Digital Communications Combined Frame Detector and Carrier Frequency Offset Estimator

by

Stephen Chandler

Submitted in partial fulfillment of the requirements
for the degree of Master of Science

at

Dalhousie University
Halifax, Nova Scotia
September 2017

Table of Contents

List of Tables . . . . **iv**

List of Figures . . . . **v**

Abstract . . . . **viii**

List of Abbreviations Used . . . . **ix**

Acknowledgments . . . . **x**

Chapter 1 Introduction **1**

1.1 Context 1

1.1.1 Approaching Channel Capacity 2

1.1.2 Channel Effects 3

1.1.3 Available Information 4

1.2 Problem and Significance 4

1.3 Approach 5

1.4 Scope 6

Chapter 2 System Overview **7**

2.1 OFDM Design 7

2.1.1 Samples Per Frame 7

2.1.2 Pilot Information 7

2.1.3 Cyclic Prefix 8

2.1.4 Other Required Features 9

2.2 Testbed Hardware 10

2.2.1 Digital and Analogue 10

2.2.2 Number Formats 12

2.2.3 Calibration 13

2.2.4 Implementing the Reference Design 15

2.3 Software Tools 16

2.4 Golden Model 17

Chapter 3 Frame Detector **18**

3.1 Correlator 19

3.2 Peak Detector 20

3.2.1 Peak Timing Calculation Strategies 20
3.2.2 Alternating Peak Finder 20
3.2.3 Receiver Behaviour in Different Situations 23
3.3 Frame Detect Controller 31
3.3.1 Frame Detect Controller Core 33
3.3.2 Frame Detect Controller FSM 36
3.3.3 Check Pulse Received Within Variance 37
3.3.4 Frame Detect Variance FSM 38
3.3.5 Frame Detect Countdown Timer 41
3.4 Markov Analysis of Peak Detector and Frame Detect Controller . . . 42

Chapter 4 Carrier Frequency Offset Estimator 52
4.0.1 Classic Phase Estimation 52
4.0.2 Differential Detection 53
4.0.3 Limits of Phase Offset Estimation 57
4.0.4 Example Phase Offset Estimates 58
4.1 Arctangent 63
4.2 Division 67
4.3 Internal Arctangent 71
4.4 Smoothing Filter 72

Chapter 5 Conclusion 75

References 76

List of Tables

Table 1 The project cores are broken down by hierarchy and software environment used to work with it. 19

Table 2 Example LUT with $n = 2$. 70

List of Figures

Figure 1 Representation of how the OFDM pilot is arranged within a frame of OFDM data in the time-domain. 7

Figure 2 Overview of the radio testbed connections and structure left side. 10

Figure 3 Overview of the radio testbed connections and structure right side. 11

Figure 4 A visual mapping of the OFDM pilot tones to their indexes. . 13

Figure 5 Constellation errors caused by different circuit effects [17]. . . 14

Figure 6 The System Generator logo 16

Figure 7 The Vivado logo 17

Figure 8 The Frame Detector design in System Generator. 18

Figure 9 The Carrier Frequency Offset Estimator design in System Generator. 18

Figure 10 The maximum value within each frame timing is represented by a pulse in time. 21

Figure 11 A peak detector which finds the peak balanced around the center point of the simulated correlation. 22

Figure 12 The probability that a peak will be calculated at a particular array index by the alternating peak finder for Gaussian white noise. 24

Figure 13 The probability densities of delay between correlation peaks at different SNRs. 26

Figure 14 The peak timing distribution caused by noise behaves the same independent of when the most recent pulse has been received. 27

Figure 15 The probability densities of delay between correlation peaks at different SNRs near the acceptance window. 28

Figure 16 The probability measured before, during, and after the acceptance window, along with some fits to the data. 28

Figure 17 One sequential peak finder component which forms part of an alternating peak finder. 29

Figure 18 The logic which ensures only the detected peaks close to the center of the first sequential peak finder are produced as outputs. 30

Figure 19 An overall picture of the Golden Model simulation with the peak finder detecting correlation peaks. 31

Figure 20 The frame_detect_controller is implemented as a System Generator black box. 32

Figure 21 The results of the FIRST_PULSE_TIMING test. 35

Figure 22 THIRD_MISS_TIMING test for the $-w$ side. 35

Figure 23 GOLDEN_MODEL_5_DATA uses simulation data from Golden_Model_5. 36

Figure 24 An illustration of Frame Detect Controller FSM advancing from the state of no detection to being locked-on. 37

Figure 25 First pulse being received by Frame Detect Variance FSM. . . 39

Figure 26 Second pulse being received and counting as within the acceptable time window by Frame Detect Variance FSM. 39

Figure 27 Second pulse being missed within the acceptable time window by Frame Detect Variance FSM. 40

Figure 28 Test of Frame Detect Countdown Timer. 41

Figure 29 A basic graphical description of a Markov chain. 42

Figure 30 As the number of fall-out states increases, lock-on occurs more at lower SNRs. 48

Figure 31 The minimum values are shown approximately at -15 dB SNR, which represent the portion of state transitions spent locked-on to a noise signal when there is no signal being received. 49

Figure 32 Derivatives of the transition functions. 50

Figure 33 System Generator implementation of Carrier Frequency Offset Estimator top level. 52

Figure 34 A local picture of the peak finder locating the maximum value. 59

Figure 35 The sum of magnitudes of the real and imaginary components of the correlation is shown in red, and the estimated phase offset using the Full Arctangent function at the time given by the peak detector is shown in blue. 60

Figure 36 The decrease in the variance of phase offsets detected at different SNRs is shown. 60

Figure 37 OFDM example affected by AWGN where we plot the received correlation's peaks, real value, imaginary value, and phase offset. 61

Figure 38 Real and imaginary autocorrelation values. 61

Figure 39 Real and imaginary autocorrelation peaks. 62

Figure 40 A high level description of the Phase Offset Estimator. 63

Figure 41 Output type selection logic and pipelining. 65

Figure 42 Division type selection, division circuit, arctangent circuit with constant and negation selection, and pipelining. 66

Figure 43 The appropriate output is selected using a multiplexer. 66

Figure 44 This diagram contains the first part of division where the absolute value is taken, the bits of the representation are shifted so the denominator lies on the range [0.5,1), and the index of a LUT step is acquired and used to obtain the stored value to send *d* close to 1. 67

Figure 45 This diagram shows the multiplication of D' and N' with the LUT factor, followed by the 2-D" calculation, and then multiplication of N" with the GS factor. 70

Figure 46 The arctangent index is obtained from the incoming number representation, and then applied to a LUT to get the result. . 72

Figure 47 Overview of the AR filter in System Generator. 73

Figure 48 The System Generator block diagram of the AR filter datapath. 74

Abstract

This thesis describes a flexible digital logic design which allows a communications receiver to synchronize with a data frame and estimate its frequency offset using only its cyclic prefix and tail data. The design is implemented as digital logic running on a field-programmable gate array which controls a software defined radio as a testbed. Synchronization reliability is improved using a finite state machine which locks-on to a received signal and falls-out-of-lock in noise. The state machine reliability is calculated using Markov chain analysis. The frequency offset is estimated using a custom complex argument function which does not use a small angle approximation. When run on the testbed, the design successfully provides synchronization and a frequency offset estimate using only the received data's cyclic prefix and tail data.

List of Abbreviations Used

ADI	Analog Devices Inc.
AR	autoregressive
ARMA	autoregressive-moving-average
AWGN	additive Gaussian white noise
CDF	cumulative distribution function
CP	cyclic prefix
CPU	central processing unit
CRC	cyclic redundancy check
DAC	digital-to-analogue converter
DFT	discrete Fourier transform
FFT	fast Fourier transform
FPGA	field programmable gate array
FSM	finite-state machine
FSMD	finite-state machine with datapath
GPU	graphical processing unit
GS	Goldschmidt
HDL	hardware description language
IIC	inter-integrated circuit
ILA	Integrated Logic Analyzer
IP	intellectual property
ISI	inter-symbol interference
LDPC	low-density parity check
LO	linear oscillator
LUT	lookup table
MA	moving average
MIMO	multiple-input multiple-output
ML	maximum likelihood
MSE	mean squared error
OFDM	orthogonal frequency division multiplexing
PAPR	peak-to-average power ratio
PLL	phase-locked loop
PSK	phase-shift keying
QAM	quadrature amplitude modulation
SISO	single-input single-output
SDK	Xilinx Software Development Kit
SDR	software defined radio
SNR	signal-to-noise ratio
SSB	single side-band
SysGen	System Generator
UART	universal asynchronous receiver/transmitter
UMDCC	Ultra Maritime Digital Communications Centre
XSDK	Xilinx Software Development Kit

Acknowledgments

I would like to thank my internal supervisors Christian Schlegel and Kamal Sankary, external supervisor Marcel Jar, as well as Robert Hang who has provided FPGA hardware design experience and guidance. Thanks to the co-op students Guanxiong Gong and Muhamud/Abdi Naka for their assistance in helping set up the test environment, and to other lab members, friends, and family for their support. I would also like to thank NSRC who have provided funding through a research grant.

Chapter 1 Introduction

Digital communications has become one of the most useful technologies in the world. It allows cellphones and computers to download or upload text, data, audio, and video within milliseconds using wired connections and wirelessly from Wi-Fi routers, cell towers, satellites, and more. However, digital communications are less developed underwater. Most radio waves are attenuated too quickly to be useful at a range. The underwater acoustic channel uses sound instead of light to transmit information, and is promising because sound can travel much further than radio for the same amount of energy underwater. For example, a 1 kW projector could transmit a useful signal 100s of kilometers [1].

The problem is that the underwater acoustic channel has more ways to corrupt a message than radio. These effects in underwater acoustic are made worse than similar effects using light because the speed of sound through water is much slower than the speed of light through air. The underwater environment also contains complicating factors such as surface waves, salinity gradients, temperature gradients, and more. Although each effect can be studied and accounted for on its own, the variety of effects means the underwater acoustic channel is perhaps the most challenging communications channel which exists. Despite this, it remains the most promising way to communicate through the ocean. Developing digital communications designs, such as for a cellular network, is already an incredible feat which uses vast amounts of knowledge, time, and resources. Designing for the underwater acoustic channel is even harder.

To create a working system for the underwater acoustic channel requires a system which can at least work over a radio channel. The system needs a design framework which is flexible enough to handle most different underwater situations, and an implementation which is flexible enough to test various new components over radio before being used underwater.

1.1 Context

My research has been performed within the Underwater Maritime Digital Communications Centre (UMDCC), whose goal is to perform research which supports underwater acoustic digital communications. The approach has been to develop a big picture view of the underwater acoustic channel, and to approach its capacity using successful theory and practice from digital communications.

1.1.1 Approaching Channel Capacity

The framework employed is intended to create designs with the flexibility to transmit information from a source to a receiver near channel capacity. The channel capacity describes the limits of a channel, and is the maximum rate of information which can be reliably transmitted across a communications channel measured in units of spectral efficiency $\frac{\text{bits}\backslash\text{s}}{\text{Hz}}$ versus signal power-to-noise power ratio $\frac{E_b}{N_0}$. At channel capacity the spectral efficiency is increased at the cost of more energy per bit transmitted. A practical expense is the complexity of developing a system which can approach capacity for a particular situation.

The channel capacity for a noiseless single-input single-output (SISO) channel was originally described by Shannon in [2]. The SISO limits have been approached in recent years using coding techniques such as well-designed low-density parity check (LDPC) codes and turbo codes. Multiple-input multiple-output (MIMO) channels have also been developed which take advantage of multiple antennas to create spatially-separated channels which reach a higher overall capacity than the SISO case. Recent developments attempt to use a large number of antennas for massive MIMO to approach the limits of the MIMO channel capacity. An example of the channel capacity achievable by underwater communications is given in [1].

Flexibility is important to the field of digital communications. Increasing the possible designs which could be created increases the chance the designers can find an appropriate design for a particular situation. Also, if generic design parameters can be found across different communication situations and mapped into effective design choices, then fewer resources are needed to approach capacity.

One of the ideas underlying this flexibility is to separate the hardware which must be specialized for a particular communications system, such a radio transmitter or an acoustic hydrophone, from the computations which are generic to any particular communications system such as frame detection or channel estimation. The analogue hardware which interfaces with the channel is almost necessarily fixed, while the generic calculations can be performed by digital hardware. SDRs can alleviate some of the analogue restrictions by, for example, providing an adjustable carrier frequency. Practical constraints still exist for SDR such as interface types, required clock rates, required clock precision, cost, and more.

1.1.2 Channel Effects

There are several effects on a transmitted signal which contribute to reduce how much information it can transmit. Some of these channel effects can be predicted and partially or completely compensated for. Some common channel effects are outlined in this section, and more effects may exist depending on the channel medium.

A **carrier frequency offset** occurs when a signal arrives at a receiver with a different carrier frequency from the receiver's demodulation frequency. This can occur when there is an offset between the transmitter's modulation frequency and the receiver's demodulation frequency, when there is Doppler in the system caused by the motion between the transmitter and receiver, or by motion in the channel medium.

Fading occurs when a signal produced by a transmitter is projected into the environment of a channel, it may be reflected, scattered, or diffracted by several surfaces. The multiple paths which the energy takes may possess a different delay, amplitude, and phase offset, and the received signal will be affected in a way known as **multipath fading** [3]. Some paths may also be grouped into bundles with similar amplitude or phase effects which can be analyzed separately. **Shadowing** is the effect of an obstacle blocking the signal or changing its wave propagation so that its energy is directed away from the receiver [4].

Rayleigh and Rician fading are the main theoretical models which describe the phase and amplitude distribution of a received signal. Rayleigh fading occurs where there are many paths and reflections which act to create an independent and normally distributed real and imaginary signal. Where there is a single approximately clear path and some side reflections with smaller energy, the signal may fade according to a Rician distribution. In the Rician case, the signal contains a real and imaginary part which each vary as independent and normal random variables, and also a nonzero average real and imaginary value [5].

Interference occurs where there is signal energy from the environment which is not produced by the intended transmitter but which is received at the receiver. There are national and international guidelines intended to reduce radio interference over certain radio frequencies in order to protect important applications such as GPS, and to ensure only those who've purchased access to a particular set of frequencies can use them effectively. The underwater acoustic channel may receive interference from non-communication sources such as ocean life or ships.

Attenuation is loss of received signal power. Two types of attenuation include **absorption** and **free-space path loss**. Absorption occurs where objects in a channel environment convert a transmitted signal into non-receivable forms of energy, such as radio energy into heat. There are certain radio frequencies which are strongly affected by absorption in the atmosphere, such as at the attenuation of radio frequencies by oxygen at around 60 GHz. Free-space path loss is attenuation over distance from the transmitter due to the signal energy spreading across space as it propagates [6]. This can occur for light in free space, and other environments such as an underwater acoustic channel where the energy can travel away from the receiver and become attenuated below the ambient noise.

Signal clipping is the effect that occurs when a signal to be transmitted is larger than the amplifier can produce, or lies in the non-linear operating range of the amplifier [7]. Signal clipping can occur for information dense signals with a large peak-to-average-power-ratio (PAPR), such as an OFDM signal. Some modulation schemes with lower-entropy signals such as phase-shift keying (PSK) have low PAPR. Signal clipping is an issue where an amplifier has a non-linear high power operation such as those used in satellite communications. One method to reduce signal clipping by the transmitter is to operate the transmit amplifier below its maximum power range, which requires using a more expensive transmitter for a given power. Another method is to choose a modulation scheme with a low PAPR at the sacrifice of some channel capacity. Finally, a transmitter-side feedback loop can be used to adjust the produced signal in a way that allows the amplifier to remain linear across its entire operating range.

1.1.3 Available Information

In order to estimate the channel frequency response or other effects, the receiver needs to know something about the transmitted information. This knowledge can be an assumption of a linear channel, prior distributions or other statistical information about the transmitted data, or training data/pilot information known at the receiver. **Non-blind** methods use training data, and **blind** methods use none [8].

It is desirable to transmit as little pilot information as possible in order to increase the spectral efficiency of the transmitted signal. The cost of reduced pilot information may be loss of reliability or increased computational effort by the receiver.

1.2 Problem and Significance

The very first thing a receiver needs to do is to find when a frame of data has been received. It cannot rely on any other circuit because it is the starting point for every other algorithm which can be used on the data. Whatever method chosen must be robust against noise, and it is desirable that it not take up channel capacity which could otherwise be used for transmitting data.

Another problem which can affect a testbed is the difference in clock frequency between the transmitter and the receiver. This appears as a frequency offset in the carrier frequency at the receiver, and is a similar effect as the coarse Doppler offset caused by motion between the transmitter and receiver or by motion in the channel medium. This frequency offset causes the received data, visualized as a constellation, to rotate over time.

These problems are some of the first which must be tackled for a successful testbed. They also can both be tackled by taking advantage of the same information.

1.3 Approach

This thesis is part of a project to design a flexible framework for creating digital communications systems operating in different scenarios. The focus is on identifying when a frame of data has been received and then estimating its average Doppler offset in a combined way using only the received data. The framework is based on an orthogonal-frequency division multiplexing (OFDM) design, and is implemented using a field-programmable gate array (FPGA)-controlled software-defined radio (SDR) testbed.

OFDM is a modulation scheme which provides control over the time and frequency behaviour of a transmitted signal in a computationally efficient way. An FPGA is a circuit which can implement concurrent digital logic. An FPGA can be compared to a central processing unit (CPU) which performs calculations in series, and a graphics processing unit (GPU) which performs calculations in parallel. A SDR is a radio which can be adjusted to different applications based on the digital logic used to control it.

Designs or components can be developed within the Ultra Maritime Digital Communications Centre (UMDCC) lab and verified using the radio testbed. Having a platform on which a design can be tested in real time is invaluable because it provides:

Confidence that a system or component has been implemented correctly

Diagnostic information to help identify design issues where they are easier to fix

Evidence of the system working in ”real world” scenarios

The frame synchronization and carrier frequency offset estimation are both performed using the cyclic prefix, which is already included as part of the OFDM system.

1.4 Scope

The purpose of this thesis is to communicate how my work contributes towards a flexible design of digital communication systems. The scope of my contribution includes the development and implementation of:

An interface between the FMCOMMS1 reference design and a custom design

A frame detection algorithm

A phase offset estimation algorithm

The system overview describes the OFDM design, the testbed hardware, the software tools used, and the simulation environment. The frame detection algorithm consists of a correlation calculation, determining when this correlation is at its peak value, and using this timing information in Frame Detect Controller to obtain a better estimate. A Markov chain is used to analyze the performance of the Frame Detector system. Finally, the timing of the correlation peak is used along with the correlation samples to estimate the carrier frequency offset. Different aspects of this calculation are considered including a way to retain a simple but accurate number representation throughout the calculation and a novel way to perform division. The result is smoothed using a simple autoregressive filter and produced as an output.

Chapter 2 System Overview

2.1 OFDM Design

The design described in this document is tailored to a specific OFDM scheme developed by the UMDCC. In particular, there are three main design components selected for this system which help make it flexible. The detailed structure of the OFDM signal is described in [9], and shown in Figure 1.

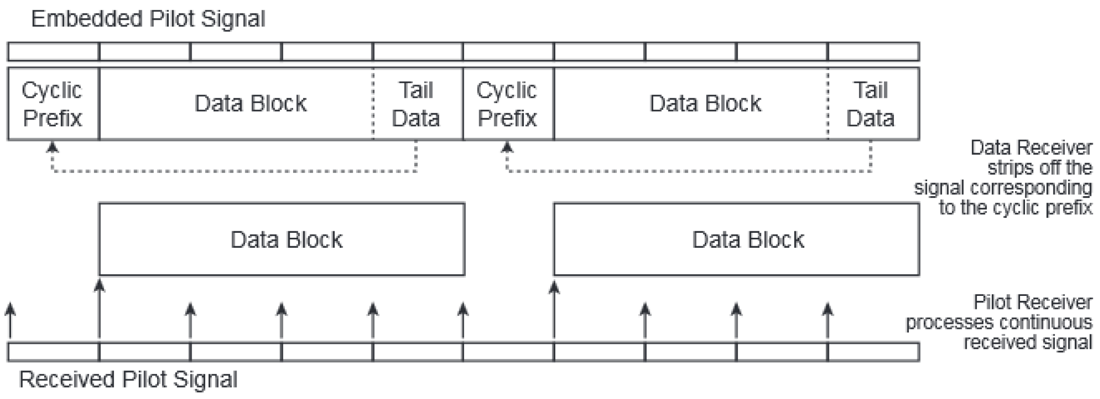


Figure 1: Representation of how the OFDM pilot is arranged within a frame of OFDM data in the time-domain.

2.1.1 Samples Per Frame

The first design choice is the number of samples N which compose the OFDM frame to be transmitted. In the example design, $N = 1024$ in order to account for a large variance in a channel frequency amplitude response. A larger N represents more frequency bands across the same bandwidth, and so each band will be narrower. The narrower the bands, the closer each one appears to be frequency flat, and the better approximation each band is to the Nyquist criterion. The main expense of using a larger value of N is the computational complexity of using a larger discrete Fourier transform (DFT) calculator, and the latency of the calculation requiring N samples. The DFT is performed using a fast Fourier transform (FFT) algorithm.

2.1.2 Pilot Information

Another decision is the use of pilot data. Pilot data can be used to estimate the channel frequency response [10] and to correct for fine delay or phase offsets within the received signal. The pilot data or recovered channel information can be averaged over time to provide a better estimation of the channel.

The pilot data is selected to be frequency-flat when recovered at the receiver by a pilot-data-sized DFT. These frequency-flat sequences will produce tones at discrete frequencies across the entire spectrum of the received OFDM frame when acted on by the N-input DFT. At each frequency these tones will possess the amplitude which corresponds to the channel frequency response at that frequency. The remaining channel frequency response can be estimated by interpolating between the pilot tones. This interpolation is performed using a DFT calculation described in [9].

A trade-off must be made between the amount of pilot data in the frame versus the amount of transmitted information. In the example design, the pilot data is an $M = 64$ sample sequence. An image of the pilot frequency amplitude response is shown in Figure 4.

2.1.3 Cyclic Prefix

A common OFDM design choice also used in the generic design is to include a cyclic prefix (CP) [11]. The cyclic prefix is a tool used to approximate the effect of a circular convolution of the transmitted signal with the channel in the time domain. It also helps correct for inter-symbol interference (ISI). The cyclic prefix samples are copied from the tail data of the OFDM frame and placed at the beginning of the frame as a prefix. As the channel convolves the data, the tail data repeats the cyclic prefix and fills in elements of the convolution matrix so that the convolution matrix approximates a circular convolution. This works as long as the cyclic prefix is longer than the channel's delay spread. By the circular convolution theorem,

$$\begin{aligned} \underset{\rightarrow}{C}\,\underset{\rightarrow}{x} &= \underset{\rightarrow}{c} \circledast \underset{\rightarrow}{x} \\ &= \mathbf{F}^{-1}\{\mathbf{F}\{\underset{\rightarrow}{c}\}\mathbf{F}\{\underset{\rightarrow}{x}\}\}, \end{aligned} \tag{1}$$

where $\underset{\rightarrow}{c}$ is the vector used to create the circulant matrix $\underset{\rightarrow}{C}$ acting on the transmitted data $\underset{\rightarrow}{x}$, and $\circledast$ represents circular convolution. The receiver performs a FFT on the received time-domain OFDM frame which has been affected by the channel response $\underset{\rightarrow}{H}$, plus noise which is ignored here, to obtain

$$\mathbf{F}\{\underset{\rightarrow}{H}\,\underset{\rightarrow}{x}\} = \mathbf{F}\{\underset{\rightarrow}{h}\}\mathbf{F}\{\underset{\rightarrow}{x}\}. \tag{2}$$

We can recover the sent information using the estimated channel frequency amplitude response $\mathbf{F}\{\underset{\rightarrow}{\hat{h}}\}$ as in

$$\mathbf{F}\{\underset{\rightarrow}{\hat{h}}\}^{-1}\mathbf{F}\{\underset{\rightarrow}{h}\}\mathbf{F}\{\underset{\rightarrow}{x}\} = \mathbf{F}\{\underset{\rightarrow}{\hat{x}}\}. \tag{3}$$

Channel estimation using this methodology is described in [12]. This vector-vector multiplication is a low-cost operation versus a vector matrix multiplication.

This generic OFDM design provides a platform which has adjustable parameters to ensure:

Timing estimation

A close approximation to Nyquist sampling

Channel frequency response estimation

Channel equalization

The cost of these features are small, and trade-offs can be made in the number of frequency bins or the number of pilots to account for the variance of the frequency response and the channel order.

2.1.4 Other Required Features

The required features for a minimum-working OFDM system which are currently missing include frame synchronization and coarse carrier frequency offset estimation and compensation. In most system designs a phase-locked loop (PLL) is synchronized with the transmitted signal's modulation frequency [13]. To provide flexibility for underwater acoustic communication, we do not assume that this high-speed clock is available and all frequency modulation effects are handled using the received samples digitally. We would like to perform the carrier frequency offset estimation using only the received data sampled at the Nyquist sampling rate of the transmitted information.

