced threshold to implement operations using DSP48E1 blocks using the directive “AreaMultThresholdDSP”
- Retiming Enabled
- Resource Sharing Disabled
- One-hot Encoding for FSMs

Implementation, which includes placement and routing, has its own set of potential options. Directives for implementation can determine how many iterations are attempted for each stage as well as how aggressively alternatives are explored. Enabling physical optimizations (phys_opt_design) post placement and post routing can provide substantial improvements to timing. Examples of phys_opt_design improvements can include logic replication, fanout optimizations, critical path focused optimizations, DSP register optimizations, and BRAM optimizations [37]. The implementation directives used for this project are generally “Explore” or “AggressiveExplore” when available. These directives generally result in higher performance but result in longer compile times [38].

5.4 Floorplanning

To further improve the compile time and timing closure of the design, basic floorplanning was performed. Vivado provides the ability to specify a basic floorplan constraining modules

to specific regions of the FPGA as well as the ability to fix the placement of individual primitive elements. For this design, basic floorplanning proved sufficient. The transmit baseband, receive baseband modules were constrained such that they were close to modules with which they needed to exchange information. The transmit baseband is placed close to the FMC151 and processor subsystem IO pins¹. The fast and slow receive baseband modules are placed next to each other since the fast baseband needs to pass symbols to the slow baseband block to process. The slow baseband is close to the processor subsystem since it needs to pass demodulated data to it. The fast baseband is placed such that there is a clear path from the FMC151 module for samples to pass. The floorplan of the design is shown in Figure 5.4.1.

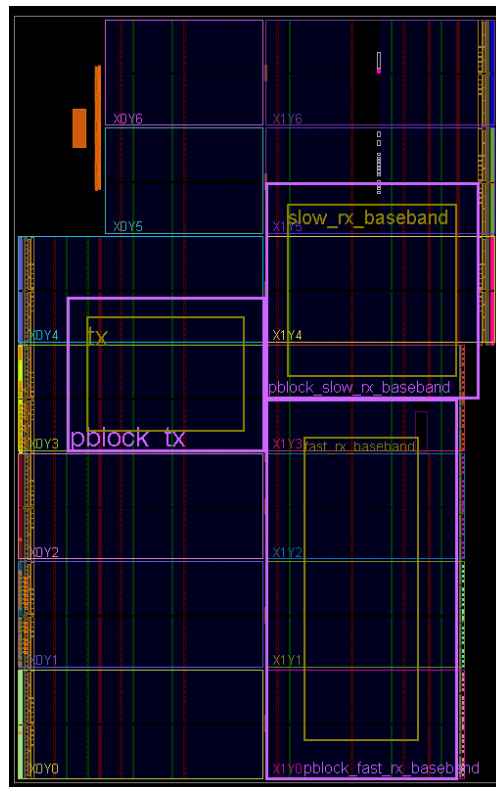


Figure 5.4.1: Synthesis Floorplan

5.5 CSR False Paths

To provide the flexibility to change parameters of the baseband without recompiling the design, Control Status Registers (CSRs) are used. These registers are set by the processor at startup and remain fixed. The baseband modules are also reset after the CSR registers are initialized by the processor, eliminating tight timing requirements on these nets. However, the synthesis tool is not aware by default that the timing on these nets is non-critical and will

¹It was unnecessary to constrain the processor subsystem and FMC151 blocks as they were already constrained by their associated IO.

consider them when performing timing centric optimizations. This can lead to suboptimal results and may cause a design capable of closing timing to fail implementation. The solution is to declare these CSR nets as false paths. By declaring false paths, the synthesis tool is free to ignore timing on these nets and to focus on actual timing critical paths in the design.

5.6 Integrated Logic Analyzer

While the baseband blocks were specified and simulated in Simulink, real world effects typically differ slightly from the simulated effects. There is a need to debug, tune, and evaluate the design directly on the final FPGA platform. However, many of the signals of interest for evaluation such as raw samples, intermediate results, and correction loop rates are superfluous for a general user of the radio. At a basic level, the only signals of concern to the consumer of the baseband is the stream of binary data fed to the Tx baseband and the stream of received binary data coming out of the Rx baseband. To facilitate design evaluation, an Integrated Logic Analyzer (ILA) is used. Vivado allows a designer to mark a subset of signals that he or she wants captured and to automatically generate an ILA core which is attached to JTAG. The Vivado Hardware Manager is then able to communicate with the ILA core to set triggering conditions and to download captured signals. For this design, a large set of signals are of interest for tuning and debugging. A large number of sample points and a wide range of triggering options are also desired to observe different phases of the radio transmission and decoding. As a result, the ILA cores used in this design tend to be large and rely heavily on BRAM blocks. This increase in design complexity can substantially add to timing closure challenges and compile time. In fact, without the ILA cores, compilation does not require extensive use of `phys_opt_design` to meet timing. A comparison of the resource utilization of the design with and without ILA cores is discussed in section 5.7.

5.7 Resource Utilization

The top level resource utilization for the design with and without the ILA core are shown in Table 5.1 and Table 5.2 respectively with the implemented floorplans shown in Figure 5.7.3 and Figure 5.7.4 respectively. While the ILA core requires extensive BRAM resources as well as non-negligible LUT and register resources, it is only needed when debugging, tuning, and evaluating the design. After these tasks are complete and the appropriate tuning parameters are set, the ILA core can safely be removed. Without the ILA core, the complete design easily fits on the ZC706 platform as shown in Table 5.1. A breakdown of the resource utilization of the top modules is shown in Figure 5.7.1. The processor subsystem contains modules that act as AXI peripherals to the processor including the CSR registers and memory mapped to streaming FIFOs. The bulk of the resource usage within the processor subsystem is in the AXI interconnect which handles arbitration and clock domain crossings for AXI transactions. The CSR registers are currently split into several banks which have individual connections to the AXI interconnect. A re-factoring of these CSR modules would likely decrease the size of the AXI interconnect and is a focus of future work.

A breakdown of resource utilization for the baseband alone is shown in Table 5.3 and Figure 5.7.2. The timing recovery block is the largest single user of each set of resources. This is expected as the timing recovery block is arguably the most complex component in the design. Most modules do not use any BRAM resources with usage limited to look-up tables and the clock crossing FIFO.

These utilization results are very encouraging at the design easily fits on the ZC706 platform with plenty of resources available for other modules. The breakdown in Figure 5.7.2 provides insight into potential future opportunities for area optimization in future versions of the baseband.

Name	Slice LUTs	Slice Registers	Block RAM Tile	DSPs
Processor Subsystem	13848	17482	11.5	0
FMC151	3118	3929	10.0	0
Tx Baseband	1147	2913	0.0	42
Rx Baseband	7554	14610	46.0	196
Calibration Assistance	658	1310	0.0	2
Other	1106	1649	0.0	0
ILA	19439	23062	368.5	0
Maximum Available	218600	437200	545	900
Percent Used	21.44%	14.86%	80.00%	26.67%

Table 5.1: Top Level Resource Utilization With ILA Cores

Name	Slice LUTs	Slice Registers	Block RAM Tile	DSPs
Processor Subsystem	13898	17482	11.5	0
FMC151	3123	3929	10.0	0
Tx Baseband	1147	2913	0.0	42
Rx Baseband	7548	14273	45.0	196
Calibration Assistance	658	1310	0.0	2
Other	1074	1609	0.0	0
Maximum Available	218600	437200	545	900
Percent Used	12.57%	9.51%	12.20%	26.67%

Table 5.2: Top Level Resource Utilization Without ILA Cores

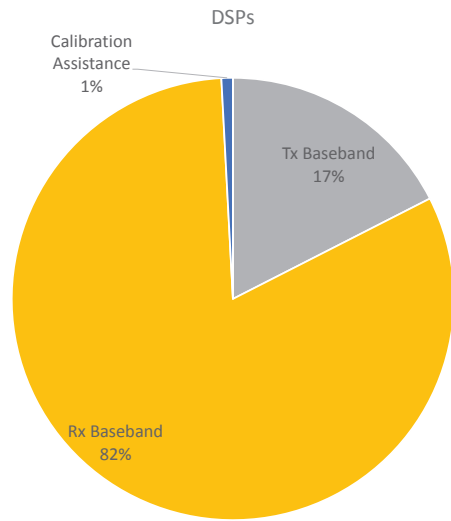
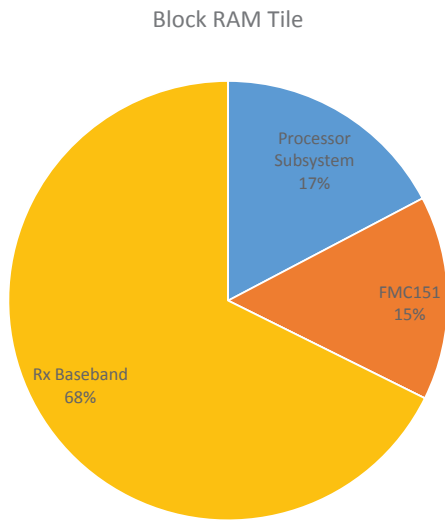
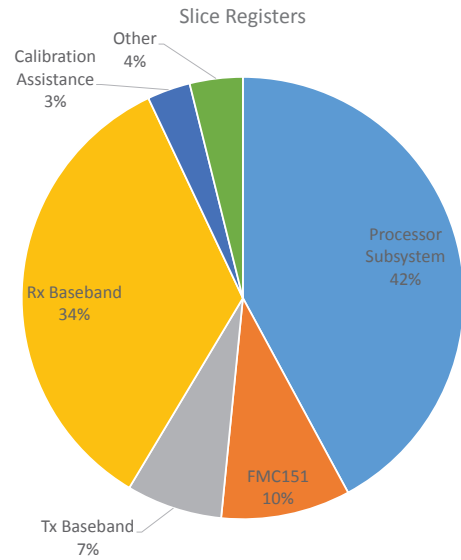
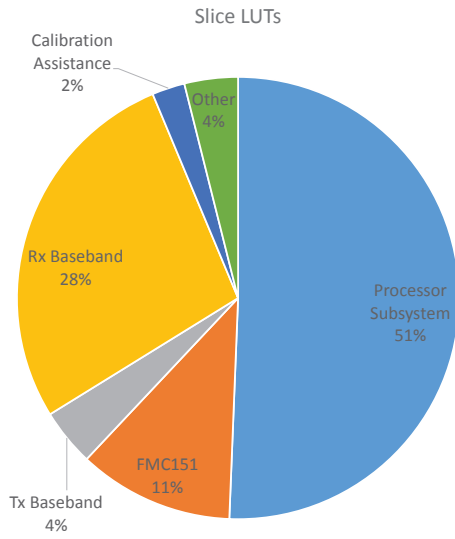