The main idea is to take advantage of the CP for both synchronization and carrier frequency offset estimation using the knowledge that the CP data is repeated in the structure of the frame. While the generic OFDM design is the desired application, the frame synchronization and coarse carrier frequency offset estimation described in this thesis can be performed for any complex signal whose discrete samples are replicated at the beginning and end of a frame.

Consider a frame of $N + M$ samples where the cyclic prefix is composed of the M tail data samples. Frame start detection is performed by correlating M samples with the corresponding samples located at N discrete time steps in the past. When the M repeated samples are multiplied by the complex conjugate of the original samples and added, a correlation of the data in the two windows is performed as in Equation 29. The correlation peak at each different signal-to-noise ratio (SNR) is simulated and shown in Figure 15.

This sliding correlator technique is a powerful and commonly used tool to detect the start time of a data frame using a known pseudorandom signal as mentioned in [14]. The main difference is that here the CP is used. Although the cyclic prefix is not a pseudorandom sequence, it has high entropy because it is formed using the data and pseudorandom pilot information being transmitted. Note that the maximum entropy signal for a given average power is Gaussian distributed [15]. A high-entropy signal will almost certainly correlate with only itself, and the correlation will be apparent even at low SNR.

2.2 Testbed Hardware

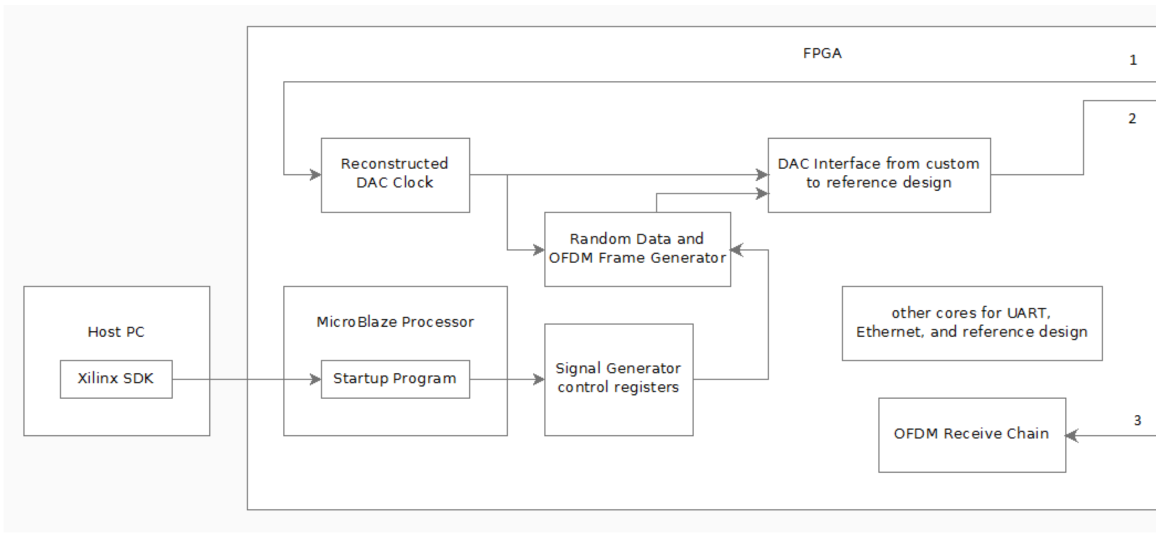

FPGA
1
2
3
Reconstructed DAC Clock
DAC Interface from custom to reference design
Random Data and OFDM Frame Generator
Host PC
Xilinx SDK
MicroBlaze Processor
Startup Program
Signal Generator control registers
other cores for UART, Ethernet, and reference design
OFDM Receive Chain

Figure 2: Overview of the radio testbed connections and structure left side.

2.2.1 Digital and Analogue

The hardware used for the testbed is separated into the digital side and analogue side. The FPGA development board implements the digital logic and:

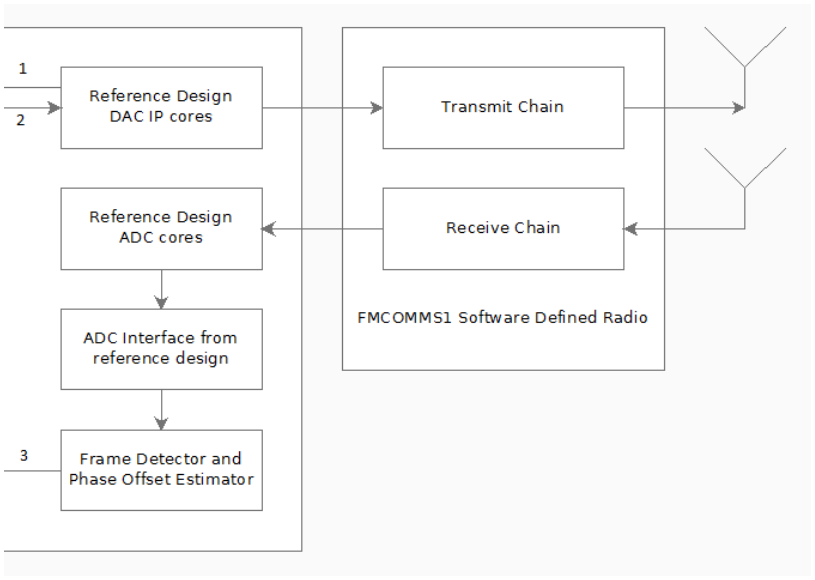

1
2
Reference Design DAC IP cores
Transmit Chain
Reference Design ADC cores
Receive Chain
ADC Interface from reference design
FMCOMMS1 Software Defined Radio
3
Frame Detector and Phase Offset Estimator

Figure 3: Overview of the radio testbed connections and structure right side.

Performs the OFDM design

Interfaces between the OFDM design and the FMCOMMS1 reference design

Operates the reference design

Implements a MicroBlaze softcore processor

Provides a universal asynchronous receiver/transmitter (UART) connection between the host PC and the MicroBlaze processor

Provides Integrated Logic Analyzer (ILA) access to the FPGA logic to measure the circuit as it operates in real time

The analogue side performs radio-specific functions such as:

Providing a stable clock between the FPGA and the FMCOMMS1

Digital to analogue conversion

Combining the real and imaginary signals into a single time-domain signal

Frequency modulation

Providing a transmit antenna/cable

Providing a receive antenna/cable

Frequency demodulation

Analogue to digital conversion

The digital hardware is a Xilinx VC707 FPGA used to control an AD-FMCOMMS1-EBZ software-defined radio by Analog Devices Inc (ADI). An overview of the connections between the systems is shown in Figure 2 and 3.

ADI is an electronics manufacturer which produces software-defined radio boards and the reference design software required to operate them on FPGA platforms. ADI's SDRs can be controlled electronically to operate over a range of carrier frequencies and bandwidths.

The ADI reference design consists of a set of files provided by ADI used to set up an initial design. By default it produces a real and imaginary sinusoid waveform under the control of software which runs on a MicroBlaze softcore processor on the host FPGA. This reference design was used as a basis to interface with the lab's generic OFDM design.

2.2.2 Number Formats

One way to describe the number formats of a binary representation is <number representation type>(<number of bits in the representation>, <number of fractional bits>). For example, "unsigned binary (2,1)" describes the number representation consisting of $(0.0)_2 = (0.0)_{10}$, $(0.1)_2 = (0.5)_{10}$, $(1.0)_2 = (1.0)_{10}$, and $(1.1)_2 = (1.5)_{10}$.

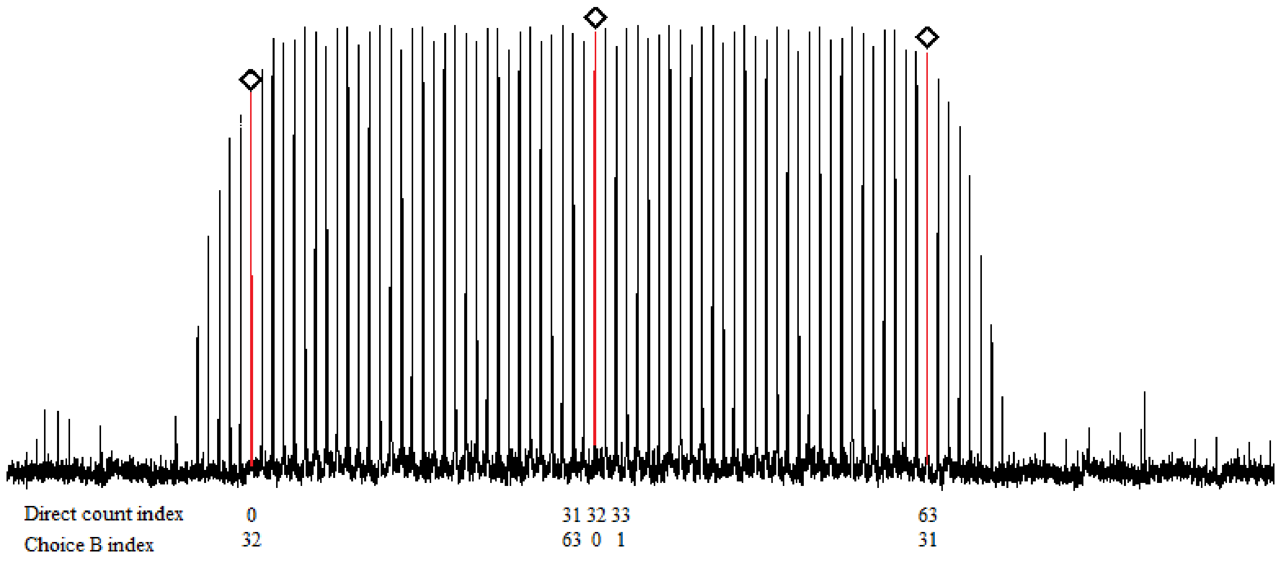
The expected inputs for the FMCOMMS1's digital-to-analogue converter (DAC) interface are two 14-bit STD_LOGIC_VECTOR signals in VHDL, or standard signals in Verilog. Each of these signals is in the fixed-point two's complement (14,13) number format. Using $\{\}_2$ to represent binary numbers and $\{\}_{10}$ to represent decimal numbers:

$$\{\textstyle\sum_{i=-13}^{0} b_i 2^i\}_{10} = \{b_0.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}b_{-7}b_{-8}b_{-9}b_{-10}b_{-11}b_{-12}b_{-13}\}_2$$

where b_i are binary values and the subscript $i \in \mathbb{Z}$, $-13 \leq i \leq 0$ corresponds to the power of two multiplied by the bit value. A similar interface was created for the FMCOMMS1's ADC. Note that the in-phase signal corresponds to the real portion of a complex-valued sample, and the quadrature-phase signal corresponds to the imaginary portion.

It is desirable to be able to handle any input signal on the 14-bit analogue-to-digital converter (ADC) range. One design choice could be to ensure the incoming signal is always balanced using feedback so that it lies within a certain range of the two's complement (14,13) representation. Another method could be to use a floating point number representation, but floating point numbers require a large amount of design time and circuit complexity [16]. Assuming the use of fixed-point arithmetic throughout most of the circuit, the design described in this thesis retains full precision arithmetic for the correlation. The aspects of the circuit which do not use full precision are the circuits which no longer need the correlation values, such as the timing pulses from the Frame Detect Controller, and the approximations made within the arctangent calculation. The desire for this 14-bit input to retain full precision has driven several design choices.

Several test signals are built into the transmit side calculations. These include a sinusoidal test signal, a square test signal, a triangle test signal, and a test signal using the OFDM pilots. The power spectrum of the transmitted OFDM pilots are shown in Figure 4 along with the index of the individual pilots.



Choice B index	CHOICE_B
0	0x0000_0000_0000_0001
16	0x0000_0000_0001_0000
32	0x0000_0001_0000_0000
48	0x0001_0000_0000_0000

Figure 4: A visual mapping of the OFDM pilot tones to their indexes.

2.2.3 Calibration

There exist some parameters of the FMCOMMS1 which can be calibrated for improved operation versus the shipped device behaviour. The main effects which can be handled using calibration are linear oscillator (LO) leakage, quadrature phase mismatch, and I/Q gain imbalance [17].

> I/Q gain imbalance in QAM results in a scaling of the constellation as shown in Figure 5(B)
>
> Quadrature phase mismatch occurs where there is not exactly 90 degrees phase difference between the in-phase and quadrature-phase signals and its effect on the example QAM constellation is shown in Figure 5(C)
>
> LO leakage results in a constellation which is offset from the origin as shown for the example case of quadrature amplitude modulation (QAM) in Figure 5(D)

Each of these effects can be caused by multiple different factors within the FM-COMMS1. Correcting for each also improves sideband or spurious frequency suppression.

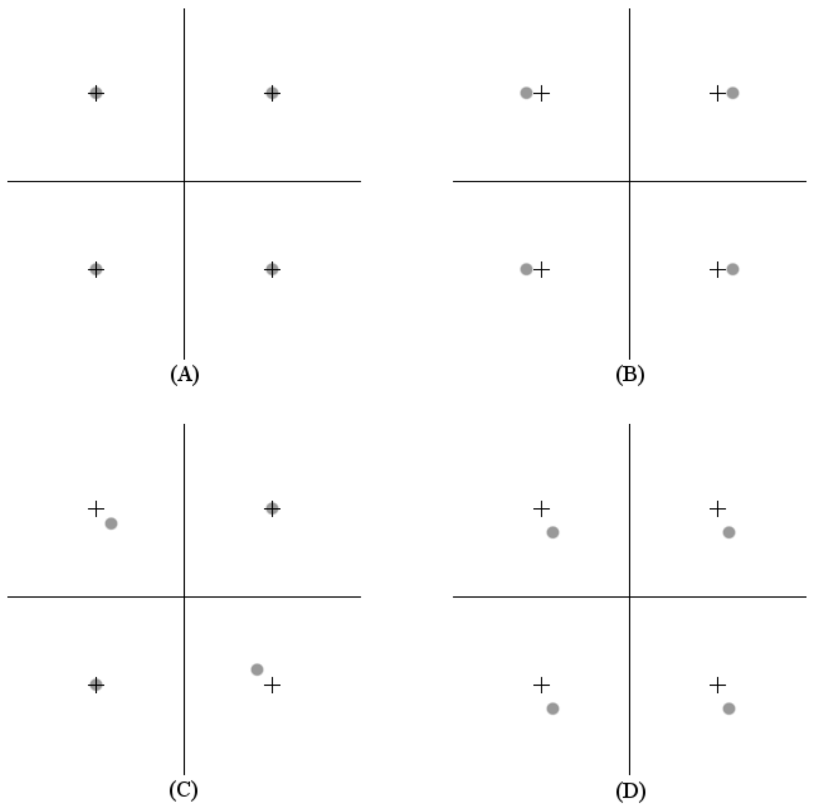


Figure 5: Constellation errors caused by different circuit effects [17].

Calibration can be performed using closed-loop feedback or open-loop control. Analog Devices suggests that an open-loop control, also known as factory calibration, can allow effective suppression of the issues presented. This is typically done by producing a test single side-band (SSB) signal and adjusting the parameters of DC offset, full-scale adjustment, and phase adjustment to maximize carrier suppression and sideband suppression. Carrier suppression results when the carrier frequency/DC offset error/LO leakage is as close to 0 as possible. Sideband suppression represents when the image of the SSB signal around the modulation frequency is as close as possible to 0.

2.2.4 Implementing the Reference Design

The process required to set up the reference design includes:

Downloading and installing the hardware files

Connecting them together appropriately

Performing Vivado synthesis

Inserting the correct integrated logic analyzer (ILA) cores

Performing Vivado implementation

Generating the bitstream file in Vivado

Generating the hardware files for Xilinx Software Development Kit (SDK) in Vivado

Reading these files into SDK

Generating the board support package in SDK

Importing the software into SDK

Generating the software files for the built-in MicroBlaze softcore processor in the hardware design

Running the FPGA hardware design while installing software on the MicroBlaze processor

The reference design software is composed of C files which can be modified and compiled. These files are installed on the MicroBlaze while it is emulated by the FPGA. These software changes can be used to change different parameters on the FMCOMMS1. The MicroBlaze can communicate to a microprocessor on the FMCOMMS1 using an Inter-Integrated Circuit (IIC) connection through a pin in the FMC connector.

Modifications were made to the Verilog code of the reference design in order to form an interface with the OFDM design. The changes include making the signals being sent to the DAC and being received from the ADC available outside the reference design. These signals are connected to the ADC and DAC interfaces which account for the number format and offer reliable connections for the transmitter chain and receive chain.

The UART provides some control while the software is running on the MicroBlaze. Logic signals on the FPGA can be measured during operation using ILA cores.

2.3 Software Tools

Several software tools are used in this project to work at different levels of hardware design. Theory and simulation have been performed using MATLAB and also Xilinx System Generator (SysGen). The OFDM design interfaces with the FPGA design using Xilinx's Vivado suite. Xilinx Software Design Kit (XSDK or SDK) is used to interface with the softcore MicroBlaze processor. The project is constrained to specific versions of each software which include Windows 7 Professional 64-bit, MATLAB 2013b, and Vivado 2014.2. System Generator and XSDK are installed as part of the Vivado suite.

MATLAB is a common software package for performing mathematical simulation. System Generator is a plugin designed by Xilinx which interfaces with MATLAB so that FPGA hardware blocks can be placed in MATLAB's Simulink environment. It is operated similarly to Simulink where diagrams and dialogue boxes are used to create computational representations of systems. Some examples of SysGen designs are shown in Figures 17 and 20. It provides a simulation environment suited for hardware logic testing using clocked logic, and Simulink calculations can be combined with SysGen calculations to help with simulation. An appropriately created design in SysGen can be imported to Vivado as a hardware intellectual property (IP) core.

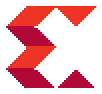


Figure 6: The System Generator logo

Vivado is a software suite developed by Xilinx which is used to interface with their FPGAs and development boards such as the VC707. It was created from their previous software suite Project Navigator, and ties together several different software tools used to create FPGA designs in a more consistent way than its predecessor.

Vivado can receive digital logic designs in various different forms such as VHDL or Vivado hardware description language (HDL), IP cores, and more. It can take digital logic through synthesis, implementation, and installing onto an FPGA. Vivado also includes Xilinx Software Development Kit (XSDK or SDK) which can be used to create operating systems and software to place into a processor implemented on an FPGA.

Figure 7: The Vivado logo

2.4 Golden Model

The Golden Model simulation is a System Generator model developed by the UMDCC to simulate and implement the OFDM calculation. The transmit chain:

Generates the data to be transmitted

Calculates and attaches a cyclic redundancy check (CRC)

Computes the inverse DFT to produce the time domain signal of the data

Prepends the time domain signal with the cyclic prefix

Adds the pilot information in the time domain

At this point the OFDM frame is complete. Golden Model simulates the effect of an additive Gaussian white nose (AWGN) channel acting on the transmitted data before sending it to the receive chain. The SNR can be adjusted for the simulation. The receive chain

Performs frame detection and carrier frequency offset estimation

Performs timing correction using the pilot information

Performs a DFT to obtain the received OFDM frame in the frequency domain

Estimates the channel using the pilot information

Compensates for the action of the channel on the data and recovers the data

Compares the transmitted and received data

Golden Model 6 is the version used for tests performed throughout this thesis, unless otherwise specified.

Chapter 3 Frame Detector

Frame detection and carrier phase offset estimation are calculated using the CP correlation. Given the correlation's real and imaginary components, its magnitude squared is calculated. Peak Detector locates the peak correlation values in time. Frame Detect Controller locks-on to the received signal if the correlation peaks arrive with appropriate timing. Once locked-on to the signal, Phase Offset Estimator is started and estimates the phase offset. This estimate is provided to a smoothing filter which produces a smoothed estimate as its output. The IP cores which compose the design are shown in their hierarchy in the table below, and the top level is shown in Figures 8 and 9.

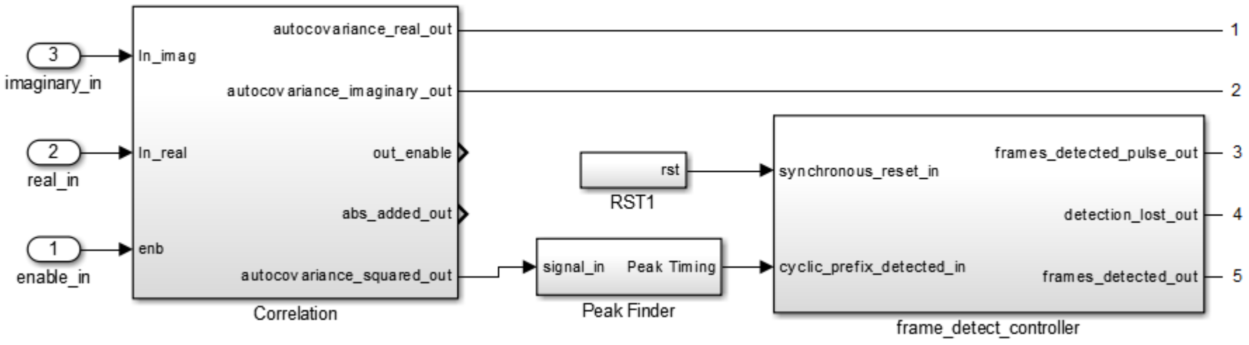


Figure 8: The Frame Detector design in System Generator.

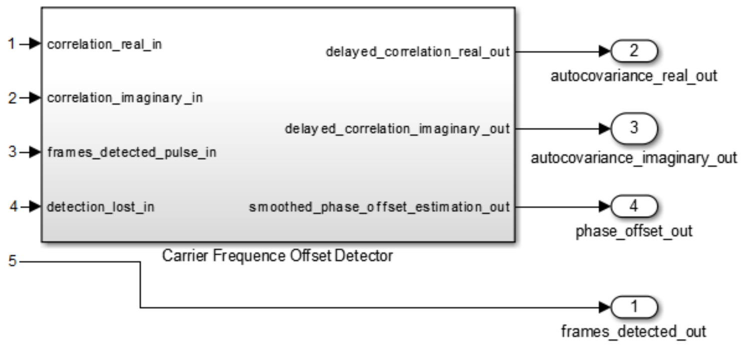

1
2
3
4
5
correlation_real_in
correlation_imaginary_in
frames_detected_pulse_in
detection_lost_in
delayed_correlation_real_out
delayed_correlation_imaginary_out
smoothed_phase_offset_estimation_out
Carrier Frequence Offset Detector
2
autocovariance_real_out
3
autocovariance_imaginary_out
4
phase_offset_out
1
frames_detected_out

Figure 9: The Carrier Frequency Offset Estimator design in System Generator.

Table 1: The project cores are broken down by hierarchy and software environment used to work with it.

IP Name	Design Environment
RefD_pilot_CP_CRC_FDCFOE	Vivado Project
Reference design	Analog Devices
Tx Core	System Generator Design
Rx Core	System Generator Design
Frame Detector and Carrier Frequency Offset Estimator	System Generator Design
Peak Finder	System Generator Design
Test Setup	System Generator Design
Frame Detect Controller	System Generator Design
frame_detect_controller	VHDL Design
Test Bench Simulation using GHDL	VHDL Design
arctan	System Generator Design
Full Arctangent	System Generator Design
Division using LUT and Goldschmidt Binomial	System Generator Design
Factor Inputs	System Generator Design
35 x 35 Multiplier	System Generator Design
Test Setup	System Generator Design
sub_arctan	System Generator Design
Test Setup	System Generator Design
AR_filter_half/Smoothing Filter	System Generator Design
AR Filter Controller	System Generator Design
AR_filter_controller_FSM	VHDL Design
AR Filter	System Generator Design
Test Setup	System Generator Design

3.1 Correlator

The autocorrelation of a random process produces a maximum value [18]. This is calculated by performing

$$u_n = \sum_{m=0}^{M-1} r_{n-m} r*_{n-N-m} \tag{4}$$

where r_n is a sample received at time n. This calculation runs continuously over time.

3.2 Peak Detector

Peak Detector is a module used to estimate the location of the maximum value given an array of values. The maximum correlation between the CP and tail data identifies the beginning of an OFDM frame. At the same time, knowing when the correlation peak values occur provides the best time to sample the correlation's real and imaginary components to estimate the phase offset as described in Section 4.

3.2.1 Peak Timing Calculation Strategies

Given an array of real numbers greater than or equal to 0, find the maximum value within the array. How do we know which maximum values correspond to the correlation peaks? There are two main solutions which approach the problem in a different way.

The simplest solution is to search the array sequentially. To do this:

Begin with the largest value being the first array value

Compare the last sample with the current sample

Choose which sample is the largest

Store its index

Repeat until the array ends

This method implicitly creates a threshold which increases until the maximum value, and then stays at the maximum value until the end of the array.

The second way is to estimate the correlation noise power using a method such as an auto-regressive moving-average (ARMA) filter, multiply it by a threshold, and then say that every sample above the threshold is "close enough" to the start of the frame.