Figure 5.7.1: Top Level Resource Utilization Without ILA Cores

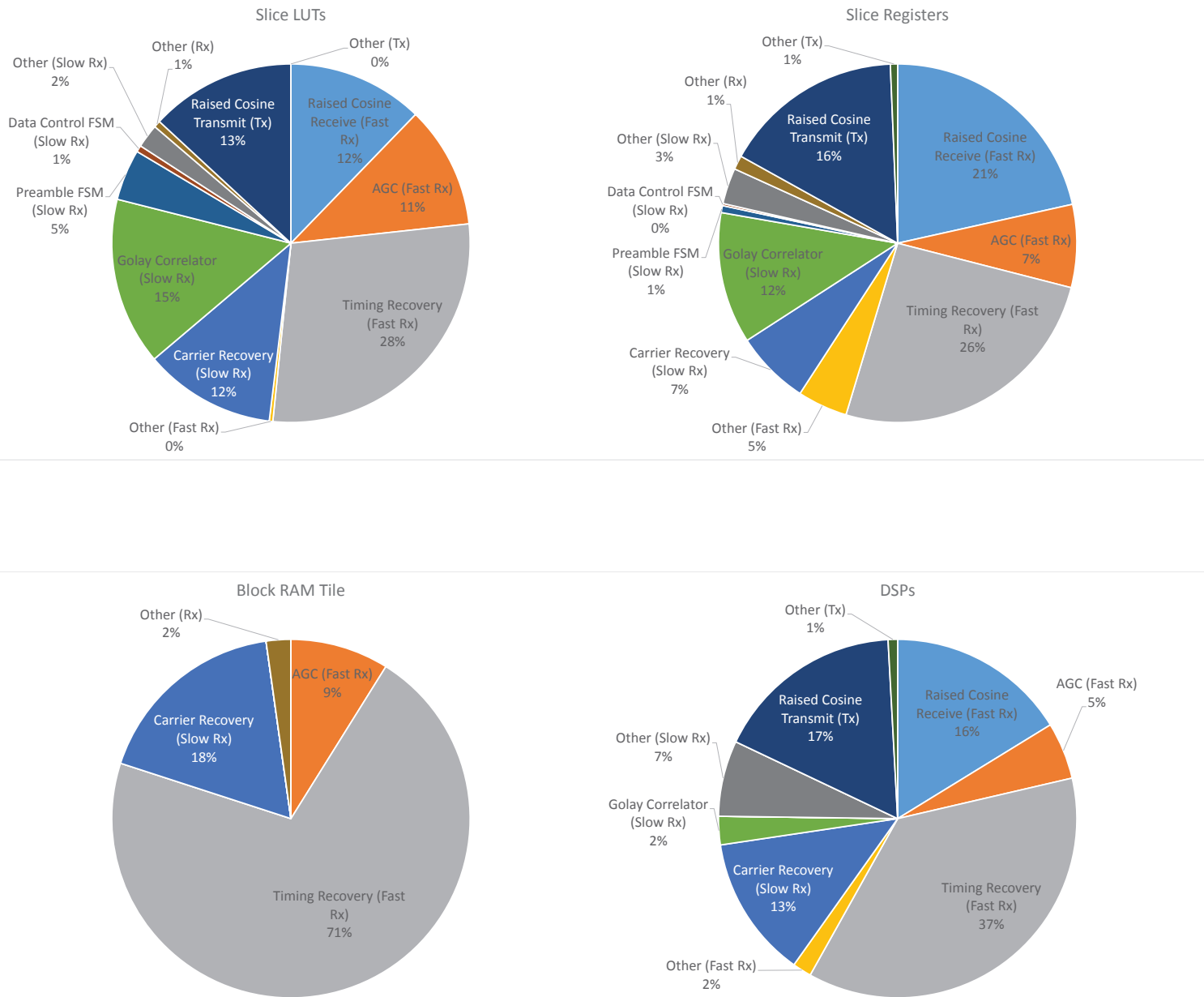


Figure 5.7.2: Baseband Resource Utilization Without ILA Cores

Name	Slice LUTs	Slice Registers	Block RAM Tile	DSPs
Raised Cosine Receive (Fast Rx)	1064	3696	0.0	38
AGC (Fast Rx)	958	1285	4.0	12
Timing Recovery (Fast Rx)	2462	4409	32.0	86
Other (Fast Rx)	28	776	0.0	4
Carrier Recovery (Slow Rx)	1032	1152	8.0	30
Golay Correlator (Slow Rx)	1314	2037	0.0	6
Preamble FSM (Slow Rx)	400	108	0.0	0
Data Control FSM (Slow Rx)	51	33	0.0	0
Other (Slow Rx)	185	557	0.0	16
Other (Rx)	53	216	1.0	0
Raised Cosine Transmit (Tx)	1140	2798	0.0	40
Other (Tx)	0	112	0.0	2

Table 5.3: Baseband Resource Utilization Without ILA Cores

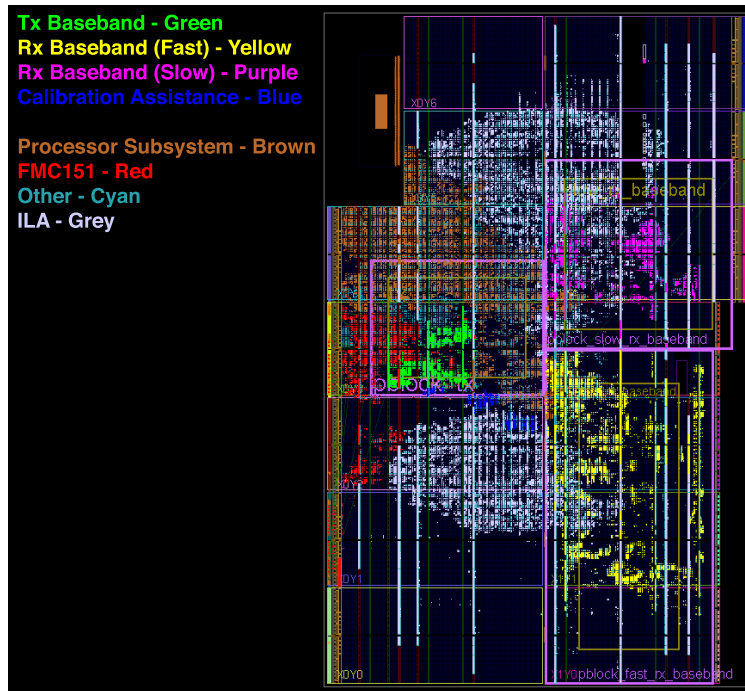


Figure 5.7.3: Post Place-and-Route Floorplan With ILA Cores

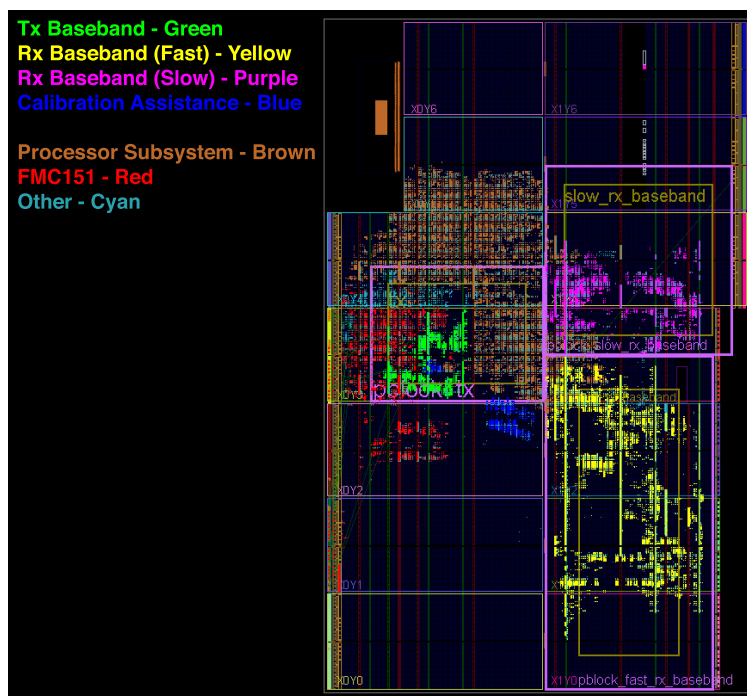


Figure 5.7.4: Post Place-and-Route Floorplan Without ILA Cores

Chapter 6

Testing & Evaluation

6.1 Testing & Evaluation During Development

The first layer of testing and evaluation was facilitated through the use of a Simulink and Matlab based development flow. Because the Tx and Rx baseband were implemented completely in Simulink, they could be connected to object from the Matlab Communications Systems Toolbox to emulate various real world effects such as channels and carrier frequency offset (CFO). The availability of fractional delay filtering also facilitated the simulation of timing frequency offset. Matlab was used to provide stimulus to the Tx baseband and was used to check the consistency of the output decoded by the Rx baseband. A single frame transmission simulation, including Tx baseband, channel impairments, synchronization impairments, and Rx baseband, can be conducted in seconds. This speed allowed simulations to be run throughout the development process to check the functionality of the baseband in several different conditions. The rich set of plotting tools in Simulink also simplified the evaluating each separate control loop during tuning.

One caveat to the Simulink development flow (at least at the time of writing) is that certain HDL centric block parameters (such as internal pipelining) are not represented in the standard Simulink simulation. Simulink HDL coder does provide a method for generating a model which includes these additional delays, referred to as the *code generation model* [39]. Unfortunately, the creation of the code generation model requires executing HDL Coder which runs on the order of minutes for larger designs. Ideally, the code generation model and the standard model would be functionally identical, eliminating the need to create the code generation model. Delays introduced by individual module implementations would be matched across parallel paths using the *delay balancing* option in HDL Coder. However, while automatic delay matching is simple for feedforward systems, it is significantly more difficult (or impossible) in feedback systems. HDL Coder's delay balancing feature does not support blocks with feedback loops [40], limiting its use in this project. Because the baseband design contains several feedback loops, the behavior of the code generation model is different from the original Simulink model.

This limitation leads to a two stage approach to evaluating a change to the design. First, a standard Simulink simulation is used to check basic functionality and performance. Because standard simulations can be completed in seconds, modifications and fixes to the design can

be evaluated quickly. After performance and functionality are acceptable under standard Simulink simulations, HDL Coder is run to create the code generation model. This model is then executed to verify that additional pipeline delays did not change basic functionality and that the feedback control systems respond as expected. Several iterations of feedback control system tuning are often performed at this stage to compensate for additional delays in the feedback paths. Once the performance of the code generation model is deemed acceptable, the next stage of development can begin.

6.2 Performance Characterization - AWGN Channel

While testing the design under several different scenarios is important during development, it can be difficult to determine how well the design is performing. This is due, in part, to radio communications not having a single simple failure mode. Often, performance of a radio system degrades as the SNR or E_b/N_0 of the received signal gets worse. This performance degradation is typically seen as an increase in the Bit Error Rate (BER) and Packet Error Rate. However, under exceptionally poor conditions, there may be other failure modes such as the failure to properly detect the preamble or a loss of synchronization.

One tool to evaluate the performance of a radio system is to compare the BER vs. E_b/N_0 of that system with a given channel model against the performance of another radio system or a theoretical model. Fortunately, there is a theoretical model for the BER in an AWGN channel which is implemented in Matlab [41]. Comparing the performance of the baseband to this theoretical model reveals both if the design performs consistently across different conditions as well as how much improvement is possible.

To conduct this simulation study, multiple simulation runs of the baseband with an AWGN channel were conducted at specific E_b/N_0 values. Because the receiver baseband has several feedback loops, the computational model for the receiver baseband generated by HDL Coder was used. Since the transmit baseband does not include any feedback loops, the standard Simulink model was used for the transmitter. Each simulation had a random carrier phase offset and timing phase offset. Each simulation run also included complete frames with the preamble and a random 4096 bit payload. The bit errors for each run are recorded along with whether or not the preamble was successfully detected. The BER for each E_b/N_0 is calculated by taking the total number of bit errors across all runs for the given E_b/N_0 and dividing by the total number of payload bits sent. The BER for each E_b/N_0 is plotted along with the BER of the theoretical model.

Two different scenarios were simulated with the first containing no carrier or timing frequency offsets. The second scenario included a 100 KHz CFO and a -25 KHz timing offset. The result of 1000 simulation runs for each E_b/N_0 value in each scenario is shown in Figure 6.2.1. The results are highly encouraging as the BER of the simulated runs fall very close to the theoretical curve. There were also no preamble detection failures during any of the simulation runs. It is also highly encouraging that the results for the scenario with CFO and timing frequency offset fell slightly above the results for the scenario with no CFO or timing frequency offset. This shows that the performance of the baseband is only slightly degraded in the presence of a moderate CFO and timing frequency offset. The slight deviation of the curves for higher E_b/N_0 values is likely due to running too few trials for

high E_b/N_0 values. This is a known challenge when characterizing radios as higher E_b/N_0 values begin to have very low BERs. It can take many trials to even see an error or to properly average an error over the appropriate number of successes. In general, high E_b/N_0 values require many simulation runs to acquire good estimates of the BER. The results in Figure 6.2.1 appears to show both cases. For the scenario without CFO, the simulated BER value at 10 dB E_b/N_0 is slightly higher than expected while the simulated BER value at 11 dB E_b/N_0 is 0 and cannot be shown on this logarithmic plot. Running more simulation trials would result in more accurate simulated BER values for these high E_b/N_0 values.

The close proximity of the simulated BER curve to the theoretical curve suggest that, for BPSK under AWGN conditions, there is little opportunity for additional performance improvements. The next logical step is to extend the baseband to support higher order modulation schemes as well as to evaluate the performance of the baseband in other channel models.