The sequential peak finding method was chosen because it is simpler and provides better results. For example, it uses inexpensive hardware, and can be used with full precision even with large binary representations without causing timing failure.

3.2.2 Alternating Peak Finder

There is basic issue with the sequential peak finder. If the sequential peak finder algorithm is performed and the correlation peaks lie close to the ends of the array, then two peaks can both lie within the same array and the smaller one is dropped.

This can prevent the receiver from locking-on for several OFDM frames in a row as shown in Figure 10.

The array length is chosen to be $N + M$ samples long. The correlation peaks occur where the delays between the M samples from the tail data and the M samples from the cyclic prefix are precisely N clock cycles apart in time. This expected time between correlation peaks is $N + M$ clock cycles, though in practice the timing will vary. The DAC and ADC clocks operate at the same clock rate.

Each time the array computation is completed, it will produce the location of the peak value within the array. If an array size smaller than $N + M$ is used, then the peak detector will find a maximum more frequently than an expected maximum is being produced by the transmitter. If an array size larger than $N + M$ is used, then more than one correlation peak will be captured on average within the array. This larger case causes the receiver to miss a peak in the situations where two peaks lie within the array. Larger arrays also require more memory resources, and so are more expensive than smaller arrays. Taking these constraints into consideration, the most appropriate choice of array size seems to be $N + M$. In the example OFDM design, this is an array size of 1088 samples.

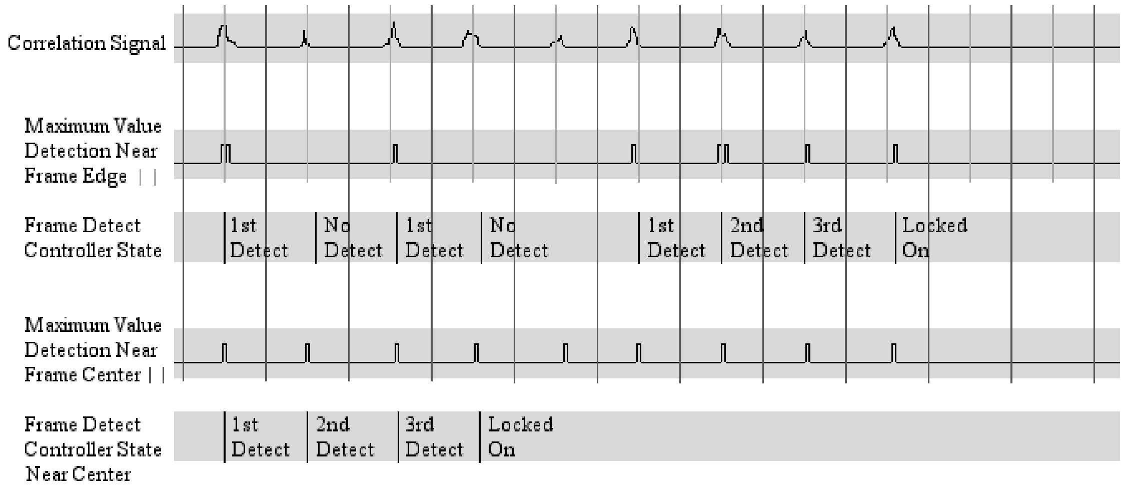

Correlation Signal
Maximum Value Detection Near Frame Edge | |
Frame Detect Controller State
1st Detect
No Detect
1st Detect
No Detect
1st Detect
2nd Detect
3rd Detect
Locked On
Maximum Value Detection Near Frame Center | |
Frame Detect Controller State Near Center
1st Detect
2nd Detect
3rd Detect
Locked On

Figure 10: The maximum value within each frame timing is represented by a pulse in time. If the correlation peaks are well within the frame, then the pulse timing is accurate.

The alternating peak finder is a simple way to get past the issue without needing feedback. The idea is to run two peak finding arrays in parallel. One array is offset from the other by half the period of the frame. If a peak is found "close to the center" of one array, then it is accepted as the location of a peak and a flag is raised corresponding to that point in time. If a peak is not found close to the center of the array, then the decision is that no detection is made and no flag is raised. This same rule applies to the other array. The flags obtained from both searches are combined using OR for the final output.

The term "close to the center" of an array means a peak detection on the range $[\frac{N+M}{4}, \frac{3(N+M)}{4})$, or halfway between the 0th index and the center value to halfway between the maximum index and the center value. The arrays are even-valued and the endpoints are chosen so that only one array covers every point. With no loss of generality, the left endpoint is selected while the right endpoint is ignored. The two array peak finder's parallel arrays are shown in Figure 11.

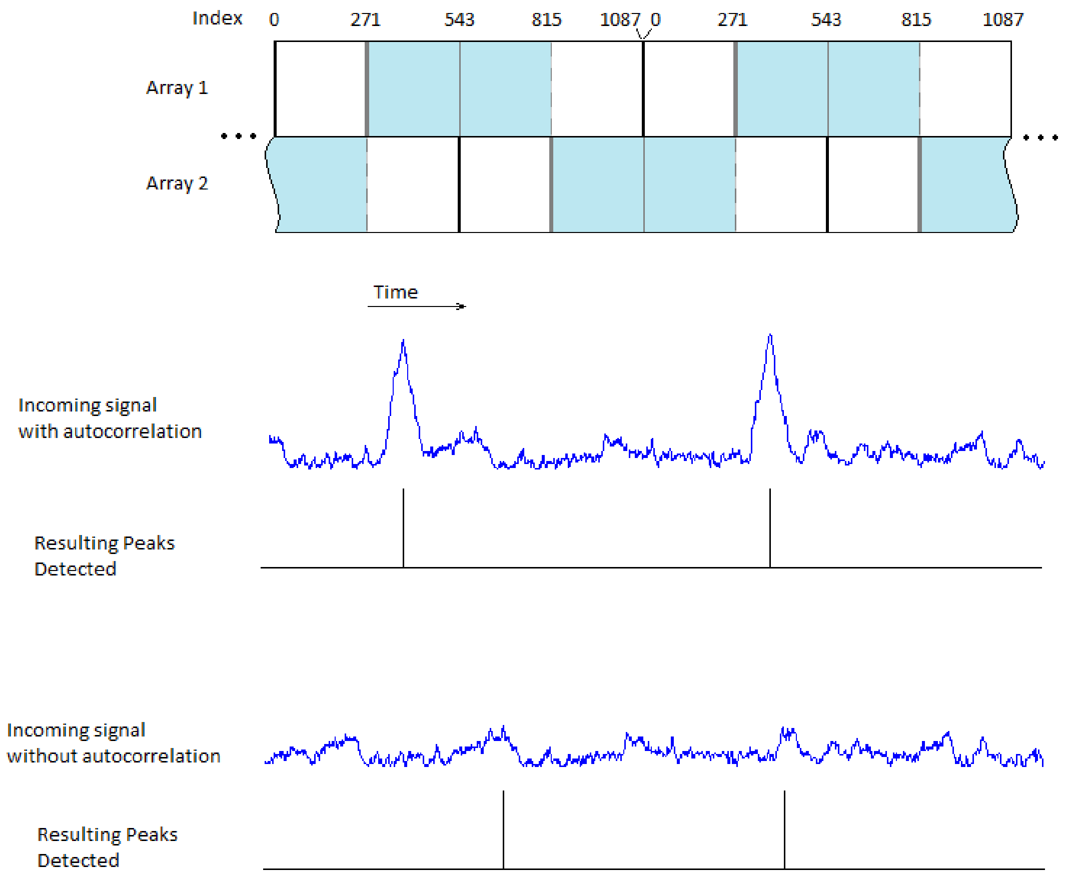


Figure 11: A peak detector which finds the peak balanced around the center point of the simulated correlation. The colored area represents close to the center of each array. The thick grey line represents the closed endpoint, and the dashed grey line represents the open endpoint. Two example signals and their corresponding peak finder outputs are also given.

A comparison was made between using the sum of the absolute values of the real and imaginary components of the correlation signal versus computing its squared magnitude. The result is that the sum of squared magnitudes loses at worst about 1.5 dB of peak height to correlation noise ratio versus the sum of squared magnitudes. In lab experiments on the FMCOMMS1, the performance using the sum of the real and imaginary components was too unreliable to lock onto the signal, while the sum of squared magnitudes is reliable.

3.2.3 Receiver Behaviour in Different Situations

When has an OFDM frame been detected? The correlation calculation must always run to wait for correlations, and Peak Detector must always calculate when the correlation peaks are received. There is necessarily a situation where the receiver can lock-on to noise. If a signal is being received, then the receiver is more likely to lock-on to it correctly the larger the SNR. It is important to know the probability of incorrectly locking-on to a noise signal and of successfully locking-on to a received signal based on the SNR.

Each correlation peak identified by the alternating peak finder is received with a delay between peaks according to a probability distribution. Two situations will reasonably estimate the receiver's performance. The first situation is the behaviour of the peak finder in the case of Gaussian white noise, which means there is no correlation. This situation can occur where there is no data being transmitted and received. The second situation is when the receiver receives an OFDM frame at a given SNR. The second case approaches the first case as the SNR of the received signal decreases.

The first step is to find the delay probability distributions for both cases. The purely-noise situation produces a delay probability distribution according to the histogram shown in Figure 12. This curve represents the probability distribution of the timing of a received correlation peak obtained using the alternating peak finder when the received signal does not correlate with itself. The Frame Detect Controller's acceptance window is shown and describes how likely it is to take a lock-on step due to noise. Frame Detect Controller performs a lock-on step when a correlation peak is received within the acceptance window. Once locked-on, Frame Detect Controller will perform a fall-out-of-lock step when a correlation peak is not received within the acceptance window.

The noise-only simulation is performed within the System Generator model "Peak Finder", and the received OFDM frame simulation is performed using the System

Generator model "Frame Detector and Carrier Frequency Offset Estimator". The peak times from the peak detector are captured as a workspace object in MATLAB and the differences between the peaks represents the number of discrete time steps between each.

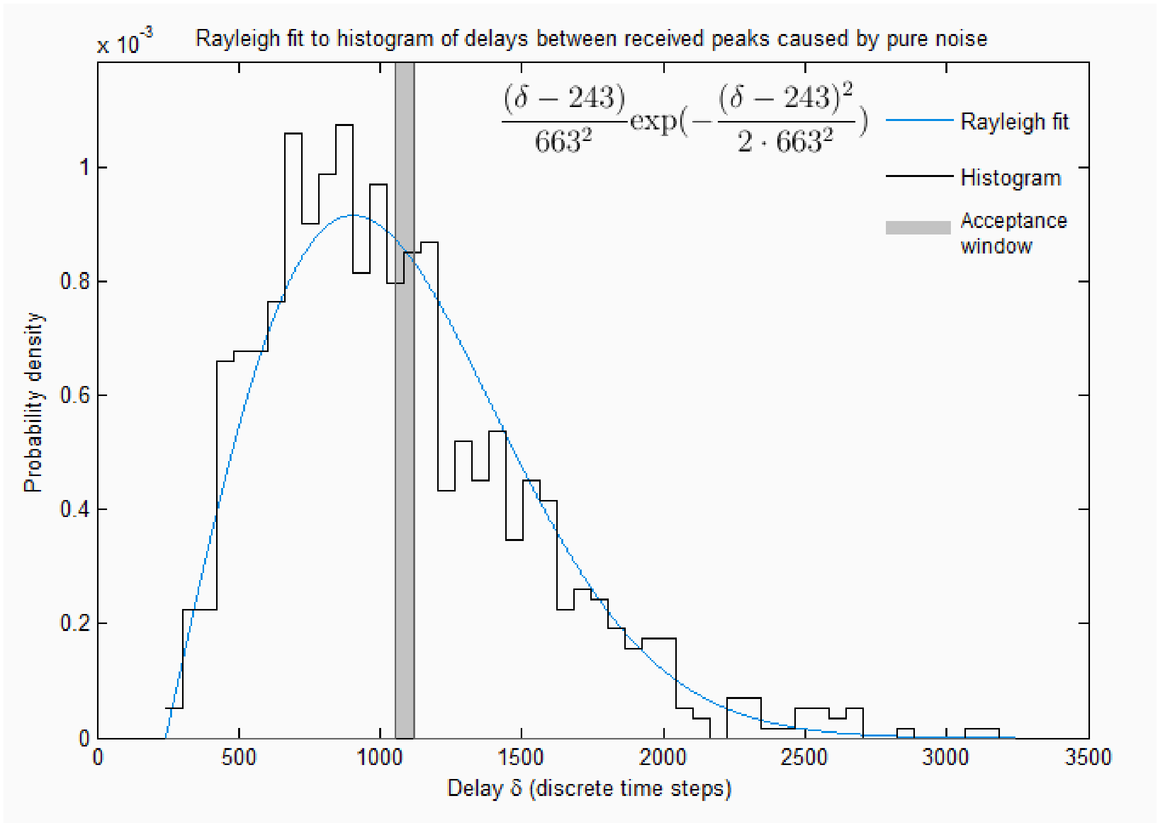


Figure 12: The probability that a peak will be calculated at a particular array index by the alternating peak finder for Gaussian white noise.

A function is fit to the calculated distribution to account for potential variance between different simulations. Two common distributions seem to fit the simulation data; a Rayleigh distribution and a Weibull distribution. The parameters for each function were selected by search using mean squared error (MSE) between the simulation data and the cumulative distribution function (CDF) of each model. The Weibull CDF produces a MSE of 0.298 from

$$CDF_{Rayleigh} = \begin{cases} 0 & \delta < 243 \\ 1 - \exp(-\frac{(\delta-243)^2}{2\cdot 663^2}) & \delta \geq 243 \end{cases}, \tag{5}$$

versus the Rayleigh CDF's MSE of 0.373 from

$$CDF_{Weibull} = \begin{cases} 0 & \delta < 243 \\ 1 - exp(-(\frac{(\delta-243)}{938})^{1.927}) & \delta \geq 243 \end{cases} . \tag{6}$$

For analysis the Rayleigh distribution is used because it provides a similar level of fit while using only a single parameter versus the two parameters required by the Weibull distribution.

When Frame Detect Controller locks-on by one step in Gaussian noise, the next peak will be received at a delay with a probability as shown in Figure 12. If the peak is received before the acceptance window, then another peak could be received before the acceptance window again, within the acceptance window, or after the acceptance window. This has the effect of shifting the delay probability distribution with respect to the acceptance window for the next received pulse. This will be discussed in Section 3.4 in the context of the Markov chain analysis.

The probability of detecting a received OFDM frame correctly depends on SNR. At low SNRs the probability of locking-on is the same as the noise-only situation. As the SNR increases the probability of locking-on increases to almost certainly. This is shown in the delay probability distributions in Figure 13 where more probability lies within the acceptance window at higher SNRs. A peak which occurs in the noise around a delay of 2000 which fades near the pure noise case. This is probably the alternating peak finder identifying the second OFDM frame correlation peak around clock cycle 2176 after missing the first one around 1088.

Figure 15 shows the same simulations located near the acceptance window. The vertical axes show how the probability peak height decreases and width widens at each drop in SNR. The probability peak is not modeled effectively by a Gaussian distribution. Each correlation peak's expected time is the center of the acceptance window. This is different from the peak timing distribution which acts independent of when a pulse is received as described in Figure 14. This detail is used for the Markov chain analysis.

Only the peak timing probability distribution for the initial received peak is known from simulation. How much does a received OFDM frame increase the probability of the receiver correctly locking-on to it? The probability of landing in the acceptance window due to the OFDM frame's correlation versus the non-correlation noise must be estimated.

The amount of probability is measured before, within, and after the acceptance window using the simulation data over the same ranges. Empirical fits, made for each

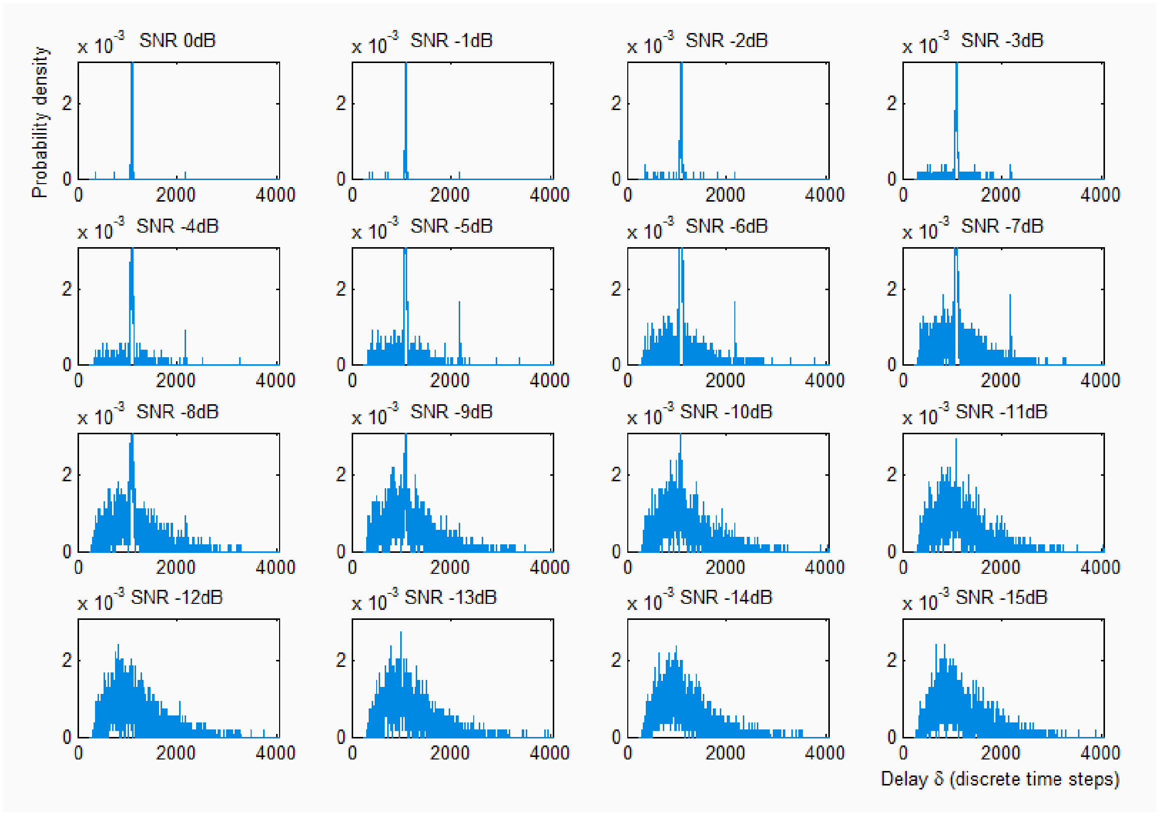


Figure 13: The probability densities of delay between correlation peaks at different SNRs.

case, are shown in Figure 16, and are given by the equations

$$P_{\text{before}} = 0.27\text{erfc}(0.352(\text{SNR} + 6.375)), \tag{7}$$

$$P_{\text{within}} = 0.5\text{erf}(0.3613(\text{SNR} + 6.48)) + 0.5, \text{ and} \tag{8}$$

$$P_{\text{after}} = 0.2\text{erfc}(0.3802(\text{SNR} + 6.3)). \tag{9}$$

Notice that the fit to the mass within the window given by Equation 8 is weighted more towards simulations at higher SNRs. This allows an approximation to the peak delay probability lying within the acceptance window. The idea is that when the SNRs are high, the empirical fit will be almost entirely due to the probability peak. As the SNR decreases the value of the probability peak approaches zero. A curve fit can be made to the noise before and after the acceptance window. The probability due to noise within the acceptance window can be estimated, and the probability due to the noise plus OFDM frame is given by the simulation data. Comparing the probability peak probability due to the empirical fit versus an approximate noise fit

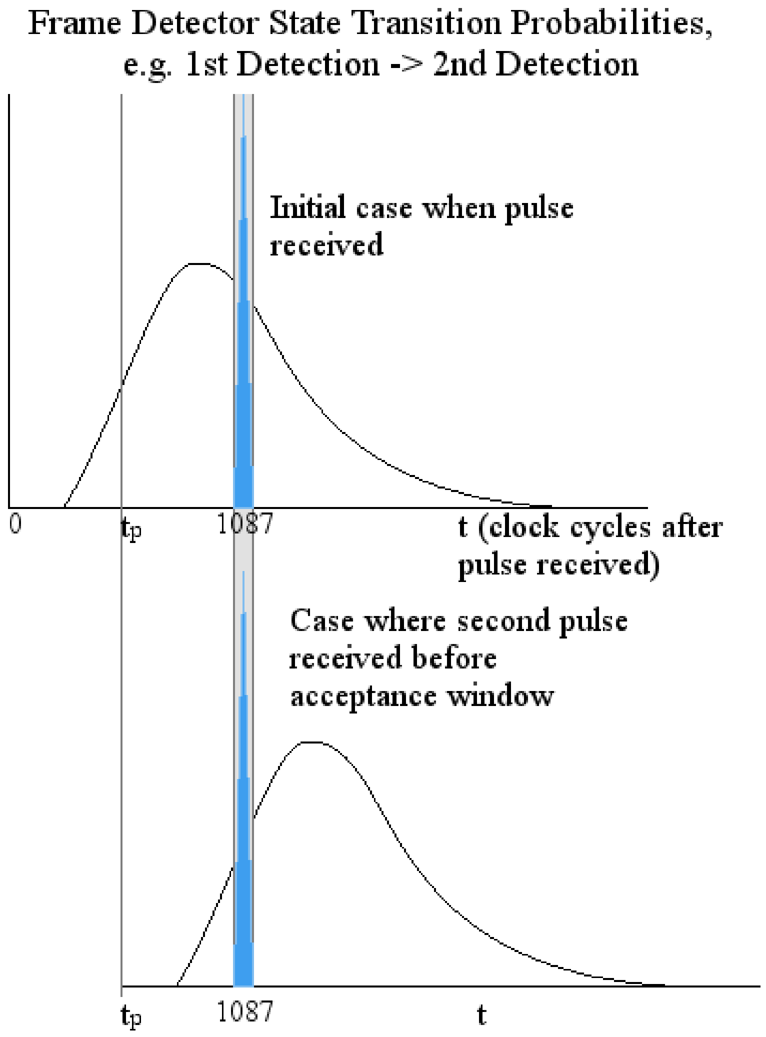

Frame Detector State Transition Probabilities,
e.g. 1st Detection -> 2nd Detection
Initial case when pulse received
0
tp
1087
t (clock cycles after pulse received)
Case where second pulse received before acceptance window
tp
1087
t

Figure 14: The peak timing distribution caused by noise behaves the same independent of when the most recent pulse has been received. The peak timing distribution caused by a received OFDM frame always lies within the acceptance window. The acceptance window timing is fixed by Frame Detect Controller based on the last lock-on step timing.

at 10 dB SNR shows a difference of about 10%.

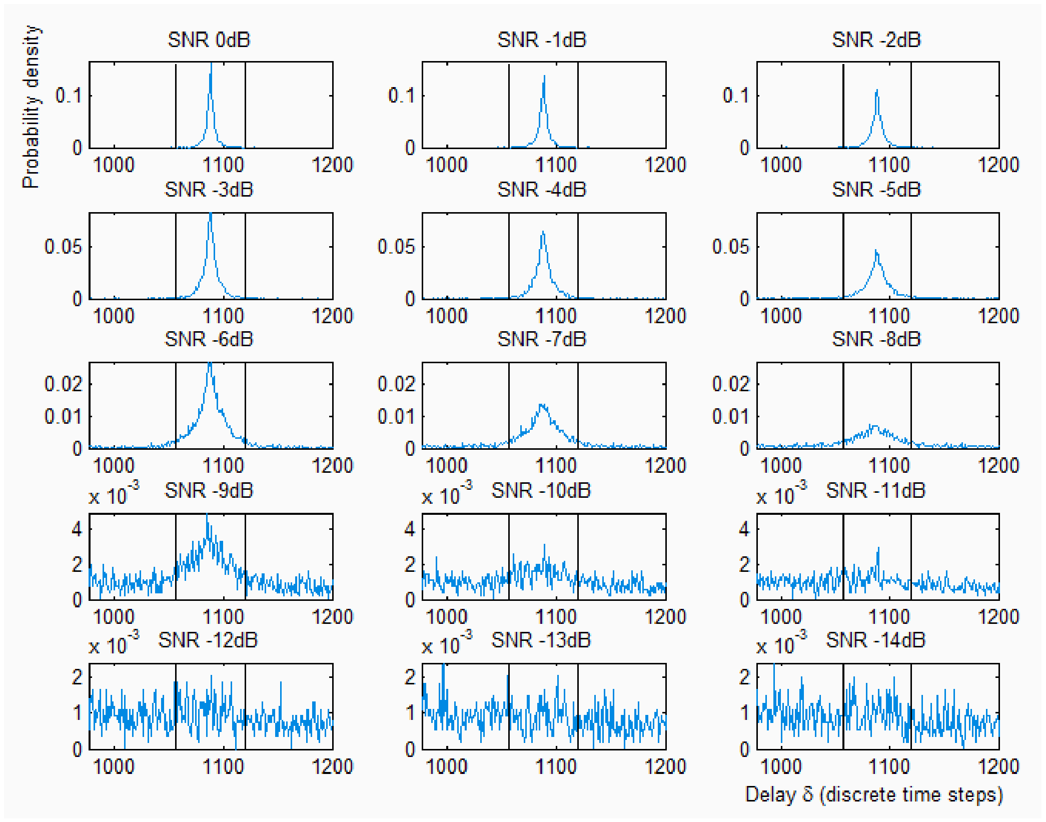


Figure 15: The probability densities of delay between correlation peaks at different SNRs near the acceptance window.