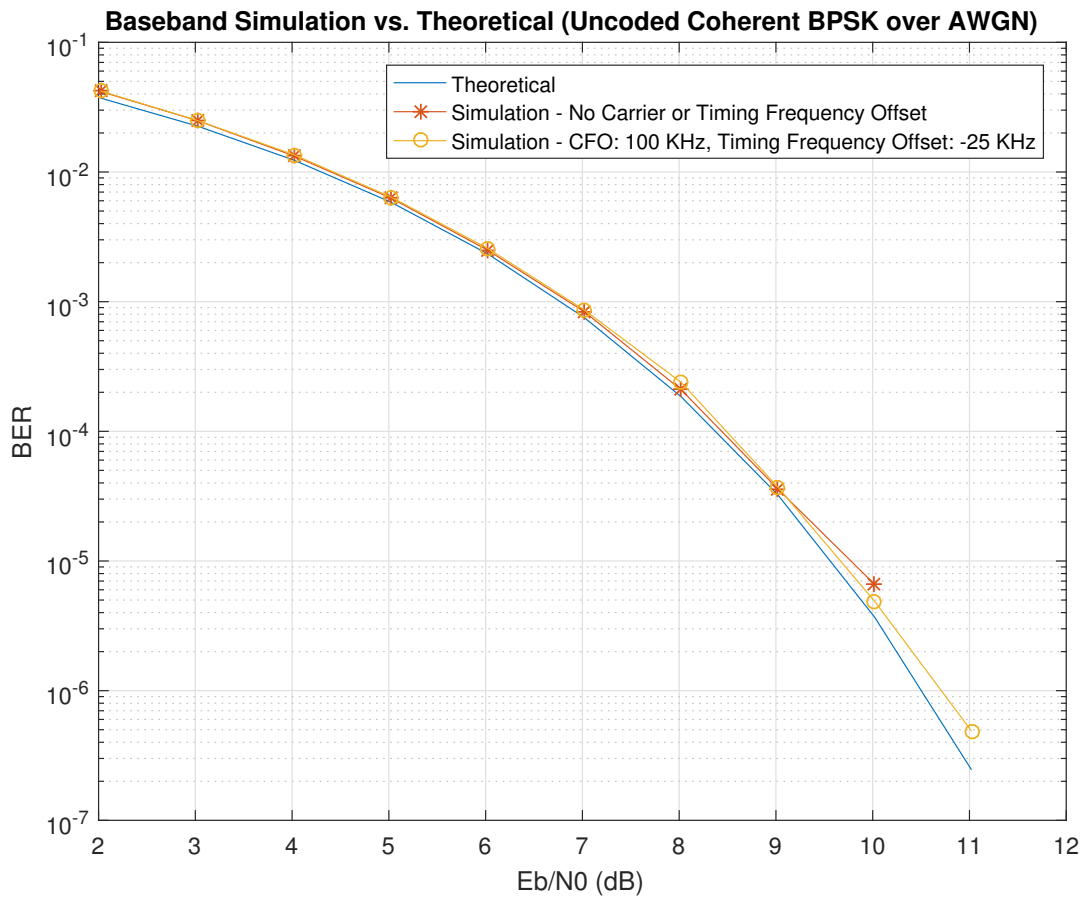


Figure 6.2.1: Simulated vs. Theoretical Baseband Performance in AWGN Channel

6.3 Example Frame Reception with AWGN Channel

Having observed the overall performance of the baseband in the presence of an AWGN channel against a theoretical model, let us take a more detailed look at the operation of individual baseband components using a couple example simulations. Each of the following examples used the code generation model generated by HDL Coder with an E_b/N_0 of 16 dB and a random 4096 bit payload. The packet transmission does not start until 160 μs and ends at 346 μs to simulate times when there is no packet transmission occurring.

6.3.1 Without Carrier or Timing Frequency Offset

The first example illustrates the functionality of the baseband when there is no carrier frequency offset (CFO) or timing frequency offset that must be corrected. The expected behavior of the carrier recovery and timing recovery loops is that they will settle to a stable correction value to compensate for a fixed carrier phase offset and timing phase offset. Plots of the various baseband components are shown in Figures 6.3.2 - 6.3.5 with a snapshot of the received constellation at different points in the baseband shown in Figure 6.3.1.

The AGC behaves as expected with it amplifying the received signal until it reaches the expected power level. For the first 160 μs , there is only noise in the channel and thus the AGC amplifies the signal until the noise power is the expected power of the signal. The AGC adjusts downward at 160 μs when the packet transmission begins to bring the received signal to the expected power level.

Shortly after the AGC begins reacting to the packet transmission, the timing recovery loop begins correcting for the fixed timing phase offset. Each time the curve in Figure 6.3.3 reaches 0 or 1, it wraps around the interval $[0, 1)$ with a cycle added or removed from the integer symbol clock period generated by the timing recovery loop. In this example, there was more than a full cycle of phase offset between the transmitter and receiver symbol clocks. For the given tuning of the timing recovery loop filter, the correction settles slightly after 250 μs .

At the same time the AGC begins correcting for the new receive power level, the carrier recovery loop begins correcting for the carrier phase offset. The correction stabilizes quite quickly to the correct value and remains steady throughout the packet reception.

Because the AGC must lower its correction value in response to the actual packet transmission, there is a period where the received signal has a higher power level than it should. This results in the correlator outputs being larger than expected at the beginning of the frame. One result is that the false peaks at the beginning of the packet transmission are larger than they would normally be. The correlator output returns to the correct level when the AGC has finished adapting as shown in Figure 6.3.5. However, the false peaks did cause the preamble FSM to output a failure flag since the peaks it detected were unexpected. This is denoted by the green peaks early in Figure 6.3.6. However, the preamble FSM resets immediately and correctly handles the preamble once the AGC has settled.

The constellation diagrams shown in Figure 6.3.1 are what is expected for a signal with no CFO or timing frequency offset. The original signal contains intermediate points between symbols and appears like a linear cloud through the origin. The selected samples are the samples at the output of timing recovery. These are the optimal samples for each symbol

and are clearly grouped into two clusters. These clusters are rotated due to the carrier phase offset but are stationary since there is no CFO. Carrier recovery is able to rotate the constellation so that the two clusters are centered on the expected constellation points.

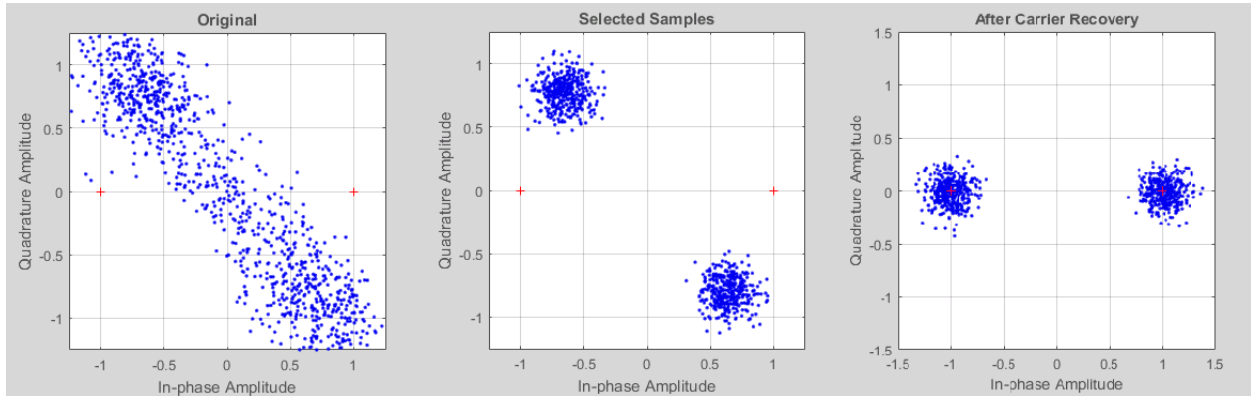


Figure 6.3.1: Received Signal at Different Correction Stages in Receiver Baseband (No Carrier or Timing Frequency Offset)



Figure 6.3.2: AGC Correction During Frame Reception (No Carrier or Timing Frequency Offset)

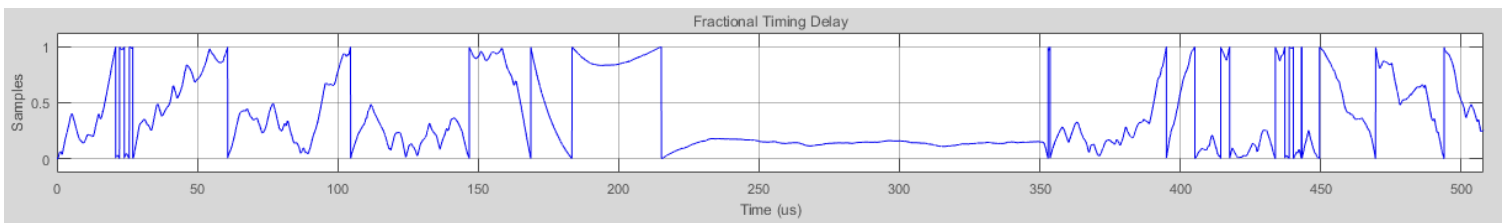


Figure 6.3.3: Timing Recovery Fractional Delay During Frame Reception (No Carrier or Timing Frequency Offset)

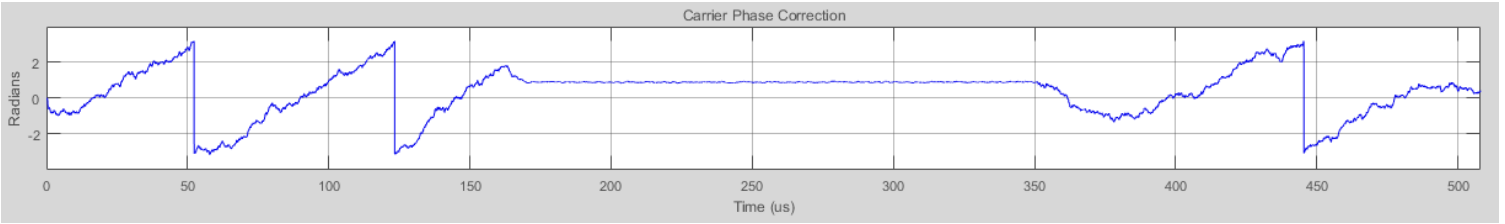


Figure 6.3.4: Carrier Recovery Rotation During Frame Reception (No Carrier or Timing Frequency Offset)

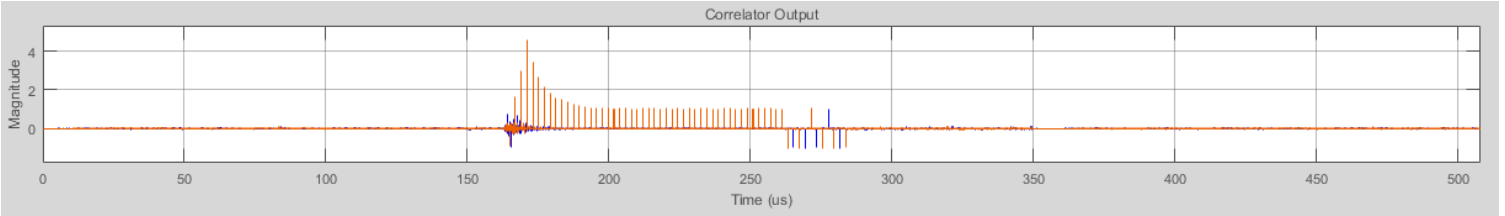


Figure 6.3.5: Correlator Output During Frame Reception (No Carrier or Timing Frequency Offset)

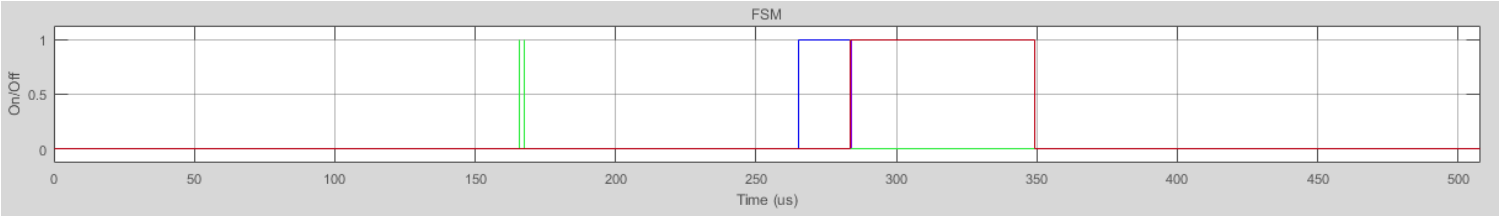


Figure 6.3.6: FSM Output During Frame Reception (No Carrier or Timing Frequency Offset). Red = Data Valid, Blue = STF Done / CEF In Progress, Green = Failure

6.3.2 With Carrier and Timing Frequency Offset

While the radio performed well in conditions where there is no carrier or timing frequency offset, it is expected for both of these effects to be present in physical radio systems. This example shows how the different radio components react with a carrier frequency offset (CFO) of 100 KHz and a timing frequency offset of 25 KHz. Unlike the previous example, it is not expected that the timing recovery loop and carrier recovery loop will settle to specific values. Instead, it is expected that they will settle into a constant rate of correction to compensate for the frequency offsets present in their respective oscillator pairs. As was the case in the previous example, plots of the various baseband components are shown in Figures 6.3.8 - 6.3.11 with a snapshot of the received constellation at different points in the baseband shown in Figure 6.3.7.

The timing recovery loop begins to adapt shortly after the packet transmission begins. However, instead of settling to a single value, the correction takes a sawtooth shape with each wrap around resulting in a sample being dropped. This constantly changing correction is expected given that a frequency offset appears as a constant rate time varying phase offset. For the current tuning of the timing recovery loop filter, the loop settles into steady state operation slightly after $250 \mu s$.

This sawtooth behavior is also present in the carrier recovery loop which settles into steady state operation shortly after the packet transmission begins. An interesting point to note is the discontinuity in Figure 6.3.10 which occurs around $190 \mu s$ and is not caused by the wrap around of the angle correction. This discontinuity is the result of the preamble FSM determining that the carrier recovery loop had locked to the wrong phase. Locking to the opposite phase resulted in the output of the B correlation being the opposite sign than was expected. After rotating 180 deg, the correlator peaks have the correct sign.

The constellation diagrams shown in Figure 6.3.1 are what is expected for a signal with a CFO. Because the constellation is rotating, the original signal is a circular cloud of points. The output of the timing recovery block results in a ring of samples at the expected magnitude of the BPSK constellation points. The carrier recovery block's continuous phase correction results in two clusters of points centered at the expected BPSK constellation points.

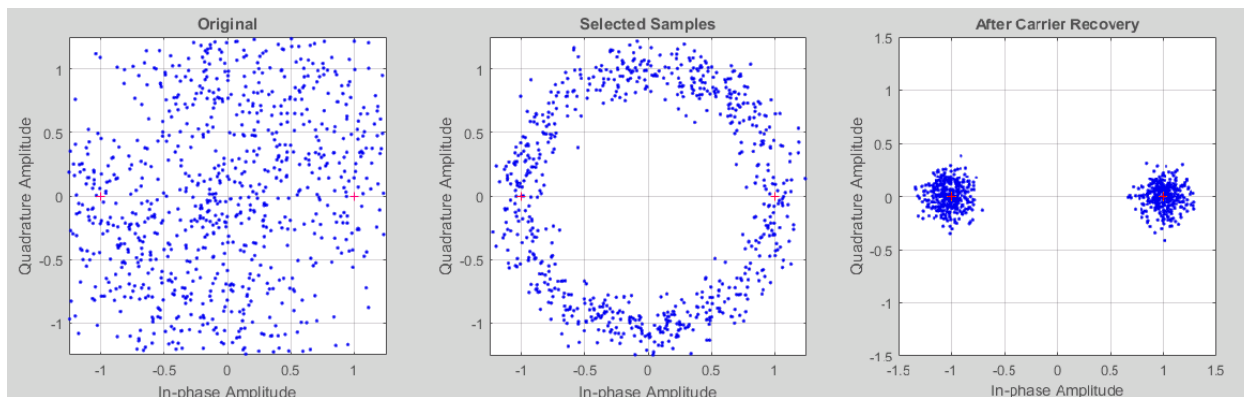


Figure 6.3.7: Received Signal at Different Correction Stages in Receiver Baseband (With Carrier and Timing Frequency Offset)

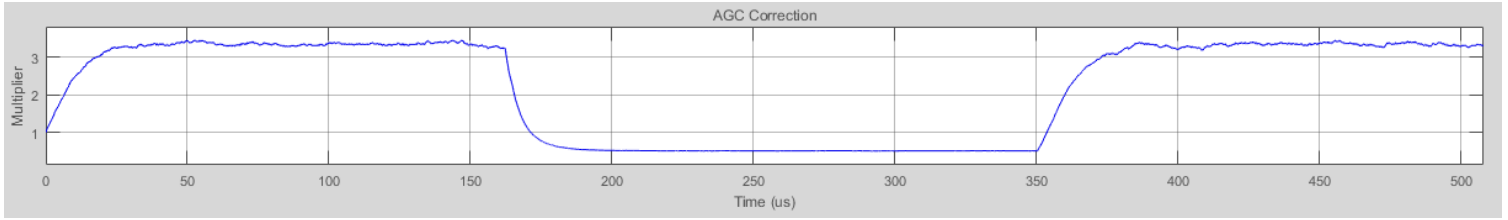


Figure 6.3.8: AGC Correction During Frame Reception (With Carrier and Timing Frequency Offset)

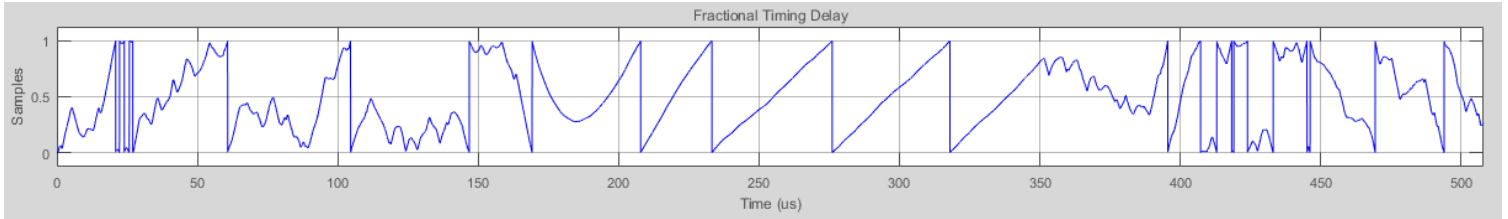


Figure 6.3.9: Timing Recovery Fractional Delay During Frame Reception (With Carrier and Timing Frequency Offset)

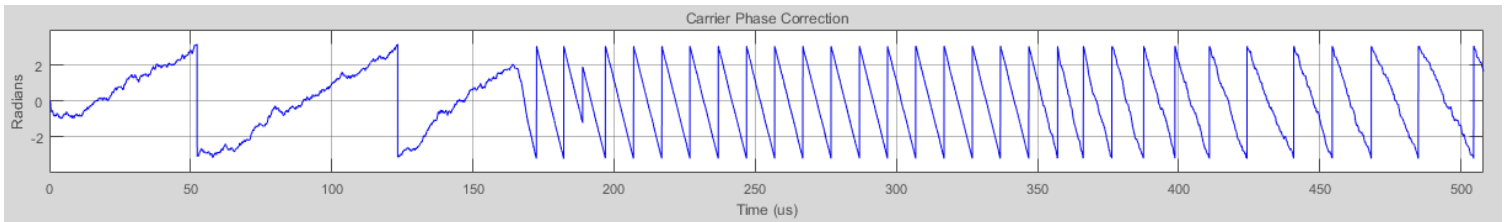


Figure 6.3.10: Carrier Recovery Rotation During Frame Reception (With Carrier and Timing Frequency Offset)

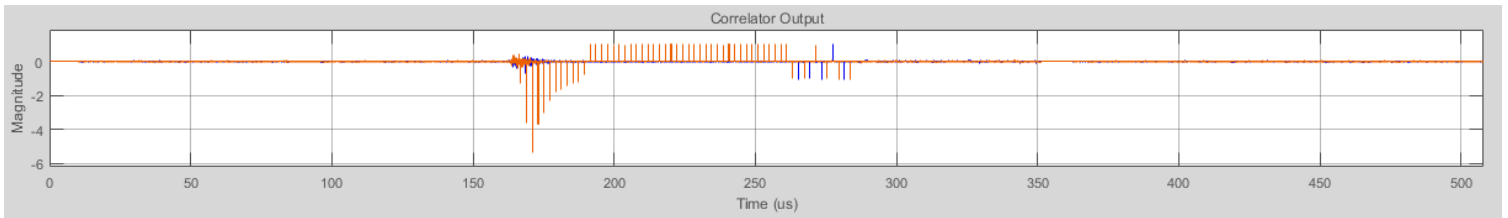


Figure 6.3.11: Correlator Output During Frame Reception (With Carrier and Timing Frequency Offset)

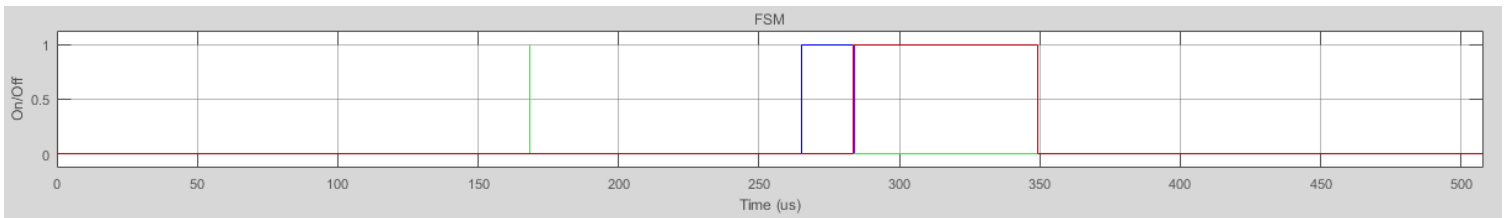


Figure 6.3.12: FSM Output During Frame Reception (With Carrier and Timing Frequency Offset). Red = Data Valid, Blue = STF Done / CEF In Progress, Green = Failure

6.4 Physical Testing

In addition to simulated tests of the baseband, full system tests were conducted to determine if the baseband could operate in real-world conditions. The system was tested in stages with the first stage testing general top level design functionality. This was accomplished by omitting the ADC and DAC components and directly connecting the Tx and Rx basebands within a single FPGA. This ensured that the datapath between the processor and the baseband blocks was operating correctly. Next, the baseband blocks were connected to the ADC and DAC on a single FPGA platform. An external loop from the DAC to the ADC (through SMA cables) was used to check that data was properly flowing from the Tx baseband to the DAC and from the ADC to the Rx baseband. Once this was successful, two separate FPGA boards with independent 4DSP cards were used. Cables were run between the DACs on one board to the ADC on the opposite board. This tested the timing recovery loop as the two FPGA boards did not share common clocks. Finally, a single FPGA with accompanying 4DSP card was connected to a pair of VubIQ frontends. The setup is shown in Figure 6.4.1. One frontend acted at the transmitter while the other acted as the receiver. These two frontends were equipped with WR15 horn antennas and were placed approximately 1 meter apart on a lab bench. After adjusting various settings in the RF frontends such as gain and I/Q imbalance, reliable 60 GHz communication was observed between the two radios. With no obstructions, no errors in reception were detected. However, introducing an obstruction such as a hand introduced bit errors as was expected. These tests were conducted with both independent carrier clocks as well as clocks fed with a common reference.

While an in depth performance analysis of the physical system would be desirable, lab space constrains and the lack of fine control over the PA and LNA in the RF frontends made a characterization such as the one presented in Figure 6.2.1 impractical. Also, characterization of the RF front-ends alone also revealed unexpected and unusual performance characteristics that did not resemble the provided specifications. The unusual operating characteristics of the RF frontends would have likely resulted in unrepresentative overall system metrics had a full system characterization been conducted.