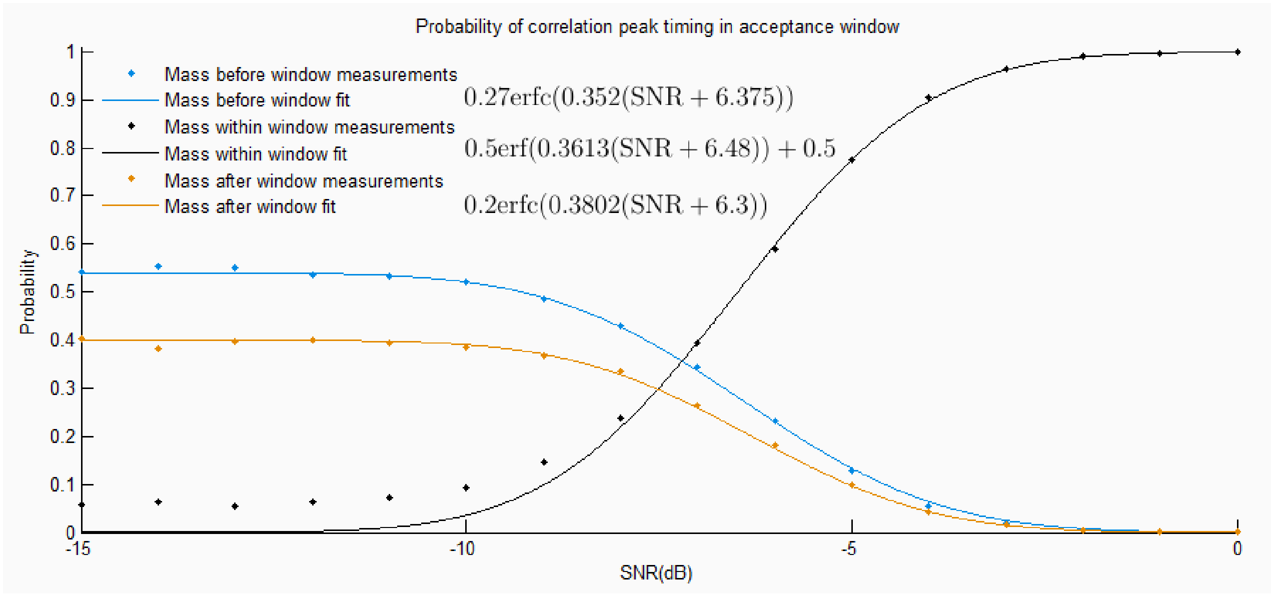


Figure 16: The probability measured before, during, and after the acceptance window, along with some fits to the data. Note that the mass within the window fit is set to fit only the mass due to the peak lying within the acceptance window.

Alternating Peak Finder Implementation The peak finder is a simple circuit consisting of a register which compares the currently stored maximum value for the duration of 1088 clock cycles. When a larger value is found, its index replaces the currently stored index. In the alternating peak finder there are two sequential peak finders. Once the full array of one peak finder has been checked, if the index lies close to the center of the array, a counter is loaded with the stored index of the peak value and it begins a countdown. When the countdown completes, the output of '0' pulses to '1' for a single clock cycle. This process repeats indefinitely. The only difference between the first peak finder and the second one is that the second peak finder is started 544 clock cycles later. The results of the output are combined using a single OR gate, and then provided as the result. The latency of the circuit is 1089 clock cycles.

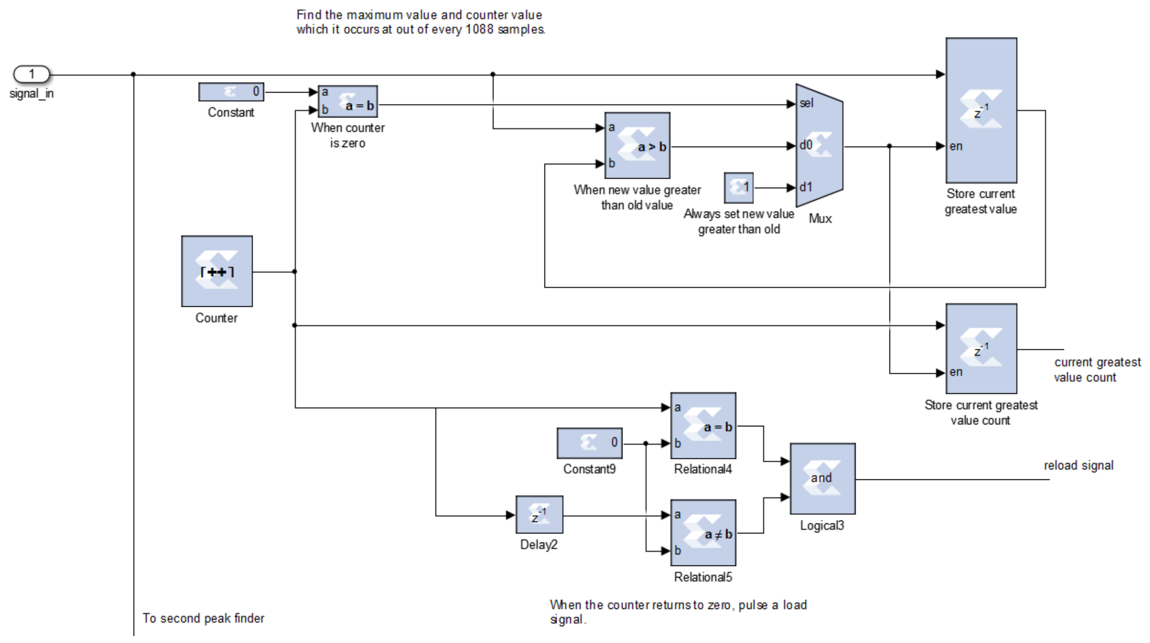

Find the maximum value and counter value which it occurs at out of every 1088 samples.
1
signal_in
Constant
When counter is zero
a = b
When new value greater than old value
a > b
Always set new value greater than old
Mux
sel
d0
d1
Store current greatest value
en
Counter
Store current greatest value count
current greatest value count
Constant9
Relational4
Relational5
a ≠ b
Delay2
and
Logical3
reload signal
To second peak finder
When the counter returns to zero, pulse a load signal.

Figure 17: One sequential peak finder component which forms part of an alternating peak finder. It includes one register holding the maximum found value and another register holding the count of the maximum found value. Both of these registers are updated when the current signal value is greater than the previously stored value. When the array has been searched, a counter loads the index of the largest value found in the array.

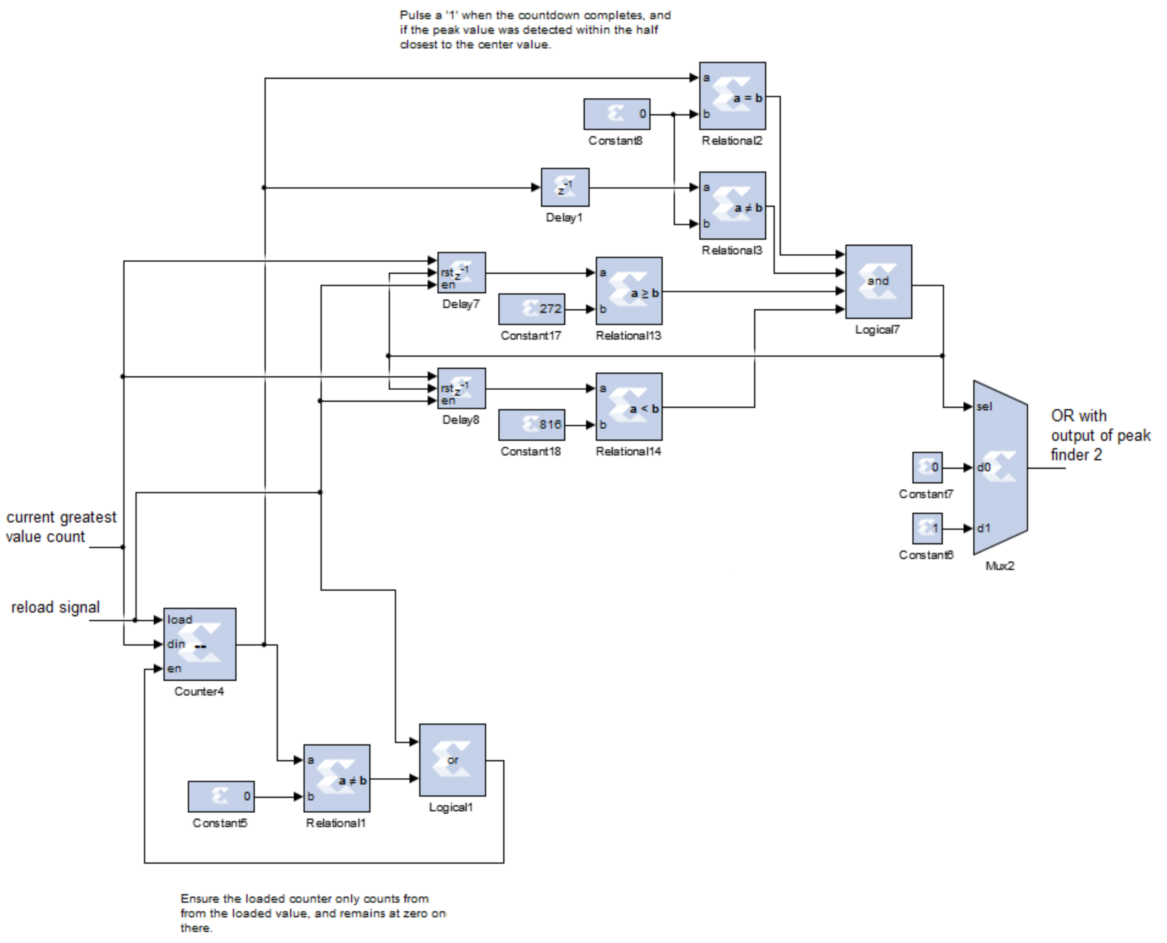

Pulse a '1' when the countdown completes, and if the peak value was detected within the half closest to the center value.
Constant8
Relational2
Delay1
Relational3
Delay7
Constant17
Relational13
Logical7
Delay8
Constant18
Relational14
OR with output of peak finder 2
Constant7
Constant6
Mux2
current greatest value count
reload signal
Counter4
Constant5
Relational1
Logical1
Ensure the loaded counter only counts from from the loaded value, and remains at zero on there.

Figure 18: The logic which ensures only the detected peaks close to the center of the first sequential peak finder are produced as outputs. Loading the index of the largest value in the array, it is compared to the brackets and a pulse is produced if it lies closer to the center index 544. The same operation is performed for the second peak finder, and the OR of both results is the alternating peak finder's output.

Peak Finder Simulation and Performance The following test uses Golden Model. Peak Finder is placed immediately after the simulated AWGN channel and produces a pulse of '1' with a constant latency after a peak has been found. An example simulation result is shown in Figure 19. Early in time in Figure 19 there is a noise signal, but two peaks within the noise are detected by Peak Finder. Once the OFDM frames are received by the reception chain, Peak Finder identifies the correlation peaks only.

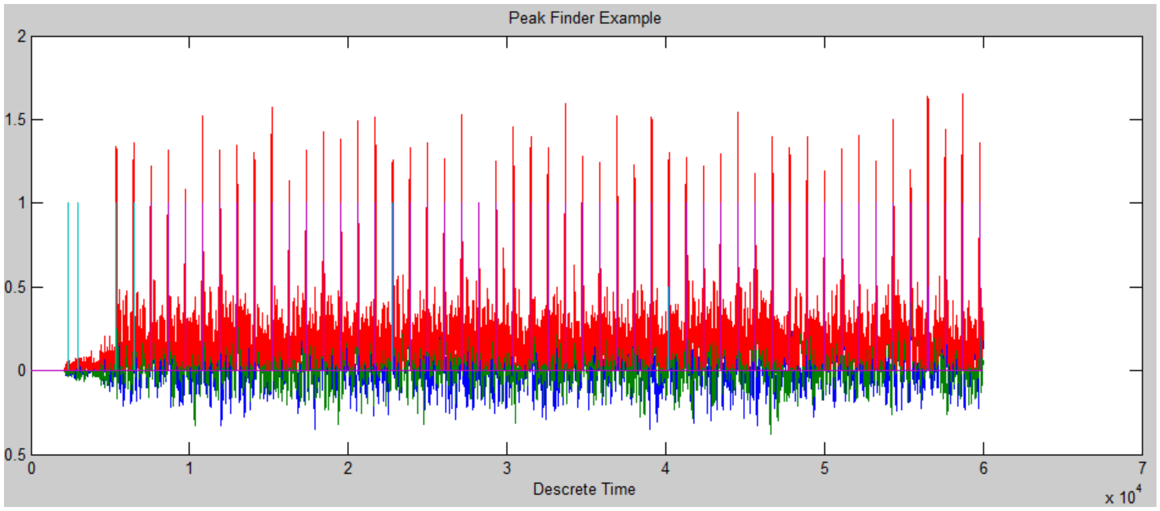


Figure 19: An overall picture of the Golden Model simulation with the peak finder detecting correlation peaks. The peak finder results are shown in teal, and correlation peak in red. The rest of the signals align with the peaks and are hard to distinguish in this image.

3.3 Frame Detect Controller

Frame Detect Controller is a finite state machine (FSM) with datapath (FSMD) which determines whether or not an OFDM frame of data has been received by entering a ”locked-on” state when the correlation peak has been detected enough times. Each new detection occurs within a window of time after the previously received detection. When the correlation peak has not been detected within this window of time for some number of consecutive time windows, the FSM ”falls out of lock” and there is considered to be no OFDM frame present. A **datapath** is a network of registers, arithmetic, and logic through which a calculation is computed [19]. A FSMD is an architecture where a FSM controls how the datapath behaves [20].

The top level of Frame Detect Controller is shown in Figure 20. Frame Detect Controller's input is a one-bit value of '0' when there is no cyclic prefix detected, and '1' when there is a detection. It has the three one-bit output signals:

frames_detected which is '0' when no detection is available and '1' when a detection is available

frames_detected_pulse which pulses from '0' to '1' then back to '0' for one clock cycle when a frame has been detected

detection_lost which is '1' whenever the Frame Detect Controller is in the ”no detection” state, and '0' otherwise

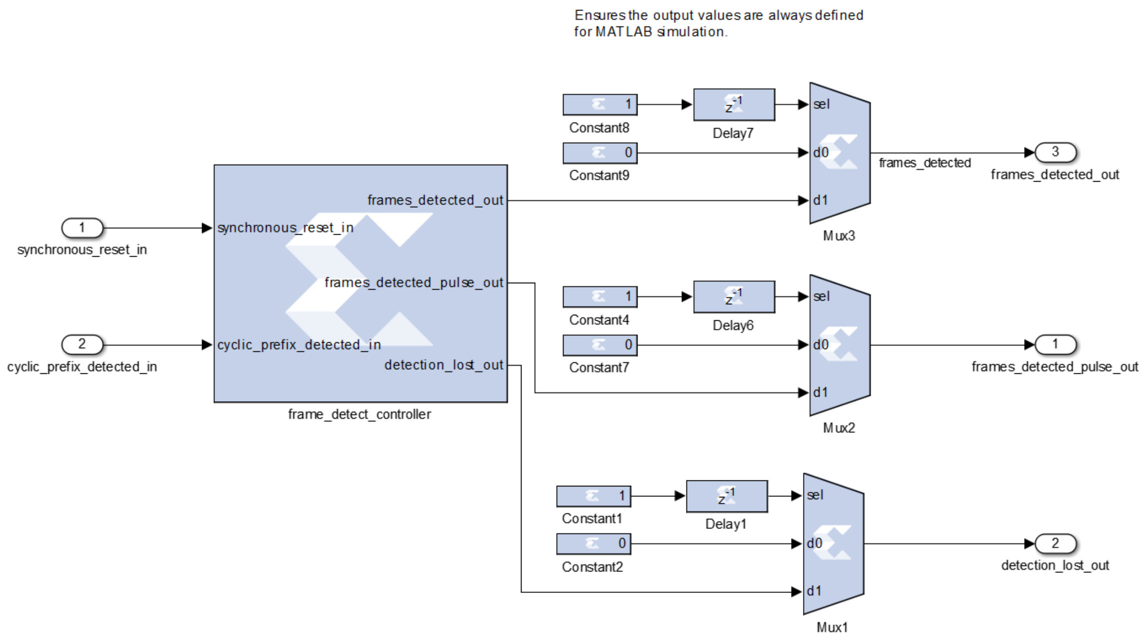


Figure 20: The frame_detect_controller is implemented as a System Generator black box. Its outputs are initialized to get past a simulation issue causing the first produced outputs to be not defined.

For the example design, Frame Detect Controller begins in the ”no detection” state and waits for three locking-on states to complete before entering the ”locked-on” state, and waits three states of falling-out-of-lock before returning to the ”no detection” state. The number of locking-on states and falling-out-of-lock states can be modified to meet the requirements of a particular communications design.

For an OFDM frame consisting of $M + N = 1088$ samples, a received ”correlation peak detected” signal begins a countdown timer which triggers the FSMD **check_pulse_received_within_variance** when the countdown is complete. This second FSMD begins another countdown time which acts as a detection window. All countdown times must be an integer number of clock cycles, and are positioned to balance the window so that an equal number of clock cycles before and after the expected time will be accepted. This means the acceptance window will have an odd number of clock cycles. While the cyclic prefix size is $M = 64$, the window size is chosen to be 63 in order to ensure that any particular correlation peak will correspond to only one copy of the embedded pilot per OFDM frame. This is important because each received correlation peak should align with only a single copy of the embedded pilot.

The Frame Detect Controller block is composed of purely VHDL entities, and its hierarchy is:

```
frame_detect_controller
    frame_detect_controller_FSM
    check_pulse_received_within_variance
        frame_detect_variance_FSM
        frame_detect_countdown_timer
```

3.3.1 Frame Detect Controller Core

The top level module shown in Figure 20 and is called frame_detect_controller and combines the frame_detect_controller_FSM and check_pulse_received_within_variance modules. There are no generics in the entity description of frame_detect_controller, but it does require constants to be set within the core in order to set up the correct timing of N clock cycles between each expected correlation peak. Constants also need to be set to describe the width of the acceptable correlation offset on the range $[-w, w]$. The "ACCEPTABLE_VARIANCE_LENGTH" is the number of clock cycles in the window of acceptable times and is referred to in the VHDL code by M, not to be confused with the size of the cyclic prefix M. M is a positive odd integer value greater than or equal to 3. The window size can also be considered as a range of values $[-w, w]$ of length M, with $w = ((\mathrm{M}-1)/2)$. The Golden Model design example uses M=63 and $w = 31$. There are several steps to modify the window size and OFDM frame length throughout the VHDL hierarchy. The modifications performed within frame_detect_controller are on the generics:

```
constant OFDM_FRAME_LENGTH : POSITIVE := 1088;
constant ACCEPTABLE_VARIANCE_LENGTH : POSITIVE := 63;
```

In the example design the detection window is timed so that it is a window beginning 31 clock cycles before the 1088th clock cycle, and ending at 31 clock cycles after the 1088th clock cycle. If a correlation peak is detected within this window then the window signals the Frame Detect Controller FSM to move to the next lock-on state. The size and timing of the detection window can be modified depending on the design requirements.

Frame Detect Controller is tested in the following ways to ensure correct operation.

TIMING Tests The same idea lies behind each of the PULSE_TIMING and MISS_TIMING tests for Frame Detect Controller. In each of these cases, a series of pulses from '0' to '1' are provided at the input to Frame Detect Controller at a particular timing. The purpose is to ensure that when the Frame Detect Controller's frame_detect_controller_FSM has reached an appropriate state. The FSM will fall-out-of-lock when pulses are not received within the window, and it will say in lock when pulses are received within the window. When a pulse has been received at the expected timing minus an integer number of clock cycles $-w_{test}$ the state is checked. This is repeated for each possible input pulse timing until w_{test}. The state can be checked by measuring the width of the frames_detected signal, or by counting the number of times cyclic_prefix_is_synchronized pulses from '0' to '1'. Choose $w_{test} > w$ so that the test window times $[-w_{test}, w_{test}]$ contain the accepted timing window $[-w, w]$ plus extra time to ensure that only pulses received within the correct window are accepted. Examples of this are shown in Figures 21 and 22.

Figure 21 shows the results of the FIRST_PULSE_TIMING test.

Figure 22 shows the results of the THIRD_MISS_TIMING test.

Figure 23 uses data from a System Generator simulation of Golden_Model_5, which provides "flag" data, as input to frame_detect_controller in VHDL. This image shows the results for GOLDEN_MODEL_5_DATA, and the output "frames_detected" is expected to remain '1' once locked on. This test effectively shows the delay required to determine three correlations received within the correct timing.

BASIC OPERATION BASIC_OPERATION tests simple timing using single pulses at the input. This test is to see the basic operation of the FSM, and only looks into whether or not the FSM makes the correct decision given an ideal input.

SHIFTING TIMING SHIFTING_TIMING_TEST tests to see if there is a shift in the timing of the detection based on how the FSM operates. There was a bug where the timing of the next expected detection shifts based on the FSM, and it has been dealt with using this test.

INPUT COMBINATIONS INPUT_COMBINATIONS tests different possible input shapes by serializing a binary number into the input signal to frame_detect_controller. It keeps the number entering constant during each run of the test, and is set up to run the test over all numbers from 0 to NUMBER_BITS_SIMULATING_CP_COMBINATIONS-1. A bit of automated checking is performed, so if the frame detect output isn't '1' at the testing time an error is produced by the simulator.

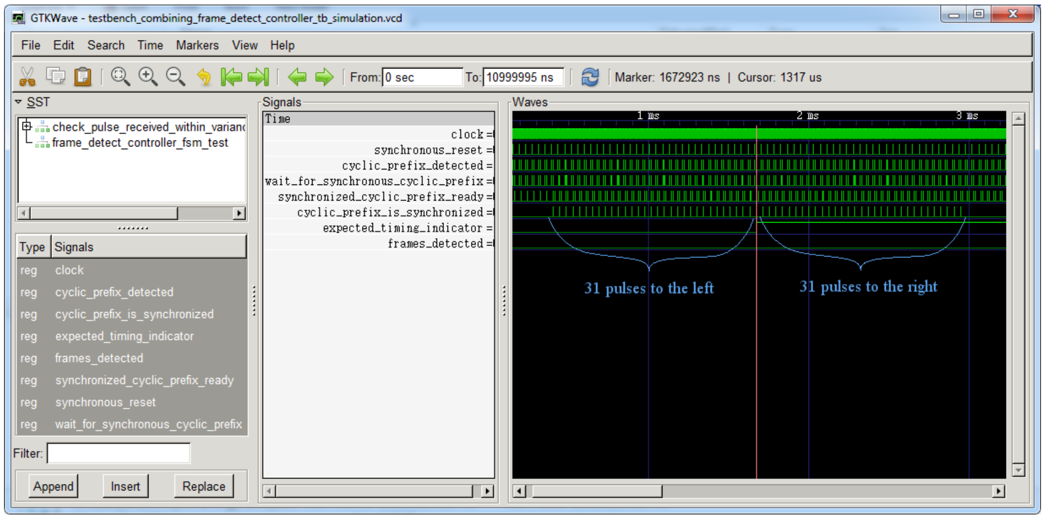

GTKWave - testbench_combining_frame_detect_controller_tb_simulation.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 10999995 ns
Marker: 1672923 ns | Cursor: 1317 us
SST
check_pulse_received_within_varianc
frame_detect_controller_fsm_test
Type Signals
reg clock
reg cyclic_prefix_detected
reg cyclic_prefix_is_synchronized
reg expected_timing_indicator
reg frames_detected
reg synchronized_cyclic_prefix_ready
reg synchronous_reset
reg wait_for_synchronous_cyclic_prefix
Filter:
Append Insert Replace
Signals
Time
clock
synchronous_reset
cyclic_prefix_detected
wait_for_synchronous_cyclic_prefix
synchronized_cyclic_prefix_ready
cyclic_prefix_is_synchronized
expected_timing_indicator
frames_detected
Waves
1 ms
2 ms
3 ms
31 pulses to the left
31 pulses to the right

Figure 21: The results of the FIRST_PULSE_TIMING test. For $w = 31$, count the 31 pulses of cyclic_prefix_is_synchronized to the right and left of the expected timing. The expected timing is shown using expected_timing_indicator.