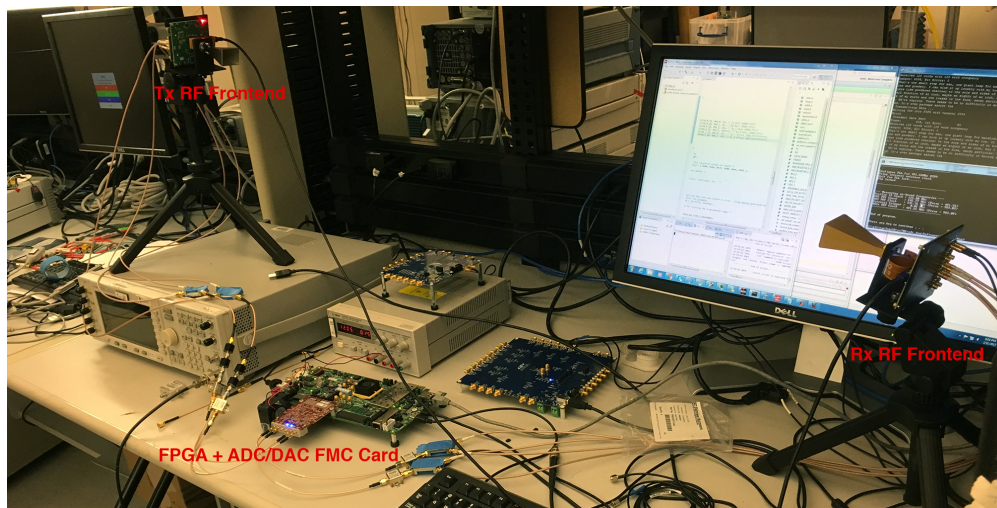


Figure 6.4.1: Test Setup with the FPGA Platform Connected to 60 GHz RF Frontends

Chapter 7

Extending the Baseband to Support Higher Order Modulation Schemes

While the baseband discussed in this report only implements BPSK modulation, it was designed to support higher order modulation schemes with relatively few modifications. The selected timing recovery scheme uses an objective function that works for both M-PSK and M-QAM modulation schemes. Therefore, no change to timing recovery should be necessary to support additional modulation schemes. A minor change to the carrier recovery loop would be required since it is decision directed. The change would involve implementing multiple decision blocks with BPSK remaining the default. The carrier recovery block can be frozen after the CEF using the STF done flag from the preamble FSM. Freezing the channel recovery block would provide time to decode the field a field in the header specifying the modulation scheme used by the frame. Once this field is decoded, the appropriate decision block can be enabled and the carrier recovery loop can be unfrozen.

A prototype of the receiver baseband implementing these changes was completed in half a day with software simulation results shown in figures 7.0.1 - 7.0.7. This simulation used the standard Simulink model (not the code generation model) with a carrier frequency offset of 100 KHz, a symbol timing frequency offset of 25 KHz, and an E_b/N_0 value of 22 dB. This simulation also froze the AGC after the completion of the STF in the preamble. The receive baseband operates as expected with both symbol timing recovery and carrier recovery adapting to the change from BPSK in the preamble and header to 16 QAM in the payload portion of the frame. The timing recovery block continues to operate properly even when the modulation scheme shifts from a PSK scheme to a QAM scheme in the middle of the frame. The brief freezing of the carrier recovery loop allows the decision mechanism to be switched from BPSK to 16QAM without indecent. With the proper 16QAM decision, the carrier recovery loop continues to operate properly during the payload portion of the frame. While some additional work is required before the changes can be integrated into the top level design¹, the speed and ease with which the prototype was implemented illustrates the versatility of the baseband design.

¹The Simulink QAM demodulators do not support power normalization for setting constellation point locations when using HDL Coder and will therefore need to be re-implemented. This should be relatively simple as the QAM demodulator operating in “hard” mode is a simple slicer with a lookup table.

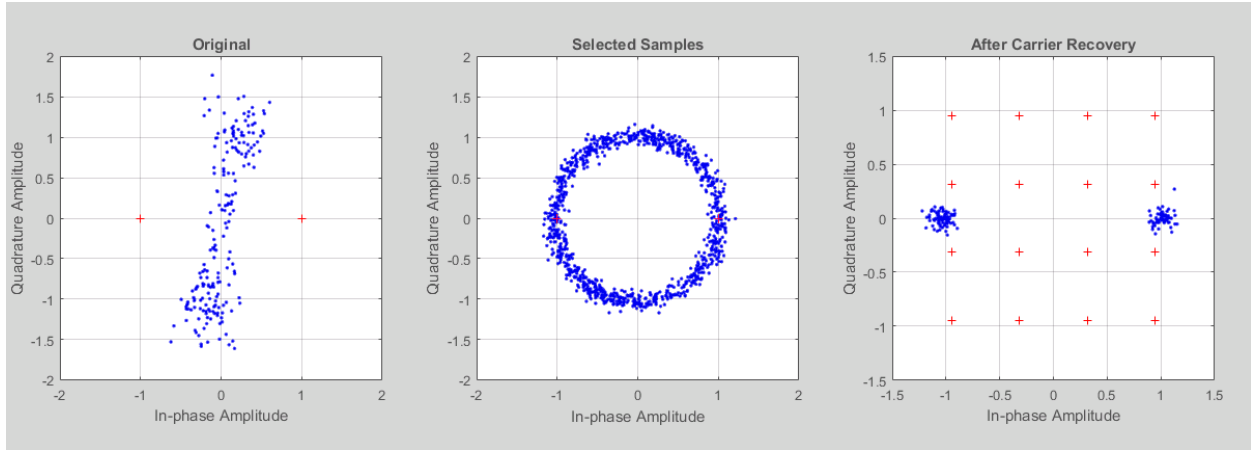


Figure 7.0.1: Received Signal at Different Correction Stages in Receiver Baseband (During the Preamble)

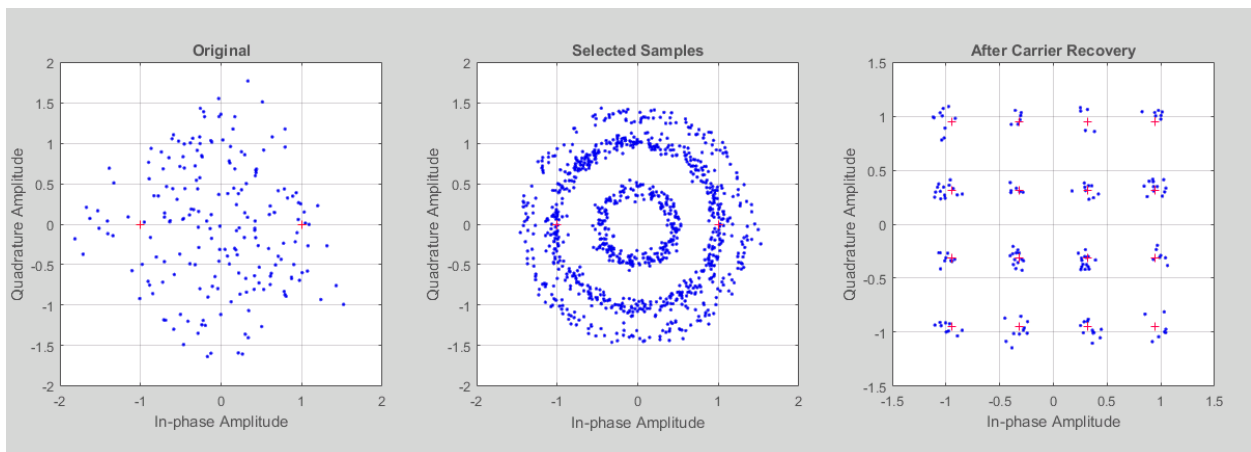


Figure 7.0.2: Received Signal at Different Correction Stages in Receiver Baseband (During the Payload)



Figure 7.0.3: AGC Correction During Frame Reception

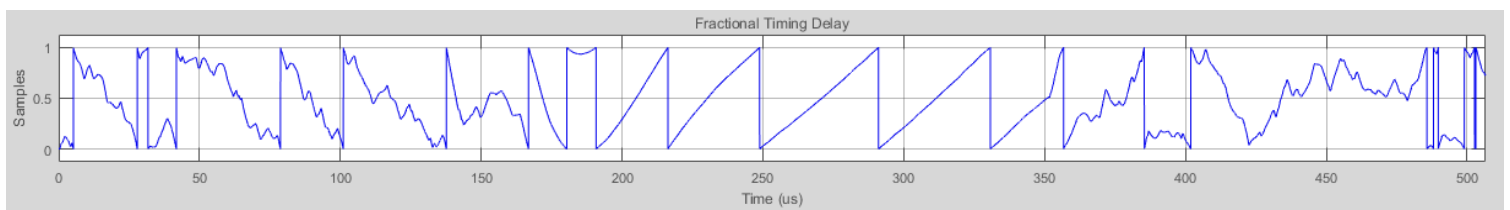


Figure 7.0.4: Timing Recovery Fractional Delay During Frame Reception

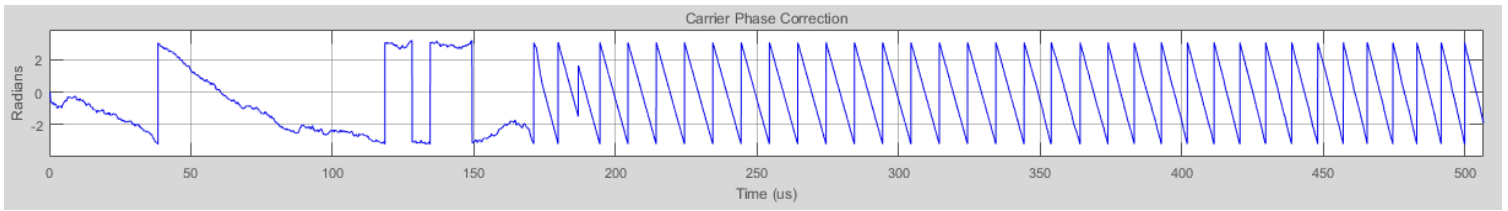


Figure 7.0.5: Carrier Recovery Rotation During Frame Reception

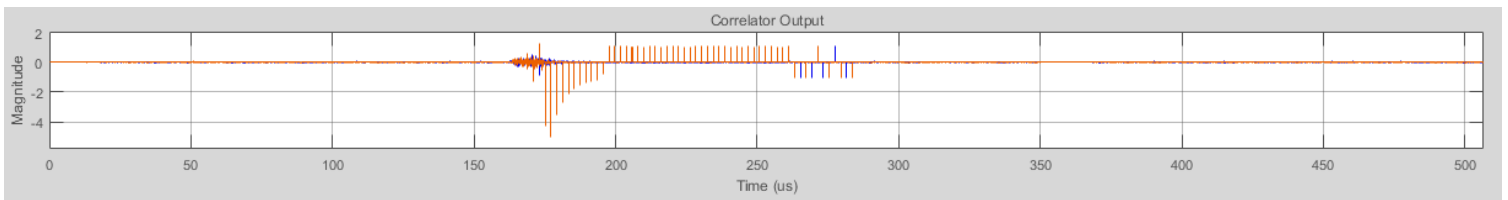


Figure 7.0.6: Correlator Output During Frame Reception

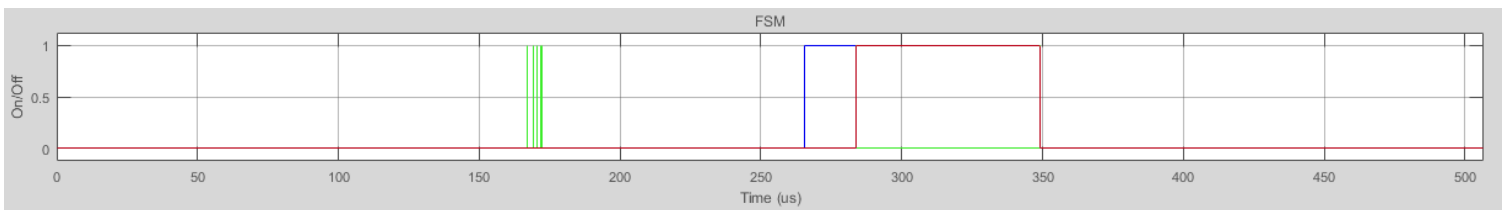


Figure 7.0.7: FSM Output During Frame Reception (Red = Data Valid, Blue = STF Done / CEF In Progress, Green = Failure)

Chapter 8

Future Work

This project has produced an FPGA based platform which is capable of providing successful and reliable BPSK communication with 60 GHz RF frontends. While the FPGA implementation results presented in chapter 5 and the baseband performance results discussed in chapter 6 were very encouraging, there is still room for future development and expansion of the system.

Potential future enhancements include the creation of a channel equalization block as well as a forward error correction (FEC) block. Channel estimation and equalization would improve the performance of the baseband in frequency selective multipath fading channels while forward error correction would help reduce the overall bit error rate (BER) of the system. Additional preamble types with shorter STF fields could also be included to improve latency and reduce the overhead of the preamble when RF conditions are good.

In addition to adding features, future iterations could further optimize portions of the design. Some areas for potential future optimization include re-factoring the CSRs for space and evaluating the use of CORDICs instead of lookup tables. System level optimizations could include the use of high data rate interfaces for data exchange with the host PC such as gigabit Ethernet and the porting of 4DSP's C# initialization code to run completely on the ARM processor. Finally, replacing the VubIQ RF frontends with Sibeam RF frontends could potentially address the unexpected behavior of the VubIQ radios while providing additional degrees of freedom with electronically steerable beam patterns. A FMC card interfacing the Sibeam RF frontends with the FPGA platform has already been designed and manufactured with full integration expected soon.