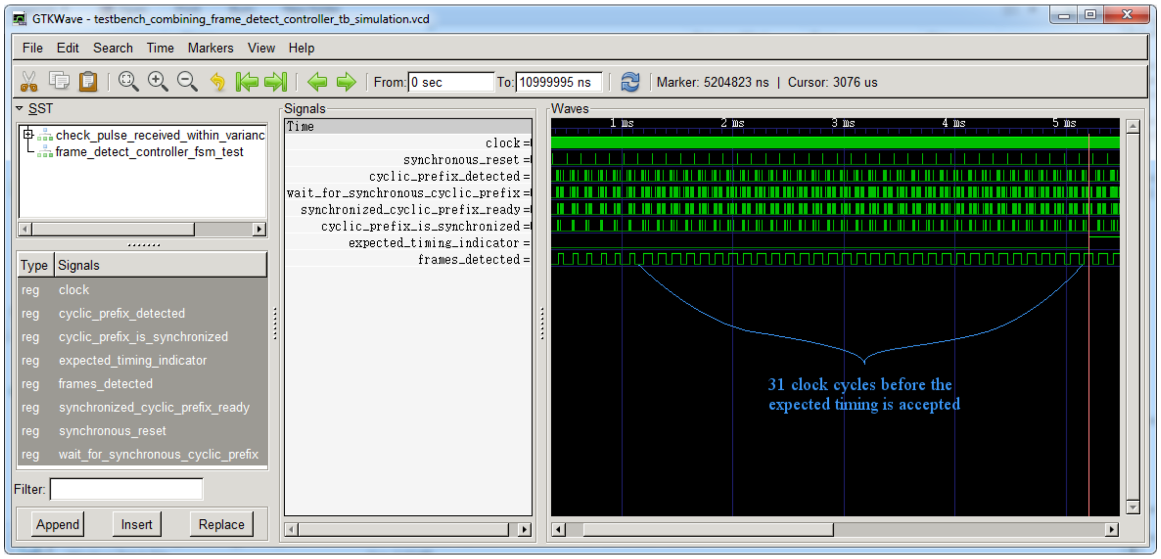

GTKWave - testbench_combining_frame_detect_controller_tb_simulation.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 10999995 ns
Marker: 5204823 ns | Cursor: 3076 us
SST
check_pulse_received_within_varianc
frame_detect_controller_fsm_test
Type Signals
reg clock
reg cyclic_prefix_detected
reg cyclic_prefix_is_synchronized
reg expected_timing_indicator
reg frames_detected
reg synchronized_cyclic_prefix_ready
reg synchronous_reset
reg wait_for_synchronous_cyclic_prefix
Filter:
Append Insert Replace
Signals
Time
clock
synchronous_reset
cyclic_prefix_detected
wait_for_synchronous_cyclic_prefix
synchronized_cyclic_prefix_ready
cyclic_prefix_is_synchronized
expected_timing_indicator
frames_detected
Waves
1 ms
2 ms
3 ms
4 ms
5 ms
31 clock cycles before the expected timing is accepted

Figure 22: THIRD_MISS_TIMING test for the $-w$ side. In this case it is easier to count the delay offsets where the controller stays locked-on based on frames_detected, rather than the number of pulses in cyclic_prefix_is_synchronized.

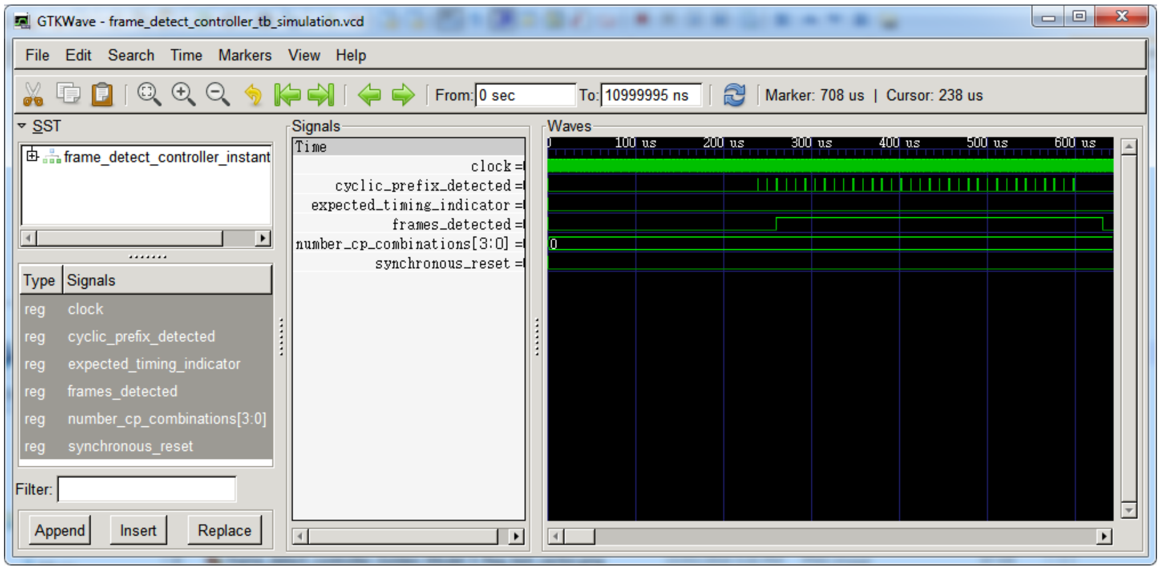

GTKWave - frame_detect_controller_tb_simulation.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 10999995 ns
Marker: 708 us | Cursor: 238 us
SST
frame_detect_controller_instant
Type Signals
reg clock
reg cyclic_prefix_detected
reg expected_timing_indicator
reg frames_detected
reg number_cp_combinations[3:0]
reg synchronous_reset
Filter:
Append
Insert
Replace
Signals
Time
clock =
cyclic_prefix_detected =
expected_timing_indicator =
frames_detected =
number_cp_combinations[3:0] =
synchronous_reset =
Waves
100 us
200 us
300 us
400 us
500 us
600 us
0

Figure 23: GOLDEN_MODEL_5_DATA uses simulation data from Golden_Model_5. The result in frames_detected is expected to always be "1" after the third pulse is received, and until the FSM falls out of lock about 3264 clock cycles (about 33 us) later.

3.3.2 Frame Detect Controller FSM

The Frame Detect Controller FSM uses Check Pulse Received within Variance as a datapath to measure if a received pulse lies within the expected time with respect to a previous pulse.

Beginning in the state no_detection, the FSM waits until it receives a detection (a '1' at the input). Once it has, it waits until it receives a detection within the acceptance window determined by check_pulse_received_within_variance. If a detection has been received within the window, then it begins waiting for a detection within a new window at

$$n + (N + M - 1) + [-w, w], \tag{10}$$

where n is the current time and $N+M-1$ is the expected time of the next correlation peak. In this particular design, it does this three times and then enters a steady state as shown in Figure 24.

Obtaining the correct delay between the expected time and the acceptance window needs to be balanced within frame_detect_variance_controller_FSM to ensure the expected time always lies in the center of the buffer calculated by check_received_within_variance. This is explained in the header of frame_detect_variance_controller_FSM.vhd, and is highlighted here. The delay states run from first_detection_hold_1 to first_detection_hold_((M+3)/2), second_detection_hold_1 to second_detection_hold_((M-

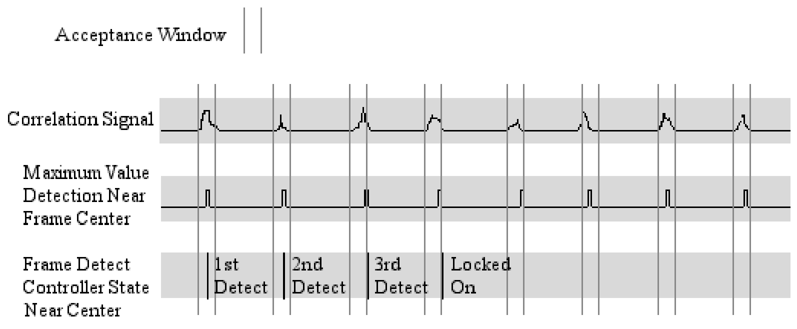

Acceptance Window
Correlation Signal
Maximum Value Detection Near Frame Center
Frame Detect Controller State Near Center
1st Detect
2nd Detect
3rd Detect
Locked On

Figure 24: An illustration of Frame Detect Controller FSM advancing from the state of no detection to being locked-on.

2)/2), third_detection_hold_1 to third_detection_hold_((M-2)/2), and new_detection_hold_1 to new_detection_hold_((M-2)/2). For example, if M= 63 and $w = 31$, the end states are first_detection_hold_33, second_detection_hold_31, third_detection_hold_31, and new_detection_hold_31. The counter itself runs a fixed number of clock cycles. Since the acceptable variance occurs over M clock cycles, pulses can be received after these 1088 samples, and so the counter is smaller than 1088. This means the counter must start its countdown later than the time an initial pulse is received for the transition out of no_detection, second_detection, third_detection, and new_detection.

When a pulse has been received within acceptable timing, the FSM must ensure that the counter within check_pulse_received_within_variance doesn't begin a countdown before the appropriate time. To do this, frame_detect_controller_FSM resets check_pulse_received_within_variance immediately after no_detection, second_detection, third_detection, and new_detection. This is necessary because the signal used to start the countdown is the same as the one used to determine if a pulse has been received within the appropriate window. A pulse may be sent into check_pulse_received_within_variance the clock cycle immediately after a clock cycle received within variance is detected. This is because it takes one clock cycle to decide if the previous value was indeed within the variance. The reset is a simple solution found for the issue of timing inconsistencies when an input remains '1' for more than one clock cycle.

3.3.3 Check Pulse Received Within Variance

The FSMD check_pulse_received_within_variance combines frame_detect_variance_FSM with the counter frame_detect_countdown_timer to detect whether or not a "1" has been received within a window of $[-w, w]$ clock cycles around the expected time.

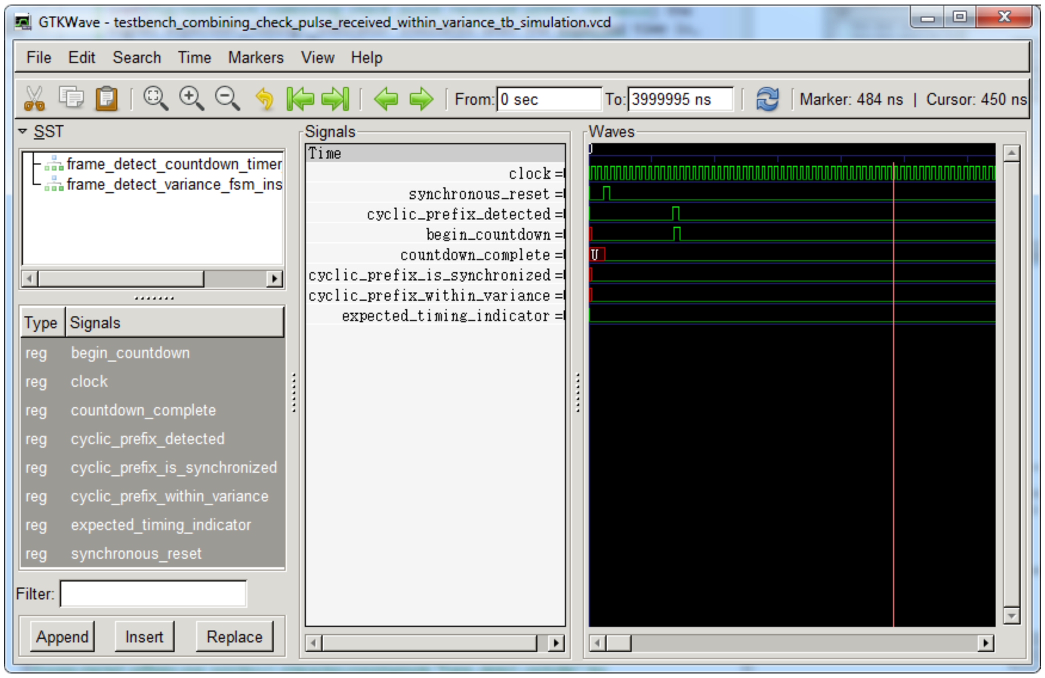
No changes must be made within check_pulse_received_within_variance to adjust the countdown time or the window size since for a particular design since its length is adjusted using generics.

In Figure 21 expected_timing_indicator indicates when the expected time is. The figure shows different cases where a received pulse in time occurs with a known delay around the expected time. Each step away from the expected time represents a test using a delay different by 1 clock cycle. The tests to the left have a delay less than the expected delay, and the tests to the right have more. The test will be correct if there are 31 pulses of cyclic_prefix_within_variance on either side of the expected time. More signals are shown in this figure than are used as the inputs and outputs of check_pulse_received_within_variance for testing purposes.

3.3.4 Frame Detect Variance FSM

The FSM frame_detect_variance_FSM controls a countdown timer and a delay during which a received input of '1' causes the "cyclic_prefix_is_synchronized_out" signal to pulse '1' at the same time as "synchronized_cyclic_prefix_ready_out". If the countdown and window complete without receiving an input, then the "synchronized_cyclic_prefix_ready_out" signal pulses, but "cyclic_prefix_is_synchronized_out" remains '0'. To modify the size of the window, change the states of the FSM so that they correspond to countdown_complete_1 until countdown_complete_M. These instructions are mentioned in the header of frame_detect_variance_FSM.vhd.

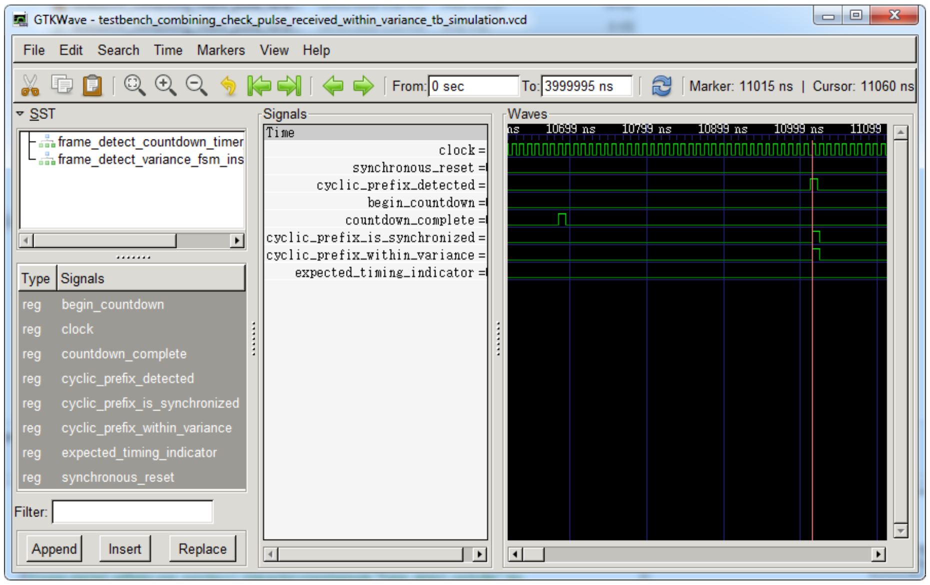
The module frame_detect_variance_FSM is tested alongside frame_countdown_timer to see the timing of the FSM with respect to the circuit it is intended to work with. Figure 25 shows the core being reset and a new pulse being received. When a second pulse is received within the expected time window, the result is similar to that shown in Figure 26. When a second pulse is not received within the expected time window, the result is similar to that shown in Figure 27.


GTKWave - testbench_combining_check_pulse_received_within_variance_tb_simulation.vcd

From: 0 sec To: 3999995 ns | Marker: 484 ns | Cursor: 450 ns

File Edit Search Time Markers View Help
SST
frame_detect_countdown_timer
frame_detect_variance_fsm_ins
Type Signals
reg begin_countdown
reg clock
reg countdown_complete
reg cyclic_prefix_detected
reg cyclic_prefix_is_synchronized
reg cyclic_prefix_within_variance
reg expected_timing_indicator
reg synchronous_reset
Filter:
Append Insert Replace
Signals
Time
clock =
synchronous_reset =
cyclic_prefix_detected =
begin_countdown =
countdown_complete =
cyclic_prefix_is_synchronized =
cyclic_prefix_within_variance =
expected_timing_indicator =
Waves
U

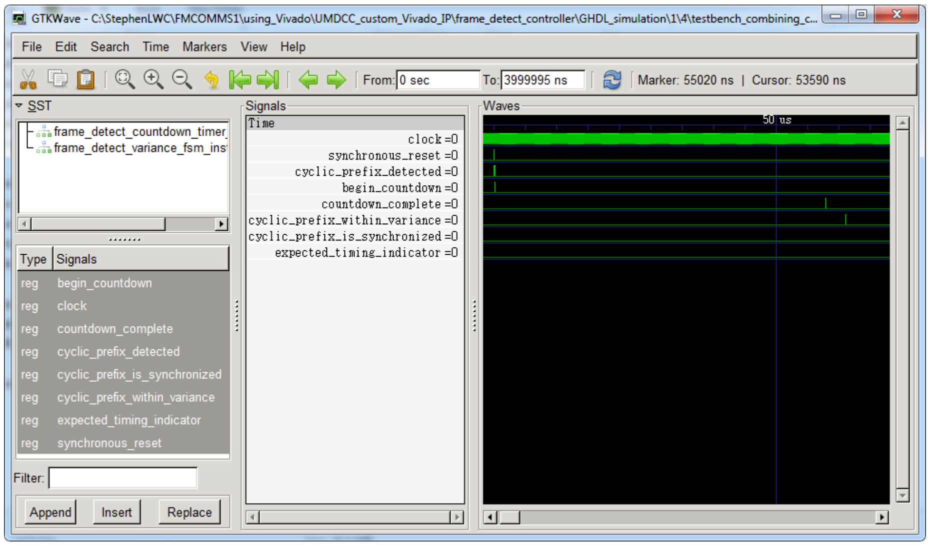
Figure 25: First pulse being received by Frame Detect Variance FSM.


GTKWave - testbench_combining_check_pulse_received_within_variance_tb_simulation.vcd

From: 0 sec To: 3999995 ns | Marker: 11015 ns | Cursor: 11060 ns

File Edit Search Time Markers View Help
SST
frame_detect_countdown_timer
frame_detect_variance_fsm_ins
Type Signals
reg begin_countdown
reg clock
reg countdown_complete
reg cyclic_prefix_detected
reg cyclic_prefix_is_synchronized
reg cyclic_prefix_within_variance
reg expected_timing_indicator
reg synchronous_reset
Filter:
Append Insert Replace
Signals
Time
clock =
synchronous_reset =
cyclic_prefix_detected =
begin_countdown =
countdown_complete =
cyclic_prefix_is_synchronized =
cyclic_prefix_within_variance =
expected_timing_indicator =
Waves
ns 10699 ns 10799 ns 10899 ns 10999 ns 11099

Figure 26: Second pulse being received and counting as within the acceptable time window by Frame Detect Variance FSM.


GTKWave - C:\StephenLWC\FMCOMMS1\using_Vivado\UMDCC_custom_Vivado_IP\frame_detect_controller\GHDL_simulation\1\4\testbench_combining_c...
File
Edit
Search
Time
Markers
View
Help
SST
Signals
Waves
50 us
Filter:
Append
Insert
Replace

From: 0 sec To: 3999995 ns | Marker: 55020 ns | Cursor: 53590 ns

frame_detect_countdown_timer_
frame_detect_variance_fsm_ins

Type	Signals
reg	begin_countdown
reg	clock
reg	countdown_complete
reg	cyclic_prefix_detected
reg	cyclic_prefix_is_synchronized
reg	cyclic_prefix_within_variance
reg	expected_timing_indicator
reg	synchronous_reset

```
Time
                        clock =0
              synchronous_reset =0
          cyclic_prefix_detected =0
                begin_countdown =0
             countdown_complete =0
   cyclic_prefix_within_variance =0
   cyclic_prefix_is_synchronized =0
       expected_timing_indicator =0
```

Figure 27: Second pulse being missed within the acceptable time window by Frame Detect Variance FSM.

3.3.5 Frame Detect Countdown Timer

The counter frame_detect_countdown_timer outputs a pulse of '1' for one clock cycle when the timer has counted from the reset value down to 0. It begins counting down when its start input is changed from '0' to '1'. VHDL generics set the number of clock cycles represented by the counter and the number of bits used to represent the countdown time. The generics account for different delays or window sizes.

Figure 28 displays a test of the countdown timer which uses a time of 5 and with a 3-bit counter representing a countdown times of 5 clock cycles. Note that 5 clock cycles exist between the start_in signal pulse and the countdown_complete signal pulse. Also note that the synchronous reset causes the countdown_complete pulse to not fire, and that a second start_in pulse shortly after the first doesn't interfere with the countdown_complete pulse.

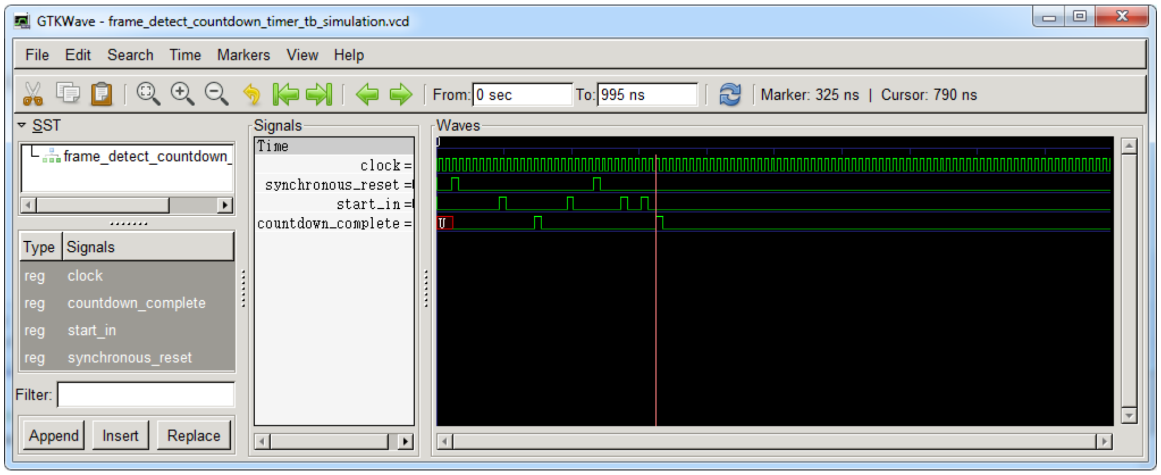


Figure 28: Test of Frame Detect Countdown Timer.

3.4 Markov Analysis of Peak Detector and Frame Detect Controller

A Markov chain is used to analyze the steady-state behaviour of the peak detector combined with Frame Detect Controller. A Markov chain is a discrete-time stochastic process consisting of a system of values where there is a single selected value that can transition to another state according to the Markov property [21]. These values can be conveniently thought of as states. The Markov property, or memorylessness, means each state has a constant probability of transitioning to each other state. These constant values can be organized into a structure called the state transition matrix. Each row in a state transition matrix represents a state, and each element of a row contains the probability of transitioning to the state number equal to its column index. The state transition matrix can be clearly represented using a state diagram such as in Figure 29.

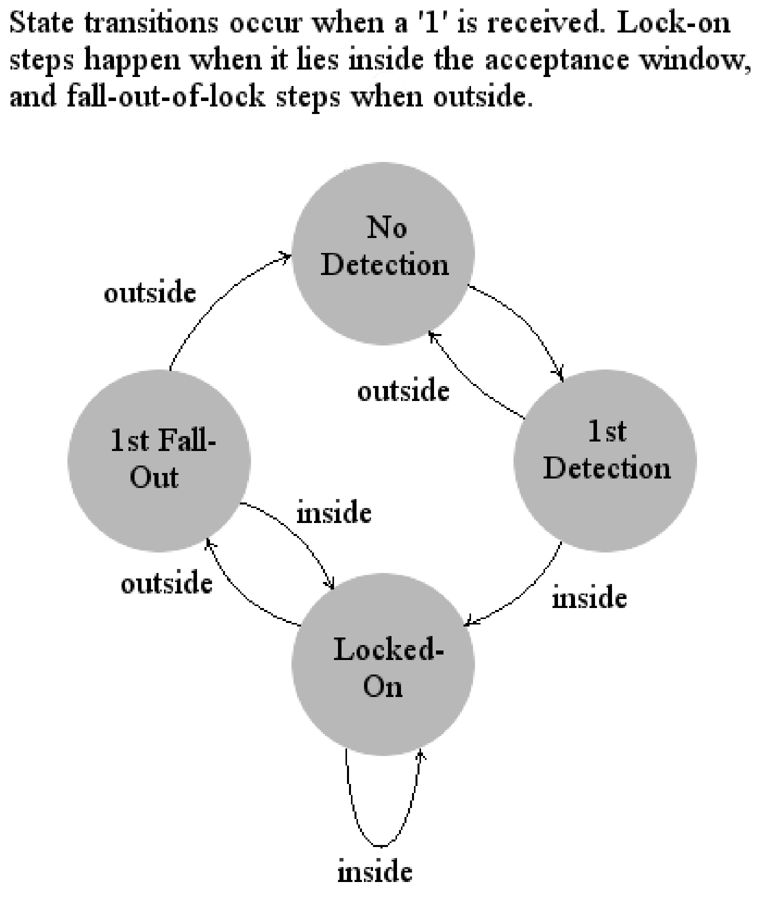

State transitions occur when a '1' is received. Lock-on steps happen when it lies inside the acceptance window, and fall-out-of-lock steps when outside.
No Detection
outside
outside
1st Fall-Out
1st Detection
inside
outside
inside
Locked-On
inside

Figure 29: A basic graphical description of a Markov chain.