Chapter 9

Conclusion

The goal of this project was to develop a test and development system to facilitate wireless systems research in mmWave based networks. This system had to be sufficiently general to allow various research projects to utilize it while operating in real time with a relatively high level of performance. These goals led to the development of an FPGA centric platform with the baseband completely resident on the FPGA, communicating directly with attached ADCs and DACs. The baseband developed for the project was designed to be both general and modular. Individual module designs are suitable for the current BPSK modulation but can also be easily adapted to other M-PSK or M-QAM modulation schemes. The implemented baseband occupies a small fraction of the Xilinx ZC706 development platform selected for this project, allowing for the inclusion of additional functionality. It is also capable of operating at a native sample rate of 250 MS/s. The baseband underwent several stages of simulated testing and evaluation using AWGN channels to verify its performance under various conditions and to compare performance against a theoretical model. The results are very encouraging with the simulated performance curves closely matching the theoretical performance curve. System level hardware tests were also performed to verify reliable performance when connected to real 60 GHz RF frontends. Having performed well in the evaluation stages, base level platform functionality is now complete and demonstrated. The platform is now ready for expansion to support various mmWave research experiments and projects.

In addition to providing a working development platform for mmWave research, this project also provided insight into the current state of DSP, FPGA, and radio development. The recent popularity of Software Defined Radio (SDR) may suggest that developing a high-quality radio system is simple. While SDR platforms may accelerate the design process for low bandwidth systems, developing reliable and high-performance radio systems remains a challenge. The radio presented in this report includes corrections for gain offset, timing phase offset, carrier phase offset, and frame level synchronization. There are many ways that each of these effects could be addressed, all of which makes different tradeoffs. Some schemes perform very well but are too resource intensive to be practical for a physical implementation operating on large bandwidth signals. These methods are best suited for applications where data is processed offline or where the signal bandwidth is exceptionally small. Other methods are very efficient to implement but have limited effectiveness or stability. It is incumbent on the designer to evaluate both the performance of a particular solution as well as its

implementation feasibility. Further complicating the design process is the fact that certain methods are dependent on other blocks within the radio baseband. Radio design is therefore not simply the task of picking individual implementations for blocks; it is the process of picking a set of practical implementations that perform well together.

Designing the baseband is only part of the work required to produce a complete radio system. While conceptually simple, passing data between the baseband blocks, data converters, and host system can be non-trivial. Many data converters specify their own method of data transfer which must be implemented in the design. While some vendors provide these interfaces, they may not conform to the specifications required for the design and may need to be modified. Modifying the data converter interface was an unexpected task that occurred during this project. Communication with a host platform can also prove non-trivial, depending on the level of control required. Fortunately, Xilinx provides IP blocks to facilitate the exchange of information between the memory mapped interface that the processor expects and the streaming interfaces that the baseband blocks expect. There are also currently ADCs and DACs on the market that conform to a new interface standard called JESD204B for which Xilinx has created IP cores. This new standard coupled with the Xilinx provided IP blocks could help simplify the system integration task.

Due in part to the large scope of the project, an alternative design methodology was investigated for the baseband implementation with the hope of improving productivity. A common design methodology used by DSP and communications engineers is to model their design in Matlab/Simulink using Mathworks provided blocks and functions. After the modeling is complete, the design is transcribed into HDL with the Matlab simulation serving as a golden reference. While this method allows engineers to experiment in the relatively flexible environment provided by Matlab, it forces them to duplicate their work when transferring the design to HDL. It also introduces the chance of inconsistencies between the Matlab model and the HDL design, especially when the design undergoes changes. One of the value propositions of Simulink HDL Coder is to eliminate the need for a manual Simulink/HDL translation by generating HDL from the Simulink model automatically. While this sounds like the perfect design flow, there are several limitations that can impact the promised benefit. While HDL coder has improved significantly since this project began, there are still lingering challenges including representing pipelines so that their delays are represented in standard Simulink simulations. There are also blocks that either do not support HDL Coder or only support HDL Coder for a small subset of options. This forces the designer to re-implement Mathworks provided blocks so that they can synthesize properly. Despite these issues, the idea behind the flow remains compelling and additional development by Mathworks could help address the flow's current limitations.

Bibliography

- [1] Ericsson AB, “Ericsson Mobility Report: On the Pulse of the Networked Society,” Ericsson AB, Tech. Rep., 2015. [Online]. Available: <https://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf>
- [2] Cisco, “The Zettabyte Era: Trends and Analysis,” Cisco, Tech. Rep., 2016. [Online]. Available: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.pdf>
- [3] Ericsson AB, “5G Radio Access: Capabilities and Technologies,” Ericsson AB, Tech. Rep., 2016. [Online]. Available: <https://www.ericsson.com/res/docs/whitepapers/wp-5g.pdf>
- [4] Qualcomm Technologies, “5G - Vision for the Next Generation of Connectivity,” Qualcomm Technologies, Tech. Rep., 2015. [Online]. Available: <https://www.qualcomm.com/documents/whitepaper-5g-vision-next-generation-connectivity>
- [5] M. Clyburn, J. Rosenworcel, and A. Pai, “Revision of Part 15 of the Commission’s Rules Regarding Operation in the 57-64 GHz Band,” Report No: FCC-13-112. Washington DC: Federal Communications Commission, August 2013. [Online]. Available: https://apps.fcc.gov/edocs_public/attachmatch/FCC-13-112A1.pdf
- [6] T. Wheeler, M. Clyburn, and J. Rosenworcel, “Use of Spectrum Bands Above 24 GHz For Mobile Radio Services,” Report No: FCC-16-89. Washington DC: Federal Communications Commission, July 2016. [Online]. Available: https://apps.fcc.gov/edocs_public/attachmatch/FCC-16-89A1.pdf
- [7] J. G. Proakis, *Digital Communications*, ser. McGraw-Hill series in electrical and computer engineering. Boston : McGraw-Hill, c2001., 2001.
- [8] Y. Huang and K. Boyle, *Antennas : from theory to practice*. Chichester, UK ; John Wiley & Sons Ltd., 2008., 2008. [Online]. Available: <http://dx.doi.org/10.1002/9780470772911>
- [9] V. Rabinovich and N. Alexandrov, *Typical Array Geometries and Basic Beam Steering Methods*. New York, NY: Springer New York, 2013, pp. 23–54. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-1074-4_2
- [10] Aruba Networks, “802.11ac In-Depth,” Aruba Networks, Tech. Rep., 2014. [Online]. Available: http://www.arubanetworks.com/assets/wp/WP_80211acInDepth.pdf

- [11] Xilinx, “ZC706 Evaluation Board for the Zynq-7000 XC7Z045 All Programmable SoC - User Guide,” Xilinx, Tech. Rep., 2016. [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf
- [12] —, “Zynq-7000 All Programmable SoC Overview,” Xilinx, Tech. Rep. v1.10, 2016. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf
- [13] Pasternack, “60 GHz Transmitter (Tx) Waveguide Module Technical Data Sheet: PEM001-MIM,” Pasternack, Tech. Rep., 2016. [Online]. Available: <https://www.pasternack.com/images/ProductPDF/PEM001-MIM.pdf>
- [14] —, “60 GHz Receiver (Rx) Waveguide Module: Technical Data Sheet PEM002-MIM,” Pasternack, Tech. Rep., 2016. [Online]. Available: <https://www.pasternack.com/images/ProductPDF/PEM002-MIM.pdf>
- [15] J. G. Proakis, M. Salehi, and G. Bauch, *Contemporary communication systems using MATLAB*. Stamford, CT : Cengage Learning, c2013., 2013.
- [16] H. Meyr, M. Moeneclaey, and S. A. Fechtel, *Digital Communication Receivers: Synchronization, Channel Estimation, and Signal Processing*. John Wiley & Sons, 1997. [Online]. Available: <http://dx.doi.org/10.1002/0471200573>
- [17] E. N. Cubukcu, “Root Raised Cosine Filters Pulse Shaping in Communication Systems Avionic Systems Analysis,” *AIAA Conference*, 2012. [Online]. Available: <http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120008631.pdf>
- [18] MathWorks, “Raised Cosine Transmit Filter.” [Online]. Available: <https://www.mathworks.com/help/comm/ref/raisedcosinetransmitfilter.html>
- [19] —, “Raised Cosine Receive Filter.” [Online]. Available: <https://www.mathworks.com/help/comm/ref/raisedcosinereceivefilter.html>
- [20] M. Rice, *Digital Communications: a Discrete-Time Approach*. New Jersey: Pearson, 2009.
- [21] IEEE, “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2008*, pp. 1–70, aug 2008.
- [22] Intel, “Altera Changes the Game for Floating Point DSP in FPGAs,” 2014. [Online]. Available: <https://newsroom.intel.com/news-releases/altera-changes-game-floating-point-dsp-fpgas/>
- [23] A. Vishwanath, “Enabling High-Performance Floating-Point Designs,” *Intel Whitepaper*, 2016. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01267-fpgas-enable-high-performance-floating-point.pdf

- [24] MathWorks, “AGC.” [Online]. Available: <https://www.mathworks.com/help/comm/ref/agcblock.html>
- [25] —, “comm.AGC System object.” [Online]. Available: <https://www.mathworks.com/help/comm/ref/comm.agc-class.html>
- [26] J. P. A. Pérez, S. C. Pueyo, and B. C. López, *AGC Fundamentals*. New York, NY: Springer New York, 2011, pp. 13–28. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-0167-4_2
- [27] Semtech, “Improving the Accuracy of a Crystal Oscillator,” Semtech, Tech. Rep., 2009. [Online]. Available: http://www.semtech.com/images/datasheet/xo_precision_std.pdf
- [28] MathWorks, “Timing Recovery Using Fixed-Rate Resampling - MATLAB & Simulink Example.” [Online]. Available: <https://www.mathworks.com/help/comm/examples/timing-recovery-using-fixed-rate-resampling.html>
- [29] Xilinx, “FIFO Generator v12.0 - LogiCORE IP Product Guide,” Xilinx, Tech. Rep., 2015. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v12_0/pg057-fifo-generator.pdf
- [30] C. Dick, F. Harris, and M. Rice, “FPGA Implementation of Carrier Synchronization for QAM Receivers,” *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 36, no. 1, pp. 57–71, 2004. [Online]. Available: <http://dx.doi.org/10.1023/B:VLSI.0000008070.30837.e1>
- [31] D. Zwillinger, *Standard Mathematical Tables and Formulae*, 32nd ed. Boca Raton: CRC Press, 2012.
- [32] IEEE, “ISO/IEC/IEEE International Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band (adoption of IEEE Std 802.11ad-2012),” pp. 1–634, 2014. [Online]. Available: <http://dx.doi.org/10.1109/IEEESTD.2014.6774849>
- [33] Agilent Technologies, “Wireless LAN at 60 GHz - IEEE 802.11ad Explained: Application Note,” Agilent Technologies, Tech. Rep., 2013. [Online]. Available: <http://cp.literature.agilent.com/litweb/pdf/5990-9697EN.pdf>
- [34] Xilinx, “7 Series DSP48E1 Slice User Guide,” Xilinx, Tech. Rep., 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
- [35] MathWorks, “Calculate Fixed-Point Arctangent - MATLAB & Simulink Example.” [Online]. Available: <https://www.mathworks.com/help/fixpoint/examples/calculate-fixed-point-arctangent.html>