Each step taken by a state in our Markov model occurs when the peak detector detects a new peak value and sends a '1' to Frame Detect Controller. The basic idea of Frame Detect Controller is a default state of No Detection, and when a '1' is received Frame Detect Controller "locks-on" by changing its state to 1st Detection. The next lock-on steps only occur if the received '1' lies inside the acceptance window. If a '1' is not received with the correct timing during lock-on, then the system will return to No Detection. Once enough '1's have been received with the desired timing, the Locked-On state is reached. If '1's continue to be received with the correct timing, Frame Detect Controller will remain in the Locked-On state. If not, then it will progress through fall-out-of-lock states until it reaches No Detection. If a '1' is received with correct timing while falling out of lock, then Frame Detect Controller will return to Locked-On. A simple way to modify the design for different situations is to change the number of lock-on states and fall-out-of-lock states.

How is the state transition matrix created? If the current state is no_detection, then the probability of transitioning to a future state given only a noise signal is shown in Figure 12. Using the situation where an OFDM frame is received at a particular SNR, the empirical relations for the probability lying in the acceptance window from Equation 8 can be applied, as well as the acceptance window probability from Equation 9, and a factor to the noise-only transition probabilities so that it approximates the noise distribution before the acceptance window when a frame is being received at a particular SNR. This noise factor comes from the probability lying before the acceptance window. This integration is expected to equal Equation 7 at the SNR under consideration, and so the factor is

$$\text{normalizing factor} = \frac{P_{\text{before}}(\text{SNR})}{1 - \exp(-\frac{(1088-243-31-1)^2}{(2\cdot(663)^2)})}. \tag{11}$$

The probability lying in each row of the state transition matrix must equal 1, but due to the approximations from above this may not be precisely the case. Another normalization step is applied to each row so that the sum of the row is always 1. This is important to prevent some Markov chain states from having a negative steady state value. Using these distributions leads to a state transition matrix with about 800 states per lock-on or fall-out-of-lock state. The behaviour of the Markov chain can be intuitively described using smaller transition probability matrices with only the probability of locking-on and falling-out-of-lock.

The state transition matrix for zero lock-on states/one lock-on step, and zero fall-out states/one fall-out step, is

$$\begin{bmatrix} 1-p & p \\ 1-p & p \end{bmatrix}. \tag{12}$$

The one lock-on state/two lock on steps state transition matrix will look like

$$\begin{bmatrix} 1-p & p & 0 \\ 1-p & 0 & p \\ 1-p & 0 & p \end{bmatrix}. \tag{13}$$

The one lock-on state/two lock on steps and one fall-out state/two fall-out steps state transition matrix will look like

$$\begin{bmatrix} 1-p & p & 0 & 0 \\ 1-p & 0 & p & 0 \\ 0 & 0 & p & 1-p \\ 1-p & 0 & p & 0 \end{bmatrix}. \tag{14}$$

Notice the pattern where there is always one more lock-on step compared to the number of lock-on states, and one more fall-out step than states. The example design given to Frame Detect Controller has three lock-on steps/two lock-on states and four fall-out steps/three fall-out states as in

$$\begin{bmatrix} 1-p & p & 0 & 0 & 0 & 0 & 0 \\ 1-p & 0 & p & 0 & 0 & 0 & 0 \\ 1-p & 0 & 0 & p & 0 & 0 & 0 \\ 0 & 0 & 0 & p & 1-p & 0 & 0 \\ 0 & 0 & 0 & p & 0 & 1-p & 0 \\ 0 & 0 & 0 & p & 0 & 0 & 1-p \\ 1-p & 0 & 0 & p & 0 & 0 & 0 \end{bmatrix}. \tag{15}$$

The more states spent locking-on the less likely the system is to lock on. The more states spent falling-out-of-lock the less likely the system will fall out of lock. A false-positive situation occurs where the receiver locks-on without receiving an OFDM frame. A false-negative situation occurs where the receiver fails to lock on to a received OFDM frame. Ideally there is a lock-on for every received OFDM frame and never a lock-on to noise. In practice there will be some amount of false-positives and false-negatives.

There exist a finite number of lock-on steps and fall-out-of-lock steps. If every sample is stored until a lock has been determined or not for it, then more states increases the latency between receiving each sample at the antenna and sending it to the channel estimation. In some applications false-positives or false-negatives will be more important, and so more states can be spent to reduce one or the other. The steady-state Markov analysis helps the designer choose the number of lock-on and fall-out states by estimating the system's performance at different SNRs.

The basic case from Equation 12 represented using the full set of states under consideration is illustrated by Equation 16 below

$$\begin{bmatrix} \begin{bmatrix} f \\ a \\ l \\ l \\ \\ o \\ u \\ t \end{bmatrix} & transitions & \begin{bmatrix} l \\ o \\ c \\ k \\ \\ o \\ n \\ \end{bmatrix} & 0 \\ \begin{bmatrix} f \\ a \\ l \\ l \\ \\ o \\ u \\ t \end{bmatrix} & 0 & \begin{bmatrix} l \\ o \\ c \\ k \\ \\ o \\ n \\ \end{bmatrix} & transitions \end{bmatrix}, \tag{16}$$

where "transitions" represents the set of probabilities of transitioning to the inner states, "lock on" represents the probability of transitioning to the lock-on state corresponding to the current step, and "fall out" represents the probability of transitioning to the fall-out state corresponding to the current step. With one lock-on state and zero fall-out states the probability transition matrix is illustrated in Equation 17 as

$$\begin{bmatrix} \begin{bmatrix} f\\ a\\ l\\ l\\ \\ o\\ u\\ t \end{bmatrix} & transitions & \begin{bmatrix} l\\ o\\ c\\ k\\ \\ o\\ n \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & 0 \\ \begin{bmatrix} f\\ a\\ l\\ l\\ \\ o\\ u\\ t \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & transitions & \begin{bmatrix} l\\ o\\ c\\ k\\ \\ o\\ n \end{bmatrix} & 0 \\ \begin{bmatrix} f\\ a\\ l\\ l\\ \\ o\\ u\\ t \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & 0 & \begin{bmatrix} l\\ o\\ c\\ k\\ \\ o\\ n \end{bmatrix} & transitions \end{bmatrix}. \tag{17}$$

In all of these matrix depictions, the matrices and vectors are arranged so the indexes match along the vertical axis. The final example will be the case for one lock-on step and one fall-out step in Equation 18 as

$$\begin{bmatrix} \begin{bmatrix} f \\ a \\ l \\ l \\ \\ o \\ u \\ t \end{bmatrix} & transitions & \begin{bmatrix} l \\ o \\ c \\ k \\ \\ o \\ n \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & 0 \\ \begin{bmatrix} f \\ a \\ l \\ l \\ \\ o \\ u \\ t \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & transitions & \begin{bmatrix} l \\ o \\ c \\ k \\ \\ o \\ n \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & 0 \\ \begin{bmatrix} 0 \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & 0 & \begin{bmatrix} l \\ o \\ c \\ k \\ \\ o \\ n \end{bmatrix} & transitions & \begin{bmatrix} f \\ a \\ l \\ l \\ \\ o \\ u \\ t \end{bmatrix} & 0 \\ \begin{bmatrix} f \\ a \\ l \\ l \\ \\ o \\ u \\ t \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & 0 & \begin{bmatrix} l \\ o \\ c \\ k \\ \\ o \\ n \end{bmatrix} & 0 & \begin{bmatrix} 0 \end{bmatrix} & transitions \end{bmatrix} . \tag{18}$$

The steady-state probabilities are computed for each Markov chain. We're interested in the the portion of transitions which are locked-on to the signal when the signal is available. This is given by taking the sum of steady-state probabilities over the falling-out-of-lock steps, because the receiver can't fall out of lock until it is locked-on. The rest of the states are locking-on, and the probability of being not locked-on is given by subtracting the portion of locked-on states from 1.

Example curves representing the steady-state probabilities of being locked-on are shown in Figure 30 along with the number of lock-on and fall-out states. The interesting design information is contained in the minimum value of the curve, how quickly the curve transfers from the noise-only probability to near 100% locked-on, and the 50% locked-on SNR.

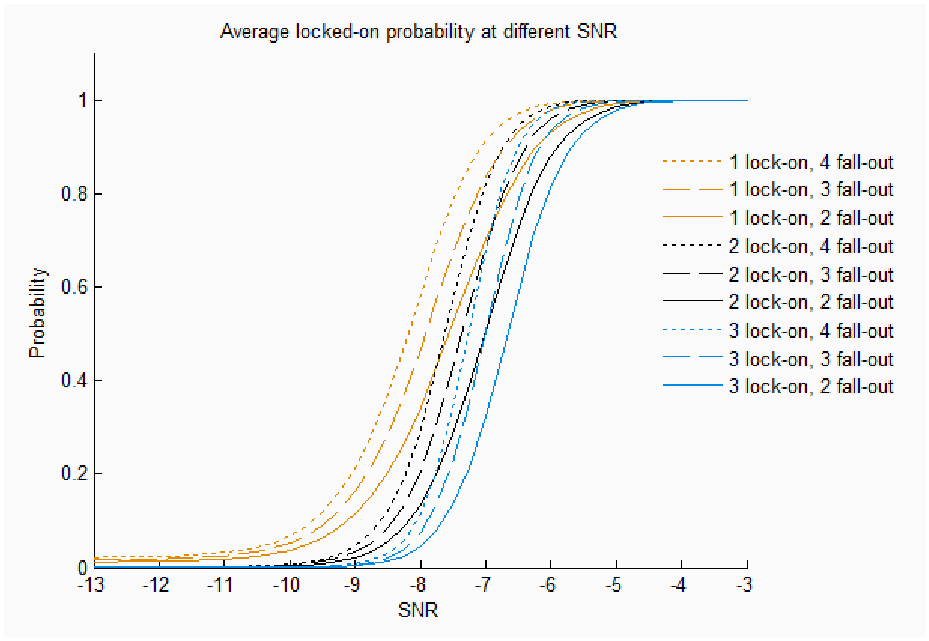


Figure 30: As the number of fall-out states increases, lock-on occurs more at lower SNRs. As the number of lock-on states increases, lock-on occurs less at lower SNRs.

The minimum value of the sigmoidal curve represents the portion of state transitions the system spends locked-on to a noise-only signal, otherwise known as the false-positive rate. These limits are shown in Figure 31, with the -15 dB SNR case assumed to be close to the minimum value. Notice that the number of fall-out stages affects the portion of false-positives less than the number of lock-on stages. Computing the ratio of lock-on vs out of lock between 1 lock-on stage and 2, and 2 lock-on stages versus 3, is about 15.9. This can be approximated intuitively by noticing the portion of time a peak may fall outside the acceptance window is 16 times greater than within the acceptance window. At the same number of lock-on steps in all three examples, increasing the number of fall-out stages from 2 to 3 gives a lock-on to not locked-on ratio of about 1.39, and from 3 to 4 gives a ratio of about 1.26.

It seems reasonable to begin designing a Frame Detect Controller situation by estimating the number of lock-on stages required using the minimum allowed portion of false-positives.

Note that the number of transitions expected when a frame is being received by the example design will be close to 1 per 1088 clock cycles, but the number of transitions expected when receiving pure noise is closer to the noise distribution peak around 906 clock cycles. This means that the number of false-positive situations is likely to be about 1088/906 ≈ 1.2 times the values shown in the Markov chain states.

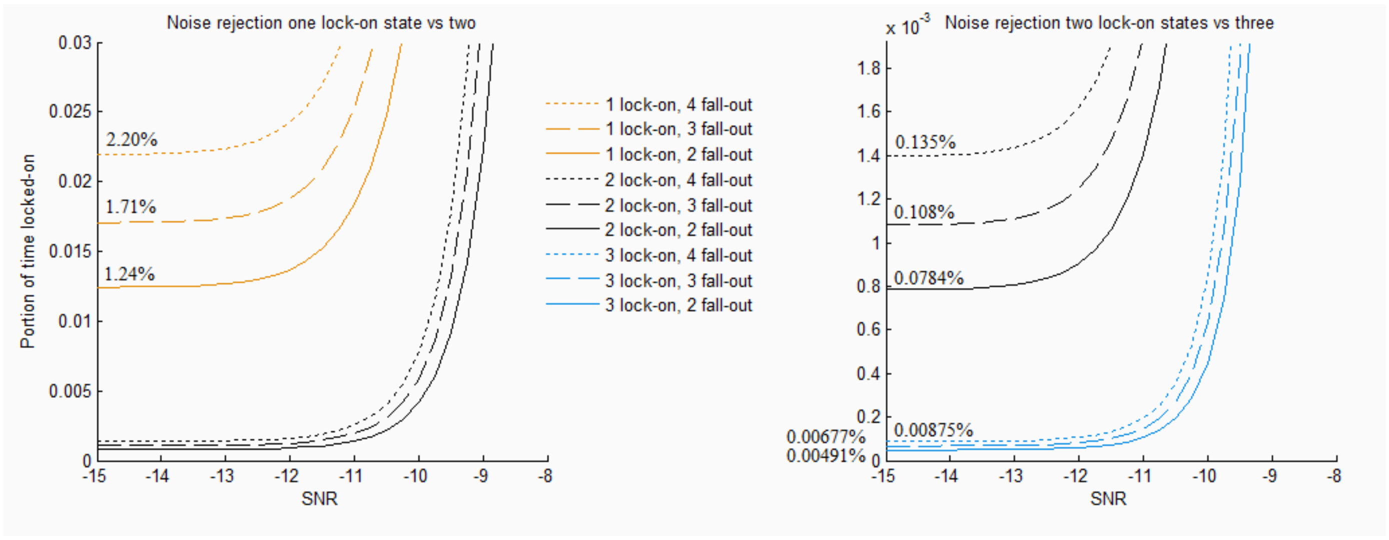


Figure 31: The minimum values are shown approximately at -15 dB SNR, which represent the portion of state transitions spent locked-on to a noise signal when there is no signal being received.

The center of the transition functions is approximately the 50% locked-on SNR. How quickly the transition occurs represents how quickly the system stops working at a given cutoff. When operating in a lower SNR range a shallower transition may be desired to lock-on more frequently. If operating at higher SNRs then a shorter transition period reduces the number of false-negatives.

Another trade-off to must make when choosing the total number of Markov chain states is the latency of the system. More states required storing more frames of data before deciding if they are locked-on or not. This latency may be dropped at the expense of losing the first few lock-on states worth of OFDM frames, and accept noise data before the lock has been lost.

Taking the derivative of the transitions provides an alternative way to see the effect of different numbers and ratios of states as in Figure 32. The functions formed are Gaussian-like, with their maximum located at the 50% locked-on SNR. Increasing the number of states reduces the variance and represents a faster transition. Increasing the number of fall-out states shifts the mean to a lower SNR so the system is more likely to be locked-on. Increasing the number of lock-on states shifts the mean to a higher SNR so the system is less likely to be locked-on. Notice that for a given number of lock-on states, increasing the number of fall-out states seems to form a ridge. Fitting a line to the peaks of these functions closely approximates this ridge, which may help a designer visualize the behaviour around a certain design value. Different situations can be simulated to explore the space of design trade-offs.

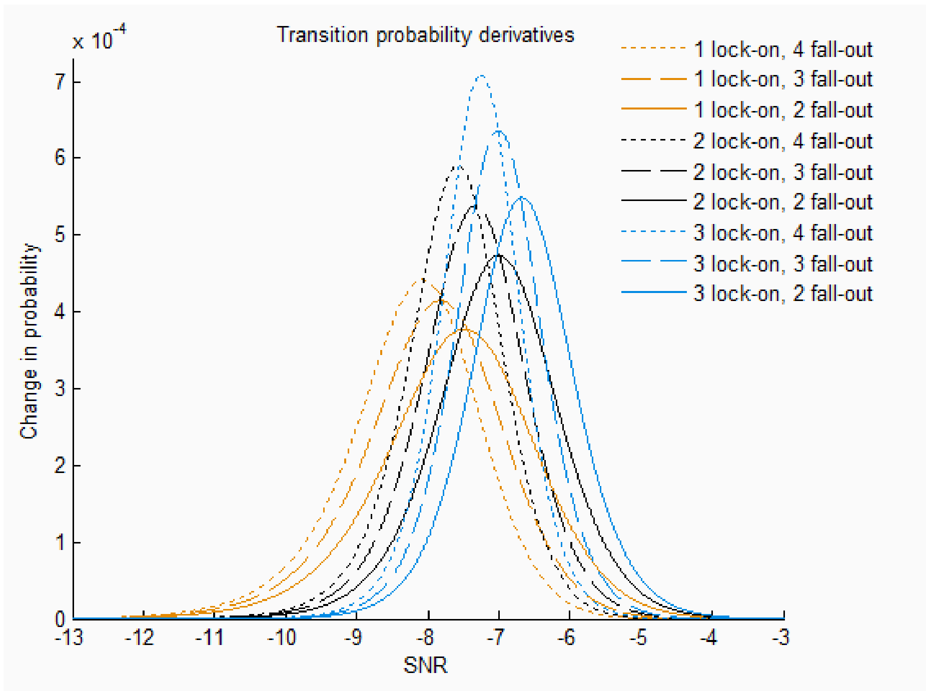


Figure 32: Derivatives of the transition functions. Notice the ridges for a given number of lock-on states tend to fall on a linear function.

Some questions to ask when designing the number of lock-on and fall-out states in Frame Detect Controller are:

What is the maximum allowed latency?

How many frames of data can be dropped before locking-on?

How many frames of noise can be received before falling-out-of-lock?

What is the worst-case expected channel SNR in which the system must perform?

What is the maximum allowed false-positive rate?

What is the maximum allowed false-negative rate?

According to these steady-state results, the Frame Detect Controller example case of two lock-on states and three fall-out states is expected to false-positively lock on to noise about 0.1% of the time, and for the system to be locked-on half of the time at about -7.3 dB SNR.

See SLWC>2>9>frame_received_peak_timing_distribution.m (2 - System Generator Projects, 9 - Peak Finder) for the MATLAB code and simulation.

Chapter 4 Carrier Frequency Offset Estimator

The Carrier Frequency Offset Estimator is a design which estimates the carrier frequency offset using the timing provided for detected signals from the Frame Detect Controller after a correlation peak detection has been found to lie within the window of acceptable timing. It uses the real and imaginary components of the correlation calculation corresponding to the correlation peak timing and accounting for the circuit latency. The basic idea behind the frequency offset estimator is that the correlation's real and imaginary values at the correlation peak correspond to the expected phase offset between the cyclic prefix and their corresponding tail data samples. This phase offset can be interpreted as a frequency since the timing between each pair of samples is known and equal to N ADC clock cycles.

The inputs to the carrier frequency offset estimator include the real and imaginary components of the correlation, which have a two's complement (35,27) number representation. The other inputs are one-bit control signals from the Frame Detect Controller, the frames_detected_pulse signal and the detection_lost signal. The outputs are the delayed real and imaginary correlation inputs, as well as the estimated angle of the complex input.

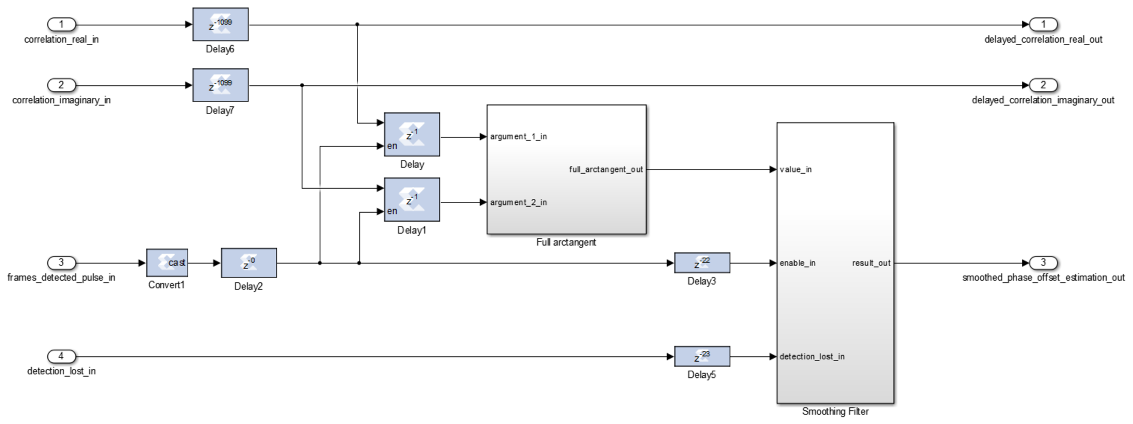


Figure 33: System Generator implementation of Carrier Frequency Offset Estimator top level.

4.0.1 Classic Phase Estimation

Given a set of complex values produced from a complex random process, and assuming a particular number of sinusoids, estimate their assumed parameters of amplitude, phase, and frequency offset. In our case, we are interested in estimating the single frequency offset for a situation like clock offset between transmitter and receiver, or a single coarse/average frequency offset for a Doppler spread across multiple frequencies. A single sinusoid hidden in additive Gaussian white noise has the following maximum likelihood (ML) Λ_{ML} for a given estimated frequency offset $\Delta\tilde{f}$ as given by [22]

$$\Lambda_{ML}(\Delta\tilde{f}) = |\sum_{i=1}^{\text{N}} r_i e^{-j2\pi\Delta\tilde{f}T_s}|^2, \tag{19}$$

where r_i is a received complex sample, T_s is the known sample period, and N in this case is the number of complex numbers being added, and not to be confused with the OFDM size. The maximum likelihood model can be viewed as representing the addition of N complex numbers which sum constructively to a maximum value where the estimated frequency offset is correct, and which sum with an expected value of 0 where the estimated frequency offset is incorrect.

The maximum likelihood frequency offset estimate is also given by the peak of the periodogram [23]. One way to find this estimate is to compute the power spectrum of the received data samples using a DFT and selecting the peak amplitude, since the sinusoid hidden within AWGN will produce a peak near its frequency. The resolution of the DFT (which is limited to the number of time samples being provided, and in our case is $M = 64$) will be $\frac{1}{M}$, and so the detected frequency will be within $\frac{1}{2M}$ of the actual frequency. The DFT effectively checks all frequencies and the frequencies which have the greatest amplitude response are the closest to correcting the offset frequency. The result can be computed closer to the precise frequency offset being estimated by performing a search step between the coarse DFT samples. Another way to get around this issue is to perform a larger DFT with zero padding, which performs interpolation in the time domain result.

This method acts almost as blind estimation and uses knowledge that the OFDM frame is composed of a set of sinusoids obscured by AWGN and potentially other effects. Using it may be effective when combined with the structure of the OFDM frame. Knowing each sample of the cyclic prefix corresponds to each sample of the tail data, an alternative method can be used which takes advantage of the statistical structure of the data and knowledge of the difference between the data at the same time.

4.0.2 Differential Detection

The number of clock cycles between replicated data is constant and known. Instead of using the samples directly, the difference in phase between the samples can be used to obtain the frequency offset. Comparing data within a communications frame is known as differential detection. Ways to compare the difference between complex samples are:

Subtraction $A - B = |A|e^{j\angle A} - |B|e^{j\angle B}$

Division $\frac{A}{B} = \frac{|A|}{|B|}e^{j(\angle A - \angle B)}$

Multiplication by complex conjugate $AB^* = |A||B|e^{j(\angle A - \angle B)}$

A frequency offset represents rotation in the complex plane, so it is convenient to retain knowledge of the two complex values' phases in the calculation. The phase difference between two complex numbers is provided by complex division or multiplying by the complex conjugate. It is convenient to use simpler operations, and since multiplying by a complex number is much simpler computationally compared to division of complex numbers, multiplying by the complex conjugate is the preferred option.