- [36] P. D. Francesco, S. McGettrick, U. K. Anyanwu, J. C. O'Sullivan, A. B. MacKenzie, and L. A. DaSilva, "A Split MAC Approach for SDR Platforms," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 912–924, apr 2015. [Online]. Available: <http://dx.doi.org/10.1109/TC.2014.2308197>
- [37] Xilinx, "Vivado Design Suite Tcl Command Reference Guide," Xilinx, Tech. Rep., 2016. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016.4/ug835-vivado-tcl-commands.pdf#nameddest=xphys_opt_design
- [38] G. Daughtry, "Top 5 Timing Closure Techniques," Xilinx, Tech. Rep. [Online]. Available: https://www.xilinx.com/publications/prod_mktg/club_vivado/presentation-2015/paris/Xilinx-TimingClosure.pdf
- [39] MathWorks, "Delay Balancing and Validation Model Workflow In HDL Coder - MATLAB & Simulink Example." [Online]. Available: <https://www.mathworks.com/help/hdlcoder/examples/delay-balancing-and-validation-model-workflow-in-hdl-coder.html>
- [40] —, "Control the Scope of Delay Balancing - MATLAB & Simulink Example." [Online]. Available: <https://www.mathworks.com/help/hdlcoder/examples/control-the-scope-of-delay-balancing.html>
- [41] —, "Bit error rate (BER) for uncoded AWGN channels - MATLAB berawgn." [Online]. Available: <https://www.mathworks.com/help/comm/ref/berawgn.html>
- [42] E. W. Weisstein, "Complex Number," Wolfram Research, Tech. Rep. [Online]. Available: <http://mathworld.wolfram.com/ComplexNumber.html>
- [43] —, "Complex Argument," Wolfram Research, Tech. Rep. [Online]. Available: <http://mathworld.wolfram.com/ComplexArgument.html>
- [44] K. Gheen, "Phase Noise Measurement Methods and Techniques," Agilent Technologies, Tech. Rep., 2012. [Online]. Available: http://www.keysight.com/upload/cmc_upload/All/PhaseNoise_webcast_19Jul12.pdf
- [45] R. Cerda, "Sources of Phase Noise and Jitter in Oscillators," *Microwave Product Digest*, no. March, 2006. [Online]. Available: <http://www.crystek.com/documents/appnotes/SourcesOfPhaseNoiseAndJitterInOscillators.pdf>

Appendix A

QAM Modulation

One key feature of many RF frontends, including the VubIQ frontends, is the inclusion of an analog Quadrature Amplitude Modulation (QAM) modulator and demodulator. The QAM modulator provides a very useful abstraction to baseband radio designers that allows two analog signal streams to be modulated together into a signal transmitted signal which can be demodulated into two separate analog signals at the receiver. To understand these benefits, as well as the challenges that QAM can introduce, it is necessary to first understand how QAM modulators and demodulator operate.

QAM modulators take in two baseband signals which are commonly referred to as the in-phase, I , signal and the quadrature, Q signal. These two signals are scaled and mixed with two sinusoidal oscillators that share the same frequency but are 90 deg out of phase. If $I(t)$ and $Q(t)$ are the analog I and Q signals before mixing and the angular frequency of the oscillator is ω , the resulting mixed signals are [20]:

$$I : I(t)\sqrt{2}\cos(\omega t) \tag{A.1a}$$

$$Q : -Q(t)\sqrt{2}\sin(\omega t) \tag{A.1b}$$

The output of the QAM modulator is the sum of these two mixed signals:

$$x(t) = I(t)\sqrt{2}\cos(\omega t) - Q(t)\sqrt{2}\sin(\omega t) \tag{A.2}$$

This equation can be transformed using Euler's Formula¹ as:

$$x(t) = I(t)\sqrt{2}\frac{e^{j\omega t} + e^{-j\omega t}}{2} - Q(t)\sqrt{2}\frac{e^{j\omega t} - e^{-j\omega t}}{2j} \tag{A.3}$$

$$x(t) = I(t)\sqrt{2}\frac{e^{j\omega t} + e^{-j\omega t}}{2} + jQ(t)\sqrt{2}\frac{e^{j\omega t} - e^{-j\omega t}}{2} \tag{A.4}$$

$$x(t) = \sqrt{2}\frac{[I(t) + jQ(t)]e^{j\omega t} + [I(t) - jQ(t)]e^{-j\omega t}}{2} \tag{A.5}$$

Note that a complex number $z = a + bj$ can be written in exponential form as $|z|e^{j\theta}$ by leveraging Euler's Formula [42]. The θ is given by $\arg(I(t) + jQ(t))$ where \arg is the

¹ $\cos(\theta) = \frac{e^{j\theta} + e^{-j\theta}}{2}$, $\sin(\theta) = \frac{e^{j\theta} - e^{-j\theta}}{2j}$ [20]

complex argument function and $|z|$ is $\sqrt{a^2 + b^2}$. It is worth noting that \arg is equivalent to the `atan2` function in many programming languages. What distinguishes \arg from \tan^{-1} is that it delivers an angle in the quadrant corresponding to the signs of the inputs [43]. Recalling from trigonometry that \tan^{-1} has a range of $[-\frac{\pi}{2}, \frac{\pi}{2}]$, \arg can be expressed using \tan^{-1} and selectively adding or subtracting π .

Take the imaginary number $z = a + bj$. For $a > 0$, $\arg(z) = \tan^{-1}(b/a)$. For $a < 0, b > 0$, $\arg(z) = \tan^{-1}(b/a) + \pi$. For $a < 0, b < 0$, $\arg(z) = \tan^{-1}(b/a) - \pi$.

A useful property of \arg is that $\arg(a - bj) = -\arg(a + bj)$. Note that for points in quadrants I and IV ($a > 0$), $\arg(a + bj) = \tan^{-1}(b/a)$ and $\arg(a - bj) = \tan^{-1}(-b/a) = -\tan^{-1}(b/a) = -\arg(a + bj)$. In this case, $-\arg(a + bj) = +\arg(a - bj)$. For points in quadrants II and III ($a < 0$), $\arg(a + bj) = \tan^{-1}(b/a) \pm \pi$; plus for quadrant II and minus for quadrant III. Note that $a - bj$ will move the point from quadrant II to III or from III to II. Therefore, for points in quadrants II and III ($a < 0$), $\arg(a - bj) = \tan^{-1}(-b/a) \mp \pi = -(-\tan^{-1}(-b/a) \pm \pi) = -(\tan^{-1}(b/a) \pm \pi) = -\arg(a + bj)$. In this case, $\arg(a - bj) = -\arg(a + bj)$. In general, $\arg(a - bj) = -\arg(a + bj)$.

Equation A.5 can then be rewritten as:

$$x(t) = \sqrt{2}\sqrt{I(t)^2 + Q(t)^2} \left(\frac{e^{j\arg(I(t)+jQ(t))}e^{j\omega t} + e^{j\arg(I(t)-jQ(t))}e^{-j\omega t}}{2} \right) \quad (\text{A.6})$$

Using the property of \arg shown above:

$$x(t) = \sqrt{2}\sqrt{I(t)^2 + Q(t)^2} \left(\frac{e^{j\arg(I(t)+jQ(t))}e^{j\omega t} + e^{-j\arg(I(t)+jQ(t))}e^{-j\omega t}}{2} \right) \quad (\text{A.7})$$

$$x(t) = \sqrt{2}\sqrt{I(t)^2 + Q(t)^2} \left(\frac{e^{j(\omega t + \arg(I(t)+jQ(t)))} + e^{-j(\omega t + \arg(I(t)+jQ(t)))}}{2} \right) \quad (\text{A.8})$$

Euler's Formula can be applied once more yielding:

$$x(t) = \sqrt{2}\sqrt{I(t)^2 + Q(t)^2} \cos(\omega t + \arg(I(t) + jQ(t))) \quad (\text{A.9})$$

Alternatively, QAM modulation can also be represented by taking the real component of the complex expression [20]:

$$x(t) = \sqrt{2}\Re([I(t) + jQ(t)]e^{j\omega t}) \quad (\text{A.10})$$

Perhaps surprisingly, the result of a QAM modulating two signals is a single cosine signal whose amplitude and phase are functions of the input $I(t)$ and $Q(t)$. This is one reason why QAM is sometimes thought of as a hybrid between amplitude modulation and phase modulation [15]. Amplitude modulation can be accomplished by simply utilizing the $I(t)$ signal and setting $Q(t)$ to 0. Likewise, phase modulation can be accomplished by solving for the Cartesian representation of $e^{j\theta}$, where θ is the desired phase shift. $I(t)$ is set to the real component while $Q(t)$ is set to the imaginary component. Combinations of both amplitude and phase modulation can be accomplished simultaneously by varying both the amplitude and phase of $I(t) + jQ(t)$.

The use of $I(t) + jQ(t)$ in Equation A.9 and Equation A.10 are why $I(t)$ and $Q(t)$ are often thought of as components of a complex signal with $I(t)$ being the *real* component and $Q(t)$ being the *imaginary* component.

While it is possible to achieve both phase and amplitude modulation using a single modulated channel, it is impractical with many conventional DACs. Most DACs sample at a fixed rate with a fixed phase. While amplitude modulation is easy, only requiring the DAC to output different voltage levels, phase shifts are more difficult. The DAC must run at a rate much faster than the signal it is outputting in order to have the time resolution required to phase shift the signal. This is impractical as most DACs sample at a rate which is close to the symbol rate of the digital signal to be sent. The QAM modulator allows engineers to achieve both amplitude and phase modulation using two independent DACs running at a lower sample rate.

Appendix B

QAM Demodulation

The modulated QAM signal described in appendix A may directly be amplified and transmitted or may go through another mixing stage before transmission. This signal is then amplified at the receiver which downconverts the signal to IF if it is a superhetrodyne receiver. At this stage, the signal appears like a noisy, and distorted, version of the transmitted modulated signal. As was described in appendix A, it was convenient to view the amplitude and phase modulated signal as a modulated complex signal. Fortunately, a similar technique allows $I(t)$ and $Q(t)$ to be extracted from the received signal. This operation is referred to as QAM demodulation.

Let us first assume perfect reception of the signal. This assumption will be relaxed later. From Equation A.8, the received signal is:

$$y(t) = \sqrt{2}\sqrt{I(t)^2 + Q(t)^2} \left(\frac{e^{j(\omega t + \arg(I(t) + jQ(t)))} + e^{-j(\omega t + \arg(I(t) + jQ(t)))}}{2} \right) \quad (\text{B.1})$$

For brevity, let $z(t) = I(t) + jQ(t)$ with $|z(t)| = \sqrt{I(t)^2 + Q(t)^2}$.

$$y(t) = \sqrt{2}|z(t)| \left(\frac{e^{j(\omega t + \arg(z(t)))} + e^{-j(\omega t + \arg(z(t)))}}{2} \right) \quad (\text{B.2})$$

This signal is duplicated and mixed with two different sinusoidal oscillators that share the same frequency as the transmitter's oscillator and are 90 deg out of phase from each other[20].

The resulting signals are:

$$\alpha(t) = y(t)\sqrt{2} \cos(\omega t) \quad (\text{B.3a})$$

$$\beta(t) = -y(t)\sqrt{2} \sin(\omega t) \quad (\text{B.3b})$$

Using Euler's Formula:

$$\alpha(t) = y(t)\sqrt{2} \left(\frac{e^{j\omega t} + e^{-j\omega t}}{2} \right) \quad (\text{B.4a})$$

$$\beta(t) = y(t)\sqrt{2} \left(\frac{-e^{j\omega t} + e^{-j\omega t}}{2j} \right) \quad (\text{B.4b})$$

Substituting in the received signal:

$$\alpha(t) = 2|z(t)| \left(\frac{e^{j(\omega t + \arg(z(t)))} + e^{-j(\omega t + \arg(z(t)))}}{2} \right) \left(\frac{e^{j\omega t} + e^{-j\omega t}}{2} \right) \quad (\text{B.5a})$$

$$\beta(t) = 2|z(t)| \left(\frac{e^{j(\omega t + \arg(z(t)))} + e^{-j(\omega t + \arg(z(t)))}}{2} \right) \left(\frac{-e^{j\omega t} + e^{-j\omega t}}{2j} \right) \quad (\text{B.5b})$$

For signal α :

$$\alpha(t) = |z(t)| \left(\frac{e^{j(\arg(z(t)))} + e^{-j(\arg(z(t)))}}{2} + \frac{e^{j(2\omega t + \arg(z(t)))} + e^{-j(2\omega t + \arg(z(t)))}}{2} \right) \quad (\text{B.6})$$

$$\alpha(t) = |z(t)| (\cos(\arg(z(t))) + \cos(2\omega t + \arg(z(t)))) \quad (\text{B.7})$$

$$\alpha(t) = \sqrt{I(t)^2 + Q(t)^2} (\cos(\arg(I(t) + jQ(t))) + \cos(2\omega t + \arg(z(t)))) \quad (\text{B.8})$$

$$\alpha(t) = \sqrt{I(t)^2 + Q(t)^2} \left(\frac{I(t)}{\sqrt{I(t)^2 + Q(t)^2}} + \cos(2\omega t + \arg(z(t))) \right) \quad (\text{B.9})$$

$$\alpha(t) = I(t) + \sqrt{I(t)^2 + Q(t)^2} \cos(2\omega t + \arg(I(t) + jQ(t))) \quad (\text{B.10})$$