Let there be a complex signal received from a communications channel at discrete time n where s_n is the transmitted signal before being acted on by the channel and w_n is AWGN at time n. Also assume that a phase offset θ_n has been applied to the transmitted data at time n. The received signal at time n is given by

$$r_n = s_n e^{j\theta_n} + w_n. \tag{20}$$

Note that the transmitted signal s_n is unknown at the receiver except for the repeated data transmitted at N samples after the cyclic prefix using the relation

$$s_n = s_{n-N}. \tag{21}$$

Since we only wish to compare known data, Equation 21 only applies between cyclic prefix samples and tail data samples which correspond to the same frame k. By Equation 21 we also know that $|s_n| = |s_{n-N}|$. Knowing the transmitted signal has its own phase ϕ_n, we can use the polar form of s_n and place the phase into the unknown phase θ'_n using

$$\begin{aligned} r_n &= |s_n|e^{j\phi_n\theta_n} + w_n \\ &= |s_n|e^{j\theta'_n} + w_n. \end{aligned} \tag{22}$$

Multiplying two complex numbers results in a number with magnitude equal to the multiplied magnitude of each number, and a phase equal to the sum of each number's phase. The complex conjugate negates the phase of the delayed sample. Since within the same transmitted frame $s_n = s_{n-N}$, the received sample multiplied by its delayed and complex conjugated copy is

$$\begin{aligned} r_n r*_{n-N} &= (|s_n|e^{j\theta'_n} + w_n)(|s_{n-N}|e^{-j\theta'_{n-N}} + w_{n-N}) \\ &= (|s_n|e^{j\theta'_n} + w_n)(|s_n|e^{-j\theta'_{n-N}} + w_{n-N}), \end{aligned} \tag{23}$$

and here the difference in phase between the two complex samples is

$$\Delta\theta_n = \theta'_n - \theta'_{n-N}. \tag{24}$$

Ignoring noise, the phase offset between two complex numbers which are known to be identical before channel effects is given by finding the argument of their complex conjugate using

$$\begin{aligned} r_n r*_{n-N} &= |s_n|^2 e^{j(\theta'_n - \theta'_{n-N})} \\ &= |s_n|^2 e^{j\Delta\theta_n}. \end{aligned} \tag{25}$$

There are two main ways we can proceed to estimate the phase offset. The first is to find the arguments of each differential phase, and then find their average as in

$$\Delta\tilde{\theta}^{(k)}_{Arg,ACF} = \frac{1}{M}\sum_{m=0}^{M-1} Arg(|s_{n-m}|^2 e^{j\Delta\theta_{n-m}}) = \frac{1}{M}\sum_{m=0}^{M-1} \Delta\theta_{n-m}. \tag{26}$$

The second way is to perform the correlation, and then find the argument of the sample at its peak as in

$$\Delta\tilde{\theta}^{(k)}_{ACF,Arg} = Arg(\sum_{m=0}^{M-1} |s_{n-m}|^2 e^{j\Delta\theta_{n-m}}). \tag{27}$$

The real and imaginary value of the correlation function at the peak is approximately an average of the complex exponentials with their phase offset, and this phase offset can be measured to estimate the carrier offset frequency. Both options are available once a pipelined full arctangent function is available. Perhaps a comparison can made between the two methods using the arctangent identity

$$arctan(a) + arctan(b) = \begin{cases} arctan(\frac{a+b}{1-ab}) & ab < 1 \\ arctan(\frac{a+b}{1-ab}) + \pi & ab > 1, a > 0, b > 0 \\ arctan(\frac{a+b}{1-ab}) - \pi & ab > 1, a < 0, b < 0 \end{cases}. \tag{28}$$

The second method will be explored further. In practice it has been found that performing the sum of the correlated values first and then computing its angle has less variance than computing the phase of each differential complex conjugate and then taking their average. The second method also avoids issues when it comes to angles laying close to the branch point. Let

$$\begin{aligned} u_n &= \sum_{m=0}^{M-1} r_{n-m} r*_{n-N-m} \\ &= \text{Re}\{u_n\} + j\text{Im}\{u_n\}, \end{aligned} \tag{29}$$

which are computed by the correlation calculator. Using the relation described in Equation 25 allows

$$u_n = \sum_{m=0}^{M-1} |s_{n-m}|^2 e^{j\Delta\theta_{n-m}}, \tag{30}$$

where the M cyclic prefix samples align with their corresponding tail data samples. An approximation used in [9] assumes that the carrier frequency offset is small between received samples. Similarly, assuming the phase difference across each pair of corresponding samples is small allows the approximation

$$\begin{aligned} u_n &\approx \sum_{m=0}^{M-1} |s_{n-m}|^2 e^{j\Delta\theta^{(k)}} \\ &= e^{j\Delta\theta^{(k)}} |u_n|, \end{aligned} \tag{31}$$

where $|u_n| = \sum_{m=0}^{M-1} |s_{n-m}|^2$ is the amplitude of the autocorrelation, and $\Delta\theta^{(k)}$ is its phase.

The amplitude $|u_n|$ is not calculated directly within the correlation calculator, but its real and imaginary components are known and its phase can be found using the complex logarithm $\Delta\theta^{(k)} = \arg(u_n)$. The correlator calculator uses $|u_n|^2 = \text{Re}^2\{u_n\} + \text{Im}^2\{u_n\}$ to estimate the frame start timing for the rest of the calculations in the receiver because it is monotonic to $|u_n|$. Alternatively |Re|+|Im| could be used with some loss in SNR. Note that although different calculations can be used to determine the timing, the phase offset must be found for u_n using the real and imaginary values. Instead of performing the complex logarithm, we can also use the identities

$$\text{Re}\{u_n\} = |u_n|\cos(\Delta\theta^{(k)}) \iff \Delta\theta^{(k)} = \arccos(\frac{\text{Re}\{u_n\}}{\sqrt{|u_n|^2}}), \tag{32}$$

and

$$\text{Im}\{u_n\} = |u_n|\sin(\Delta\theta^{(k)}) \iff \Delta\theta^{(k)} = \arcsin(\frac{\text{Im}\{u_n\}}{\sqrt{|u_n|^2}}). \tag{33}$$

For example, if the offset $\Delta\theta^{(k)}$ can be assumed small, then the common small angle approximation for sine and cosine to find and correct the angle is

$$\begin{aligned} \mathrm{Re}\{u_n\} + j\mathrm{Im}\{u_n\} &= |u_n|(\cos(\Delta\theta^{(k)}) + j\sin(\Delta\theta^{(k)})) \\ &\approx |u_n|(1 + j\Delta\theta^{(k)}) \\ \iff \ \Delta\theta^{(k)} &= \tfrac{\mathrm{Im}\{u_n\}}{|u_n|}. \end{aligned} \tag{34}$$

Using the small angle approximation to estimate the phase offset also means that correcting the result can be performed using a small angle approximation. If not using the small angle approximation, the precision of several efficient-to-compute arctangent approximations is about 0.005 radians and covers the domain from 0 to 1 [24]. The small angle approximation compares to this error at around $0.005 \approxeq 0.31 - \sin(0.31)$, or 0.31 radians. It is desirable to be able to estimate the full range of principle arguments from -π to π radians because this allows a larger range of frequency offsets to be accounted for. The decided solution was to perform a full arctangent function to obtain the phase offset.

4.0.3 Limits of Phase Offset Estimation

The maximum angle which can be predicted by calculating the principal argument of a complex number is a small angle ϵ away from the branch point on the complex plane, in other words from $-\pi+\epsilon$ to $\pi-\epsilon$ radians. The value ϵ represents a small error away from the limit based on the calculator error and the variance of the estimated phase offsets. When the phase difference is calculated between two complex numbers using the multiplication by complex conjugate, the furthest angle two complex samples can have apart is π radians. These samples lie on opposite sides of the complex plane. Define the measurement period T_m in terms of sample period T_s and sample frequency f_s as

$$T_m = NT_s = \tfrac{N}{f_s}. \tag{35}$$

Without noise and assuming a constant frequency offset, a full rotation of a complex phasor over the measurement period occurs at the frequency $f_m = \frac{1}{T_m}$, and so a half rotation occurring over the same measurement period gives the maximum limit of how much frequency offset can be predicted of

$$\Delta f_{\max} = \tfrac{1}{2T_m}. \tag{36}$$

This is an example of the Nyquist sampling rate applied to estimating the frequency offset. An example for the FMCOMMS1 uses a sampling rate of $f_s = 122.88$ MHz with $N = 1024$, and the maximum Doppler frequency which can be estimated is less than 60kHz for a carrier frequency of 2.4 GHz. This limits the allowable clock offset between transmitter and receiver to 25 ppm for the example design.

Note that the frequency offset Δf for an estimated phase offset $\Delta\hat{\theta}$ is

$$\Delta f = \frac{\Delta\hat{\theta}}{2\pi T_m}. \tag{37}$$

4.0.4 Example Phase Offset Estimates

A representative example of the peak finder performance for a single correlation maximum is shown in Figure 34. A latency of three clock cycles is required by frame_detect_controller to determine when to produce a frame detected pulse. The real and imaginary parts of the correlation signal are delayed to represent this, however the peak finder is applied to the signal three clock cycles earlier in time.

The correlation peak amplitude's calculated timing estimates the actual timing. Finding the correlation peak timing is important to identify the phase offset estimate produced by the Phase Offset Estimator. Figure 35 shows the estimated phase offset for an example where the simulated phase offset is 30° and the SNR is 8 dB. The mean value is estimated to be around 0.524 rad (30.02°), as expected.

For simulations at different SNR, means are estimated to be 0.5227 rad at 3 dB, 0.5231 rad at 8 dB, 0.5230 rad at 13 dB, and 0.5244 rad at 18 dB. The 30° in radians is 0.5236, and so the means calculated here are all close to the expected phase offset. A small known bias exists in the division calculator, and this may explain the error between the estimated and computed coarse phase offsets. The estimated variance of the phase offset estimation for the 30° example is shown at the different SNRs in Figure 36.

To account for the variance between the estimated and actual phase offset in practice, an AR filter was created and placed after the output of the phase offset estimator. This filter type can be freely changed to balance the accuracy and variance of the estimated value.

The test to understand the effectiveness of this approach is to simulate an OFDM signal with cyclic prefix being transmitted over an AWGN channel. On the receiver side of the channel, the autocorrelation is performed between the cyclic prefix and tail data. The real and imaginary parts are shown in Figure 37, as well as the resulting estimated phase offset (located at 19 clock cycles of latency after the peaks). The

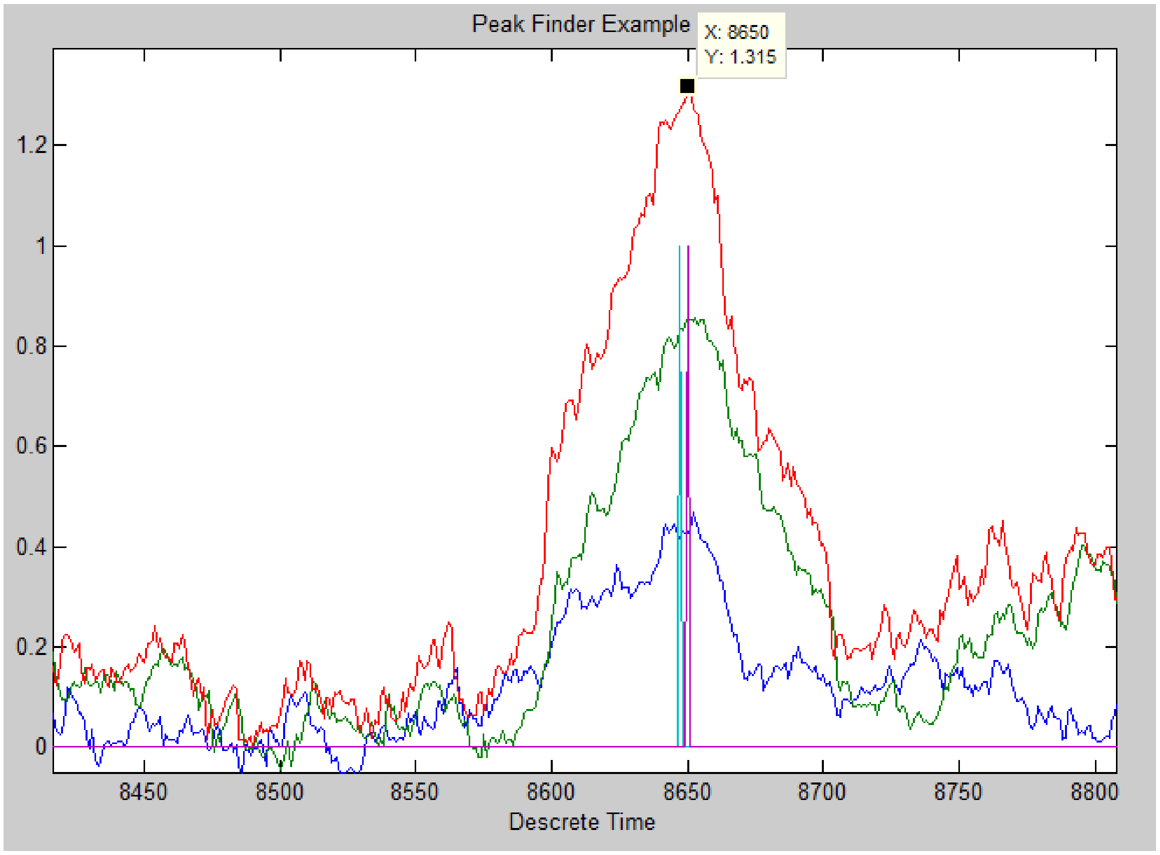


Figure 34: A local picture of the peak finder locating the maximum value. The peak finder value is teal and the frame_detect_controller is magenta. The sum of the magnitudes of the real and imaginary components is shown in red and the real and imaginary signals are green and blue respectively. The correlation signals are shown delayed in time by three clock cycles to match the timing of the frame_detect_controller pulse. The value is sampled at this time and sent to the Full Arctangent calculator.

phase offset is 15 degrees, and the result of 0.2487 radians corresponds to an angle of about 14.2 degrees. Similar tests were performed as a sanity check of the basic functionality for each special case and octant of 30°, 60°, 90°, 120°, 165°, 180° -165°, -120°, -90°, -60°, and -30°. The simulation environment used is the System Generator project "Golden Model 6". Similar results are seen in this test for signal-to-noise ratios on the order of 3 dB to 8 dB, although with more error in the estimation.

More examples are shown below in Figures 38 and 39.

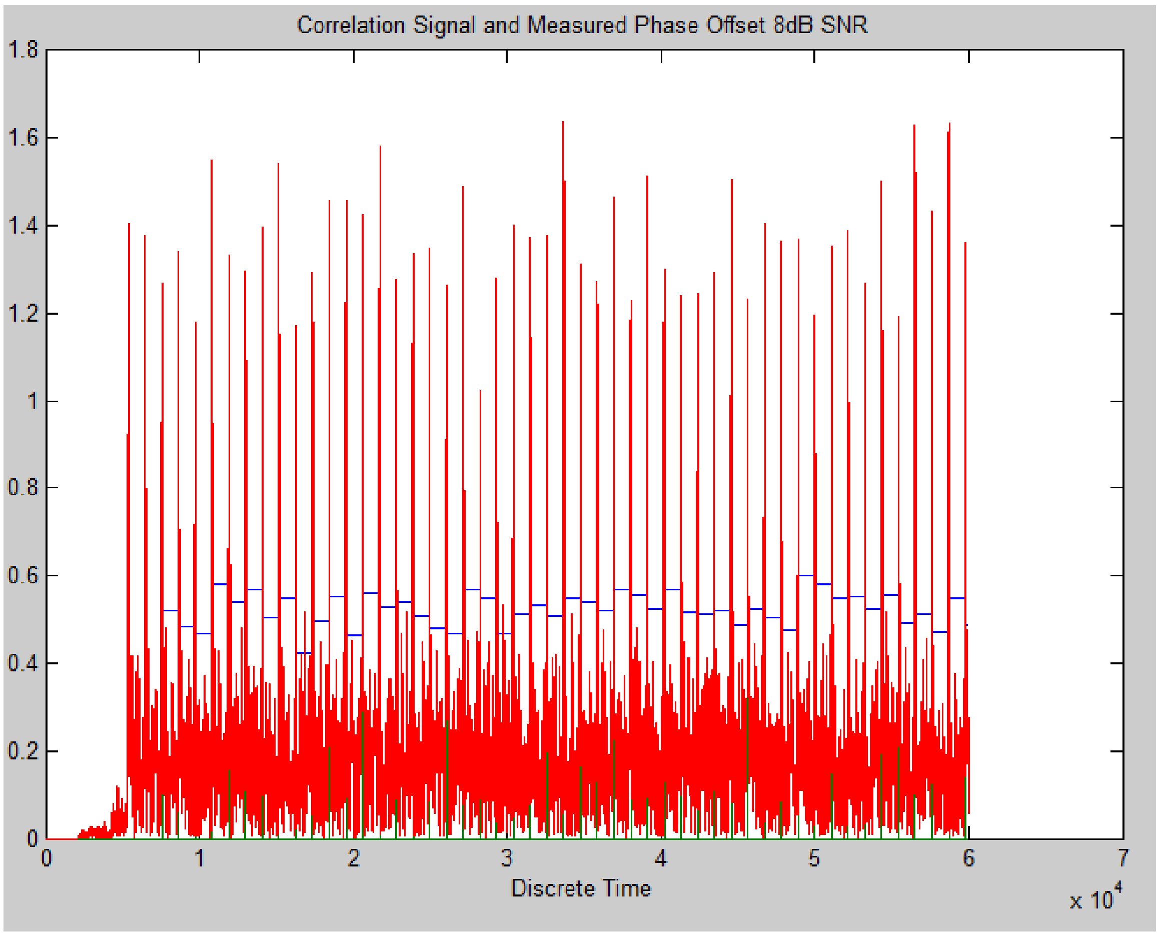


Figure 35: The sum of magnitudes of the real and imaginary components of the correlation is shown in red, and the estimated phase offset using the Full Arctangent function at the time given by the peak detector is shown in blue.

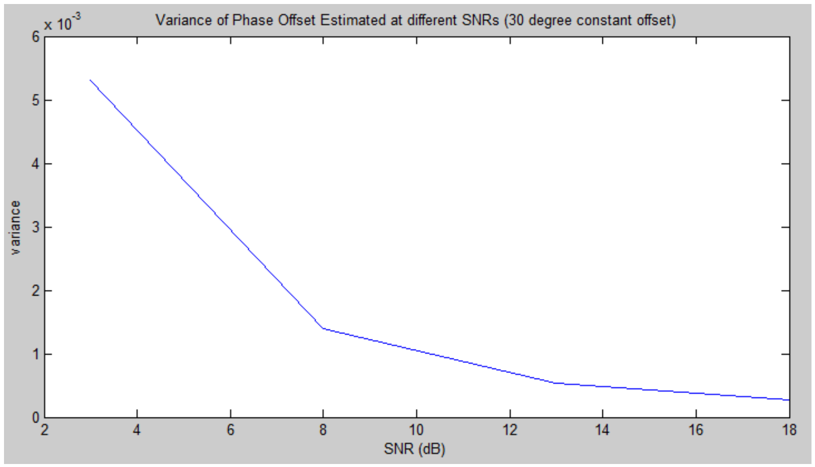


Figure 36: The decrease in the variance of phase offsets detected at different SNRs is shown. It appears that the variance decreases exponentially with SNR in dB, and so it likely varies inversely with signal power.

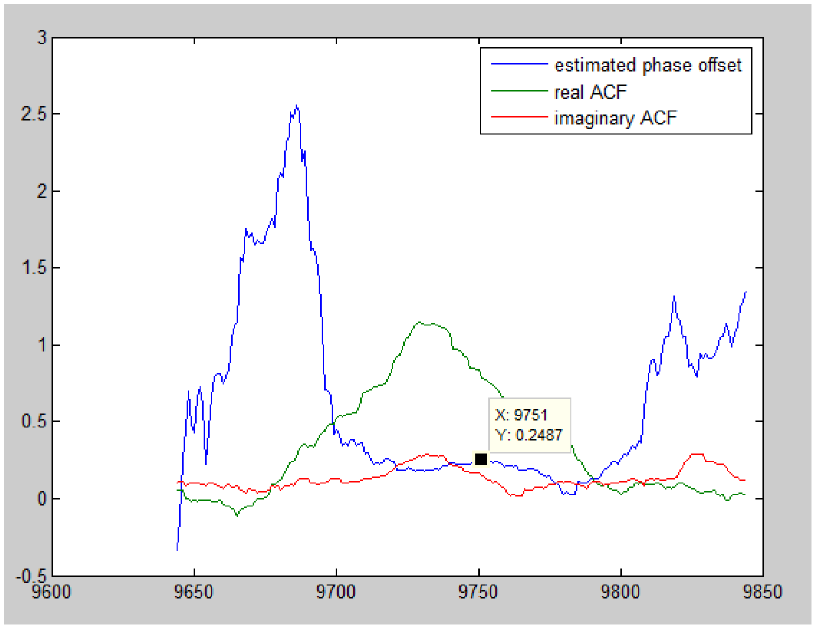


Figure 37: OFDM example affected by AWGN where we plot the received correlation's peaks, real value, imaginary value, and phase offset. The horizontal axis represents simulation clock cycle, and the vertical axis represents magnitudes developed during the autocorrelation. The marker is the sampled phase offset. In this case, the actual phase offset is 15° (0.2618 radians).

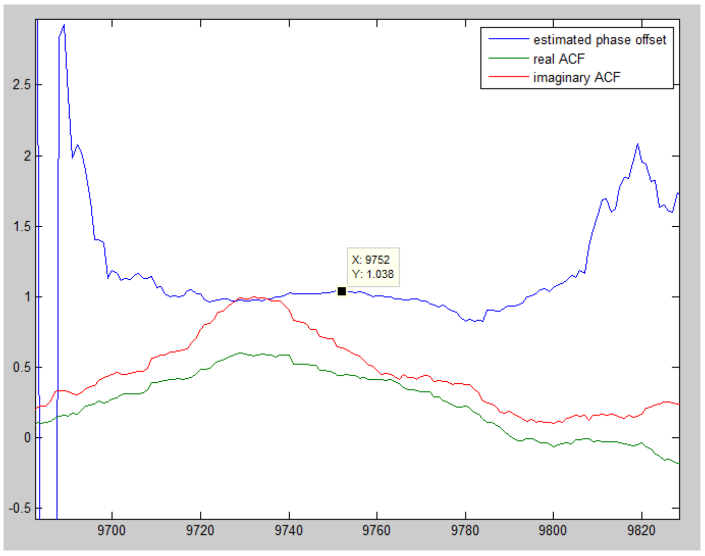


Figure 38: Real and imaginary autocorrelation values. Actual error is 60° (1.0472 radians).

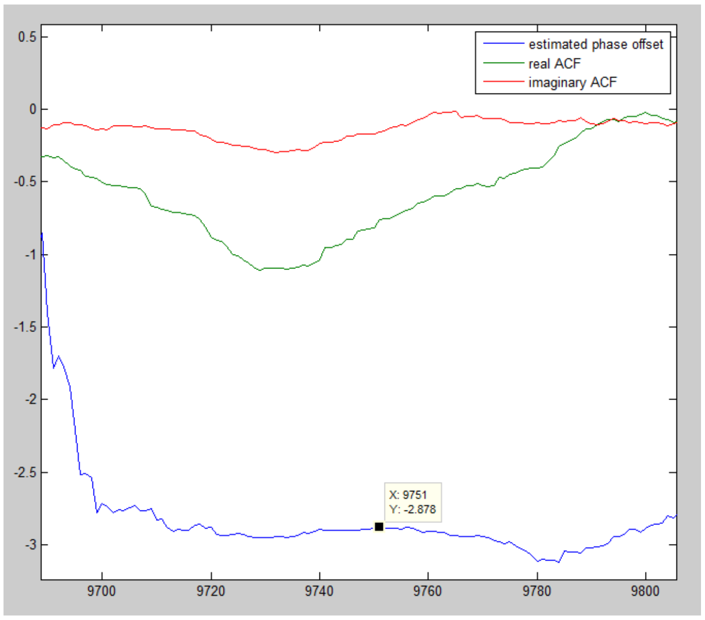


Figure 39: Real and imaginary autocorrelation peaks. Actual error is -165° (-2.88 radians).

4.1 Arctangent

The phase offset estimation can be broken down into performing the correct delays on the real and imaginary parts of the correlation signal, and then computing the full arctangent function. The first step of the full arctangent function is to implement the piecewise logic which determines the domain of the inputs and selects which calculations to perform, followed by a division operation, and finally the numerical arctangent itself is computed. This hierarchy is described in Figure 40.

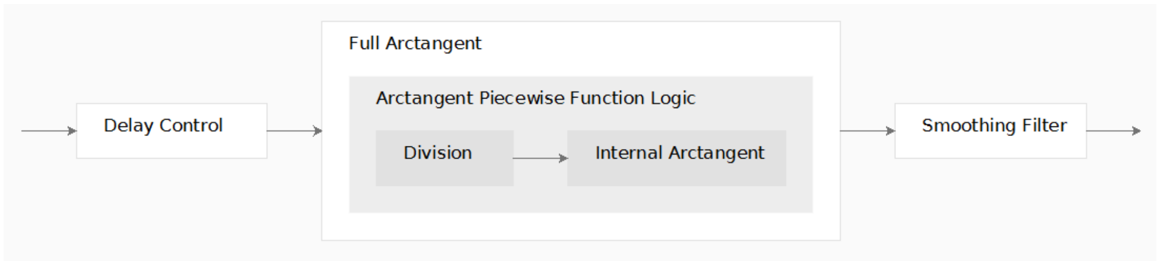


Figure 40: A high level description of the Phase Offset Estimator.