For signal β :

$$\beta(t) = |z(t)| (e^{j(\omega t + \arg(z(t)))} + e^{-j(\omega t + \arg(z(t)))}) \left(\frac{-e^{j\omega t} + e^{-j\omega t}}{2j} \right) \quad (\text{B.11})$$

$$\beta(t) = |z(t)| \left(\frac{-e^{j(2\omega t + \arg(z(t)))} - e^{-j(\arg(z(t)))} + e^{j(\arg(z(t)))} + e^{-j(2\omega t + \arg(z(t)))}}{2j} \right) \quad (\text{B.12})$$

$$\beta(t) = |z(t)| \left(\frac{e^{j(\arg(z(t)))} - e^{-j(\arg(z(t)))}}{2j} + \frac{e^{-j(2\omega t + \arg(z(t)))} - e^{j(2\omega t + \arg(z(t)))}}{2j} \right) \quad (\text{B.13})$$

$$\beta(t) = |z(t)| (\sin(\arg(z(t))) - \sin(2\omega t + \arg(z(t)))) \quad (\text{B.14})$$

$$\beta(t) = \sqrt{I(t)^2 + Q(t)^2} (\sin(\arg(I(t) + jQ(t))) - \sin(2\omega t + \arg(z(t)))) \quad (\text{B.15})$$

$$\beta(t) = \sqrt{I(t)^2 + Q(t)^2} \left(\frac{Q(t)}{\sqrt{I(t)^2 + Q(t)^2}} - \sin(2\omega t + \arg(z(t))) \right) \quad (\text{B.16})$$

$$\beta(t) = Q(t) - \sqrt{I(t)^2 + Q(t)^2} \sin(2\omega t + \arg(I(t) + jQ(t))) \quad (\text{B.17})$$

Notice from Equation B.10 that $\alpha(t)$ is the sum of $I(t)$ and a phase and amplitude modulated signal at double the frequency of the reference oscillator. Assuming that $I(t)$ has a sufficiently low bandwidth, $I(t)$ can be extracted by simply passing $\alpha(t)$ through a low pass filter [20]. Likewise, from Equation B.10, $Q(t)$ can be extracted by passing $\beta(t)$ through a low pass filter, assuming $Q(t)$ has sufficiently low bandwidth [20].

The filtered signals are:

$$\alpha(t) = I(t) \tag{B.18a}$$

$$\beta(t) = Q(t) \tag{B.18b}$$

The net result of this is that the receiver is able to present the phase and amplitude modulated signal to the baseband as two separate signals, $\alpha(t) = I(t)$ and $\beta(t) = Q(t)$. As was the case with the DAC on the transmit side, this allows the receiver to use a dual channel ADC to sample at a much lower rate than would be required to capture both phase and amplitude differences with a single channel ADC. As was the case in the QAM modulator, the received signals can be viewed as components of a complex signal with the filtered $\alpha(t)$ as the real component and the filtered $\beta(t)$ as the imaginary component. Viewed in this way, the received signal is $I(t) + jQ(t)$ which matches the view of the input signal to the transmitter as a complex signal.

Appendix C

Carrier Phase and Frequency Offset (CFO)

The analysis in appendix B assumed that the oscillator in the receive radio was precisely synchronized to the oscillator in the transmit radio. It also ignored the fact that the modulated signal experiences delay as it travels the distance from the transmitter to the receiver. In reality the oscillators in the transmit and receive radios are independent. The best case scenario is that the oscillators are operating at the same frequency but have a fixed offset in phase from each other. This is referred to as *carrier phase offset*. The delay experienced by the modulated signal as it moves from the transmit radio contributes to the carrier phase offset as the oscillator in the receive radio must be synchronized to the phase of the received signal rather than the phase of the transmitted signal. An additional complication is that oscillators are specified to operate within specific tolerances. Two oscillators specified to operate at the same frequency will operate as slightly different frequencies in practice [27]. The operating frequency of an oscillator can also change due to factors such as temperature [27]. This introduces a frequency difference between the oscillator in the transmitter and the receiver and is referred to as *carrier frequency offset (CFO)*.

To illustrate the effect these offsets have on the demodulated signal, the derivation in appendix B must be modified. It will be assumed that the received oscillator has a frequency ω_r and a phase offset from the transmit oscillator of ϕ . The signals in Equation B.5 become:

$$\alpha(t) = 2|z(t)| \left(\frac{e^{j(\omega t + \arg(z(t)))} + e^{-j(\omega t + \arg(z(t)))}}{2} \right) \left(\frac{e^{j(\omega_r t + \phi)} + e^{-j(\omega_r t + \phi)}}{2} \right) \quad (\text{C.1a})$$

$$\beta(t) = 2|z(t)| \left(\frac{e^{j(\omega t + \arg(z(t)))} + e^{-j(\omega t + \arg(z(t)))}}{2} \right) \left(\frac{-e^{j(\omega_r t + \phi)} + e^{-j(\omega_r t + \phi)}}{2j} \right) \quad (\text{C.1b})$$

For signal α :

$$\alpha(t) = |z(t)| \left(\frac{e^{j([\omega - \omega_r]t + \arg(z(t)) + \phi)} + e^{-j([\omega - \omega_r]t + \arg(z(t)) + \phi)}}{2} + \frac{e^{j([\omega + \omega_r]t + \arg(z(t)) + \phi)} + e^{-j([\omega + \omega_r]t + \arg(z(t)) + \phi)}}{2} \right) \quad (\text{C.2})$$

$$\alpha(t) = |z(t)| (\cos([\omega - \omega_r]t + \arg(z(t)) - \phi) + \cos([\omega + \omega_r]t + \arg(z(t)) + \phi)) \quad (\text{C.3})$$

Note that in Equation C.3, $[\omega - \omega_r]t$ and $[\omega + \omega_r]t$ appear. $[\omega - \omega_r]t$ is typically small for high quality oscillators while $[\omega + \omega_r]t \approx 2\omega$. Assuming the bandwidth of $z(t) = I(t) + jQ(t)$ is sufficiently narrow compared to ω , the cosine function with $[\omega + \omega_r]t$ can be filtered out using a low pass filter without affecting distorting the other cosine function. This is the same operation that was used in appendix B to extract the original signals.

After the filtering, the signal can be viewed as:

$$\alpha(t) = |z(t)| (\cos([\omega - \omega_r]t + \arg(z(t)) - \phi)) \quad (\text{C.4})$$

Using the angle sum relationship for \cos :

$$\alpha(t) = |z(t)| (\cos([\omega - \omega_r]t - \phi) \cos(\arg(z(t))) - \sin([\omega - \omega_r]t - \phi) \sin(\arg(z(t)))) \quad (\text{C.5})$$

$$\alpha(t) = \sqrt{I^2(t) + Q^2(t)} (\cos([\omega - \omega_r]t - \phi) \cos(\arg(I(t) + jQ(t))) - \sin([\omega - \omega_r]t - \phi) \sin(\arg(I(t) + jQ(t)))) \quad (\text{C.6})$$

$$\alpha(t) = \cos([\omega - \omega_r]t - \phi) I(t) - \sin([\omega - \omega_r]t - \phi) Q(t) \quad (\text{C.7})$$

Note that $\alpha(t)$ is a mix of $I(t)$ and $Q(t)$ rather than simply $I(t)$ as was the case in appendix B.

For signal β :

$$\beta(t) = |z(t)| (e^{j(\omega t + \arg(z(t)))} + e^{-j(\omega t + \arg(z(t)))}) \left(\frac{-e^{j(\omega_r t + \phi)} + e^{-j(\omega t + \phi)}}{2j} \right) \quad (\text{C.8})$$

$$\beta(t) = |z(t)| \left(\frac{e^{j([\omega - \omega_r]t + \arg(z(t)) - \phi)} - e^{-j([\omega - \omega_r]t + \arg(z(t)) - \phi)}}{2j} - \frac{e^{j([\omega + \omega_r]t + \arg(z(t)) + \phi)} - e^{-j([\omega + \omega_r]t + \arg(z(t)) + \phi)}}{2j} \right) \quad (\text{C.9})$$

$$\beta(t) = |z(t)| (\sin([\omega - \omega_r]t + \arg(z(t)) - \phi) - \sin([\omega + \omega_r]t + \arg(z(t)) + \phi)) \quad (\text{C.10})$$

As was the case with $\alpha(t)$, assuming $z(t)$ has sufficiently narrow bandwidth, the high frequency term can be filtered out with a low pass filter without distorting the low frequency term. The filtered signal is:

$$\beta(t) = |z(t)| (\sin([\omega - \omega_r]t + \arg(z(t)) - \phi)) \quad (\text{C.11})$$

Using the angle difference formula:

$$\beta(t) = |z(t)| (\sin([\omega - \omega_r]t - \phi) \cos(\arg(z(t))) + \cos([\omega - \omega_r]t - \phi) \sin(\arg(z(t)))) \quad (\text{C.12})$$

$$\beta(t) = \sqrt{I^2(t) + Q^2(t)} (\sin([\omega - \omega_r]t - \phi) \cos(\arg(I(t) + jQ(t))) + \cos([\omega - \omega_r]t - \phi) \sin(\arg(I(t) + jQ(t)))) \quad (\text{C.13})$$

$$\beta(t) = \sin([\omega - \omega_r]t - \phi)I(t) + \cos([\omega - \omega_r]t - \phi)Q(t) \quad (\text{C.14})$$

Note that $\beta(t)$ is a mix of $I(t)$ and $Q(t)$ rather than simply $Q(t)$ as was the case in appendix B.

It was useful in the QAM modulator to view the two input signals as components of a complex signal. This perspective continued to be valid for viewing the outputs of the QAM demodulator as was shown in appendix B. Let us extend this perspective for this imperfect demodulation with $\alpha(t)$ interpreted as the real component of the signal and $\beta(t)$ interpreted as the imaginary component.

The demodulated signal can then be viewed as:

$$\gamma(t) = \alpha(t) + j\beta(t) \quad (\text{C.15})$$

$$\begin{aligned} \gamma(t) &= \cos([\omega - \omega_r]t - \phi)I(t) - \sin([\omega - \omega_r]t - \phi)Q(t) \\ &+ j \sin([\omega - \omega_r]t - \phi)I(t) + j \cos([\omega - \omega_r]t - \phi)Q(t) \end{aligned} \quad (\text{C.16})$$

$$\begin{aligned} \gamma(t) &= I(t) \{ \cos([\omega - \omega_r]t - \phi) + j \sin([\omega - \omega_r]t - \phi) \} \\ &+ Q(t) \{ j \cos([\omega - \omega_r]t - \phi) - \sin([\omega - \omega_r]t - \phi) \} \end{aligned} \quad (\text{C.17})$$

$$\begin{aligned} \gamma(t) &= I(t) \{ \cos([\omega - \omega_r]t - \phi) + j \sin([\omega - \omega_r]t - \phi) \} \\ &+ jQ(t) \{ \cos([\omega - \omega_r]t - \phi) + j \sin([\omega - \omega_r]t - \phi) \} \end{aligned} \quad (\text{C.18})$$

$$\gamma(t) = \{I(t) + jQ(t)\} e^{j([\omega - \omega_r]t - \phi)} \quad (\text{C.19})$$

The result of viewing the demodulated signals, $\alpha(t)$ and $\beta(t)$, as components of a complex signal is that the received signal is the originally transmitted signal multiplied by a unity magnitude complex exponential. This manifests as a rotation of the originally transmitted signal $I(t) + jQ(t)$ about the origin in the complex plane at the receiver. If there is no frequency offset ($\omega - \omega_r = 0$), the rotation is static and determined by ϕ . If there is a carrier frequency offset, the signal rotates continuously at a constant rate determined by $\omega - \omega_r$. The wider the frequency offset, the faster the rotation. These effects are corrected in the carrier recovery block discussed in section 4.5.

In addition to phase and frequency offsets, oscillators are also subject to noise, both in amplitude and phase [44] [45]. Phase noise introduces a time varying phase shift into the oscillators. The result at the demodulator is a time varying random rotation of the received signal in the complex plane. If the phase noise is sufficiently large, transmitted constellation points may rotate into adjacent sectors in the receiver and be decoded erroneously. Phase noise therefore can place a limit on the modulation schemes that the baseband is capable of using with acceptable error rates.