The full arctangent is a function which calculates the principal argument of a complex number, and can be decomposed into

$$\text{Full Arctangent}(\text{Re}\{x\}, \text{Im}\{x\}) = \begin{cases} \text{arctangent}(\frac{|\text{Im}\{x\}|}{|\text{Re}\{x\}|}) & \text{Re}\{x\} > 0 \text{ and } \text{Im}\{x\} > 0 \\ -\text{arctangent}(\frac{|\text{Im}\{x\}|}{|\text{Re}\{x\}|}) & \text{Re}\{x\} > 0 \text{ and } \text{Im}\{x\} < 0 \\ \text{arctangent}(\frac{|\text{Im}\{x\}|}{|\text{Re}\{x\}|}) - \pi & \text{Re}\{x\} < 0 \text{ and } \text{Im}\{x\} < 0 \\ -(\text{arctangent}(\frac{|\text{Im}\{x\}|}{|\text{Re}\{x\}|}) - \pi) & \text{Re}\{x\} < 0 \text{ and } \text{Im}\{x\} > 0 \\ \frac{\pi}{2} & \text{Re}\{x\} = 0 \text{ and } \text{Im}\{x\} > 0 \\ -\frac{\pi}{2} & \text{Re}\{x\} = 0 \text{ and } \text{Im}\{x\} < 0 \\ 0 & \text{Im}\{x\} = 0 \end{cases} . \tag{38}$$

The piecewise logic determines the appropriate sign, addition, or constant selection for the output. Typically the branch point $\text{Im}\{x\} = 0$ and $\text{Re}\{x\} < 0$ is undefined. However, whenever $\text{Im}\{x\} = 0$ we assume the angle is zero. This is acceptable because if the angle reaches the branch point it cannot be estimated by this estimator and the system will fail. Detecting this failure is simpler to design separately from the calculation.

A division circuit was designed and placed in series with an internal arctangent function. The internal arctangent function is composed of a LUT composed of values from MATLAB's atan(x) function. The available arctangent approximations have an important role in deciding the number format to use when computing the division. A number of approximations are listed in [24] which have typical worst-case accuracy of 0.005 radians (0.29°) for inputs on the range $-1 \leq x \leq 1$. The RMS error propagated through an arctangent function is (for σ representing standard deviation of a variable)

$$\sigma_{\text{atan}} \approxeq \tfrac{\sigma_x}{\sqrt{1+x^2}}, \tag{39}$$

and if the input to the arctangent function $x \geq 0$, then an upper bound on the RMS error of the arctangent function's output is σ_x for $x = 0$. For an arctangent function with a worst-case error 0.005, the incoming division error does not need to be much less than 0.005 to retain full precision of the arctangent approximation.

Since the division circuit produces a value on the range $[0.5, 1)$, it is best described by a number representation which is unsigned and has only fractional bits. One step better, similarly to the IEEE floating point representation, is to use an assumed bit to represent the value 0.5. Then, the remaining bits represent the fractional number between 0.5 and 1. Performing Goldschmidt division [25] finds factors which send the denominator to 1 and the numerator to the result of the division. However, for numerators which are greater than the divisor the result is greater than 1, and a number format for the numerator is needed which covers the maximum expected numerator value. To get around this requirement we can notice that the arctangent function approximations in [24] operate on the input range $[0, 1]$ and take advantage of the arctangent identity

$$\text{Arctangent}(\tfrac{1}{x}) = \tfrac{\pi}{2} - \text{Arctangent}(x) \qquad x > 0. \tag{40}$$

Knowing the input to the arctangent function is positive, noticing that the number format of the division result is simpler when represented using a purely fractional number, and that Equation 40 allows the arctangent approximations in the range $[0, 1]$ to be applicable for $(1, x \to \infty)$. This makes a piecewise function for the division effective. The division performed is

$$\text{division}(\text{Re}\{x\}, \text{Im}\{x\}) = \begin{cases} \frac{\text{Im}\{x\}}{\text{Re}\{x\}} & \text{Im}\{x\} \leq \text{Re}\{x\} \\ \frac{\text{Re}\{x\}}{\text{Im}\{x\}} & \text{Im}\{x\} > \text{Re}\{x\}, \end{cases} \tag{41}$$

which results in the arctangent function

$$\text{Arctangent}(\tfrac{\text{Im}\{x\}}{\text{Re}\{x\}}) = \begin{cases} \text{Arctangent}(\frac{\text{Im}\{x\}}{\text{Re}\{x\}}) & 1 \geq \frac{\text{Im}\{x\}}{\text{Re}\{x\}} \\ \frac{\pi}{2} - \text{Arctangent}(\frac{\text{Re}\{x\}}{\text{Im}\{x\}}) & 1 < \frac{\text{Re}\{x\}}{\text{Im}\{x\}}. \end{cases} \tag{42}$$

The full arctangent function is composed of several sections. First, it determines which output case to perform. The possible output cases include the corner cases of inputs 0, $\frac{\pi}{2}$, and $-\frac{\pi}{2}$. The logic to do this is shown in Figure 41. The output of this logic is delayed to match the timing of the longest latency in the calculation. This delay allows the design to be considered pipelined with one input resulting in the correct output a fixed time period later. The output is selected using a MUX as shown in Figure 43. The piecewise logic of the full arctangent calculation is shown in Figure 42 and it determines whether to divide real input by imaginary or vice versa, performs the division, and performs the calculations to apply to the arctangent function. The arctangent within this design represents the numerical part of the arctangent function defined for inputs in the range of [0,1).

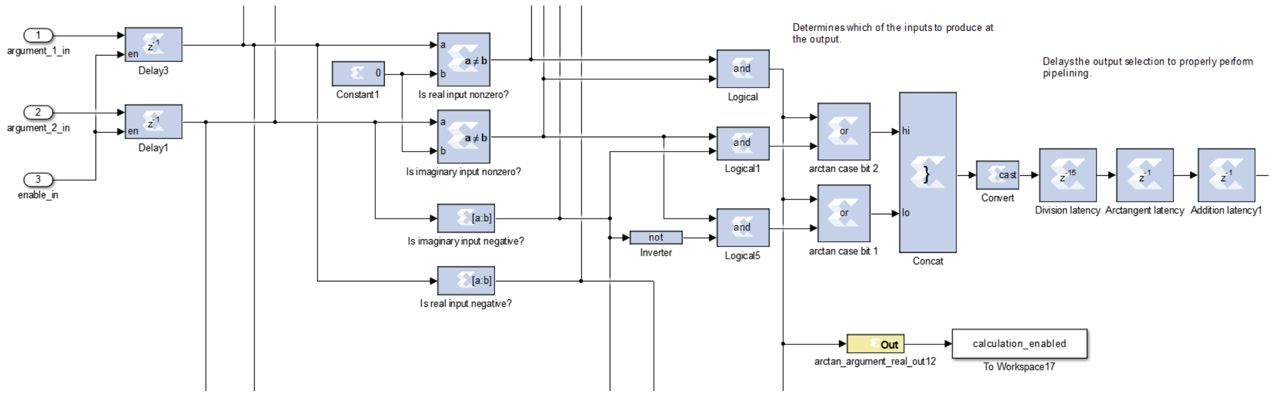
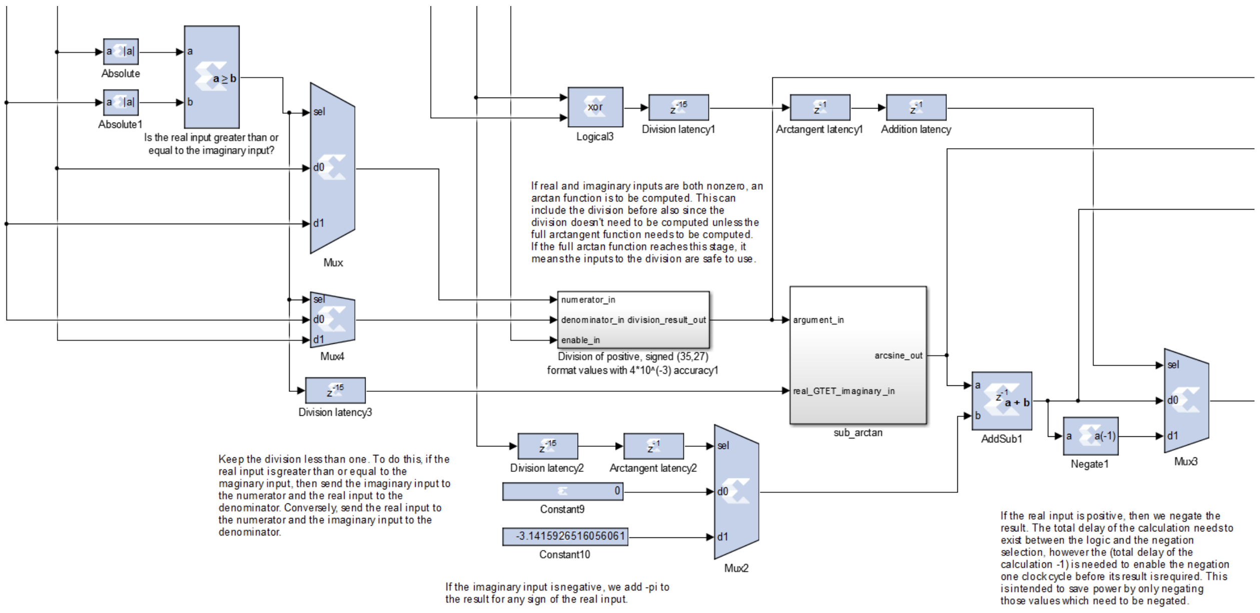


Figure 41: Output type selection logic and pipelining.



Is the real input greater than or equal to the imaginary input?

If real and imaginary inputs are both nonzero, an arctan function is to be computed. This can include the division before also since the division doesn't need to be computed unless the full arctangent function needs to be computed. If the full arctan function reaches this stage, it means the inputs to the division are safe to use.

Keep the division less than one. To do this, if the real input is greater than or equal to the imaginary input, then send the imaginary input to the numerator and the real input to the denominator. Conversely, send the real input to the numerator and the imaginary input to the denominator.

If the imaginary input is negative, we add -pi to the result for any sign of the real input.

If the real input is positive, then we negate the result. The total delay of the calculation needs to exist between the logic and the negation selection, however the (total delay of the calculation -1) is needed to enable the negation one clock cycle before its result is required. This is intended to save power by only negating those values which need to be negated.

Figure 42: Division type selection, division circuit, arctangent circuit with constant and negation selection, and pipelining.

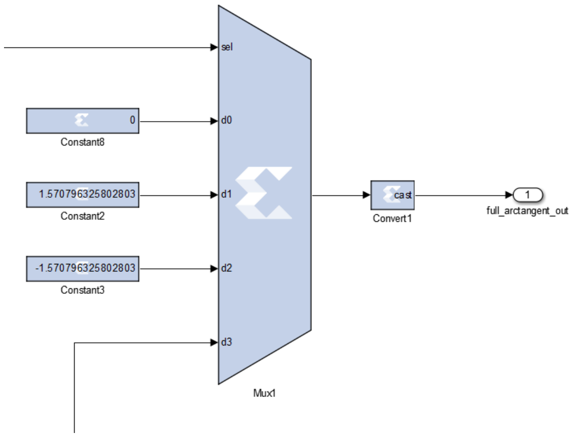


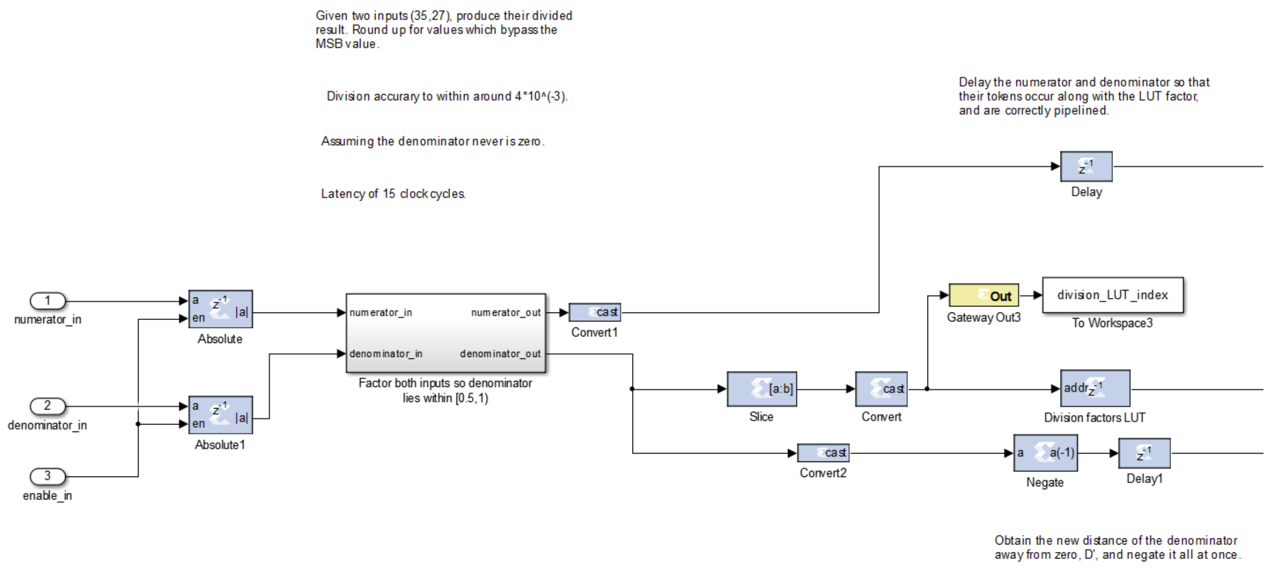
Figure 43: The appropriate output is selected using a multiplexer.

4.2 Division

The division was broken into the following steps:

Division algorithm of $\frac{\nu}{d}$ given a known numerator ν and denominator d

1. Compute $|\nu|$ and $|d|$
2. Factor the same power of 2 from $|\nu|$ and $|d|$ to create ν' and d' so d' lies within $[0.5, 1)$
3. Multiply ν' and d' by a LUT factor chosen so that d' approaches 1, giving ν'' and d''
4. Output $\nu'' \times$ the first factor of Goldschmidt division using the Binomial Theorem


Given two inputs (35,27), produce their divided result. Round up for values which bypass the MSB value.

Division accurary to within around 4*10^(-3).

Assuming the denominator never is zero.

Latency of 15 clock cycles.

Delay the numerator and denominator so that their tokens occur along with the LUT factor, and are correctly pipelined.

Obtain the new distance of the denominator away from zero, D', and negate it all at once.

Figure 44: This diagram contains the first part of division where the absolute value is taken, the bits of the representation are shifted so the denominator lies on the range [0.5,1), and the index of a LUT step is acquired and used to obtain the stored value to send d close to 1.

The divider circuit design in System Generator is shown in Figures 44 and 45.

The number format of the autocorrelation function real and imaginary values is signed (35,27). The first step is to produce the output of the absolute value circuit, which has a number format of unsigned (34,27). It is important in practice to consider the number format of the calculator's components. If the real input is always divided by the imaginary input or vice versa, then dividing a large numerator by a small denominator may be required for a particular pair of real and imaginary inputs. The largest possible output number format required for the division calculator is obtained by shifting a '1' in the MSB of the numerator by the largest left-shift performed by the factoring circuit of << 26, which corresponds to a '1' in the LSB of the denominator. This value results in a MSB value of $\frac{2^8}{2^{-26}} = 2^{34}$, and with a 27 bit fraction the division output would require a 61 bit representation. This large number format creates several challenges in implementation, such as there not currently existing a multiplier which can handle a 61-input-bit multiplication, or timing issues for addition at the clock rate of 125 MHz. Fortunately, using Equation 42 allows the division's numerator to remain less than or equal to the denominator and the output number format can use the same or fewer bits compared to the input numbers.

The second step is to factor $|d|$ so that it lies on the range [0.5,1), and to provide the same factorization to $|\nu|$. This is provided by a custom circuit. Approximate multiplication by powers of 2 using a bit-shift is essentially a free operation on an FPGA because each bit must be routed to a new location anyway, and what changes is not the hardware itself but how the following hardware interprets the bits. However, the overall circuit which performs the factorization isn't free because it is composed of a combinational part which determines the most significant '1' in the input numerator and denominator magnitudes, and uses this "shift code" to select the corresponding right- or left-shift so that the resulting number lies within [0.5,1). Every possible shift is allowed by the circuit, and so the result is the input token routed through the corresponding bit shift. One potential issue with a large combinational circuit such as this is timing failure due to a long path of logic. In practice, timing is met without issue at a clock rate of 125MHz.

After d' is obtained, a LUT value is multiplied by ν' and d' to obtain ν'' and d'', and then Goldschmidt division is performed. Goldschmidt (GS) division provides the quotient $Q = \frac{\nu''}{d''}$ by multiplying ν and d by values $K_0K_1K_2...K_n$ so that as the value $d'' \to 1$, $\nu''K_0K_1...K_n \to Q$. The constants K_i can be chosen in several ways, and using the binomial theorem they can be chosen so that for $x = 1 - d''$, $K_0 = 1 + x$, $K_1 = 1 + x^2$, and the process continues with each new multiplier being $K_i = 1 + x^i$. This has the effect that

$$Q = \frac{\nu''}{d''} = \frac{\nu''}{1-x} = \frac{\nu''(1+x)}{(1-x)(1+x)} = \frac{\nu''(1+x)}{1-x^2} = \frac{\nu''(1+x)(1+x^2)}{(1-x^2)(1+x^2)} = \frac{\nu''(1+x)(1+x^2)}{1-x^4}. \tag{43}$$

This process continues, and has quadratic convergence for values of x close to 0 and d' close to 1, with

$$\text{error}_{\text{relative, worst case}} \leq \frac{1}{d'K_0 \ldots K_i} = \frac{1}{1-(1+(2^{-1})^{2^i})} = 2^{-2^i}. \tag{44}$$

The purpose of Goldschmidt division is to obtain a division result after a short latency. Due to the arctangent approximations and reasoning from before, the division here only needs to provide an output with a relative error of about 0.005. To achieve this, the division needs to be accurate to about 2^{-8}. The worst case of error for Goldschmidt division is where $d' = 0.5$ at the beginning of the algorithm, and so we require 3 stages to reach 2^{-8}.

Looking for a solution which can approach the result in fewer multiplications, I use the idea that lookup tables (LUTs) provide low latency, and looked at different constants which can be multiplied by d' so that it approaches 1 without becoming larger than 1, but which is effective for bringing the worst-case values close to 1. The initial estimate d'' can be obtained by a LUT applied to the range of values within [0.5,1). The LUT values are formed by discretizing this range into 2^n regions, where the integer index j varies on the range $0 \leq j \leq 2^{n-1}$ and each region is specified by

$$[\frac{2^n+j}{2^{n+1}}, \frac{2^n+(j+1)}{2^{n+1}}). \tag{45}$$

Multiplying each range by the reciprocal of the upper value,

$$\frac{2^{n+1}}{2^n+(j+1)}, \tag{46}$$

causes the upper value of each range being set to 1, which satisfies the requirement of a result no greater than 1, and increases the lower value of each range so it approaches 1. Performing the LUT multiplication produces a value in the range

$$[\frac{2^n+j}{2^n+(j+1)}, 1). \tag{47}$$

Table 2: Example LUT with $n = 2$. For these entries the ability to send each index range closer to 1 is shown.

Indexed Range	LUT factor	Resulting range
$[\frac{1}{2}, \frac{5}{8})$	$\frac{8}{5}$	$[\frac{4}{5}, 1)$
$[\frac{5}{8}, \frac{6}{8})$	$\frac{8}{6}$	$[\frac{5}{6}, 1)$
$[\frac{6}{8}, \frac{7}{8})$	$\frac{8}{7}$	$[\frac{6}{7}, 1)$
$[\frac{7}{8}, 1)$	1	$[\frac{7}{8}, 1)$

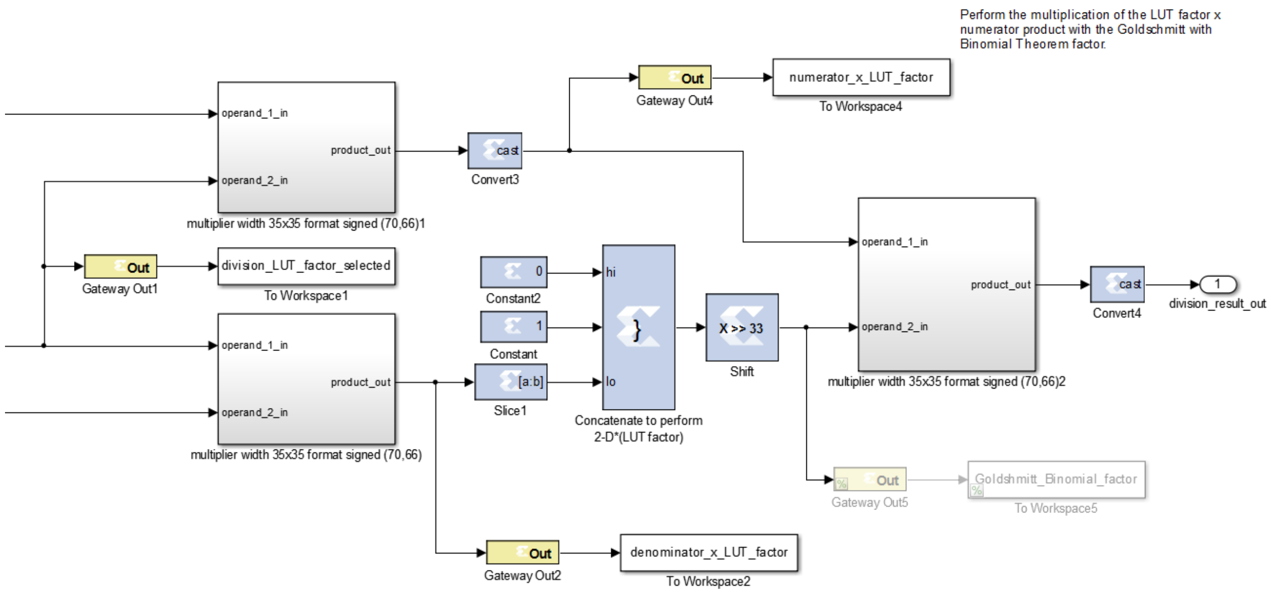


Figure 45: This diagram shows the multiplication of D' and N' with the LUT factor, followed by the 2-D" calculation, and then multiplication of N" with the GS factor. Note that D in the SysGen diagram is d in the text, and N in the SysGen diagram is ν in the text.

Using a LUT with a factor of 2^n items is convenient because the LUT index can be read directly from the binary number representation of d'. Having the first step use a LUT is better than having the first step being a Goldschmidt step. For example a table with $n = 2$ causes the worst case error to reach 0.8, versus one step of the binomial Goldschmidt algorithm's 0.75. The performance benefit of a larger LUT for the first step increases for larger n, although its resource requirements double according to 2^n. Each Goldschmidt step approaches the result quadratically, and is therefore more effective with a closer initial guess. One method of combining these two methods is to perform one LUT step to have the result quickly approach within a small distance of 1, and then perform one Goldschmidt step to get much closer.

A generic relation for combining a LUT step followed by the binomial Goldschmidt step is

$$\text{error}_{\text{LUT-GS, relative, worst-case}} \leq (2^n + 1)^{-2}, \tag{48}$$

and we can approximate the relative error as 2^{-2n} as n grows.

One unfortunate part of combining these two methods is that the Goldschmidt step's x must be based off of d'', which is the result of the LUT step. Using only Goldschmidt steps, no multiplications need to be done with the denominator. This means that for the same latency as one LUT step and then a Goldschmidt step, and the same number of multiplies, we could perform two Goldschmidt steps. However, the resources required to outperform two Goldschmidt stages are meager. Choosing n=4, or 16 LUT entries, the worst case error of the division with LUT then GS is 0.0035, while the worst case error for 2 steps of GS is $(2^{-1})^4 = 0.0625$. The implemented multiplications using the LUT values are shown in Figure 45, and then the GS factor is computed and applied in one more multiplication.

Determining the GS factor is also a bit clever, and uses the fact that $1+x = 2-d''$ is always performed, and that $d'' \geq 0.5$. In binary,

$$d'' = 0.1b_{-2}b_{-3}b_{-4}b_{-5}... \tag{49}$$

The constant level result is always 1, since $d'' \leq 0.5$. When subtracting the fractional part of d'', each fractional bit of d'' is added to 0 from the fractional part of 2. This means that the fractional bits of d'' *are* the fractional bits of $2 - d''$. Also note that the negative sign can be applied before the multiplication of d' with the LUT factor. For example:

$$\begin{array}{lc} 010.000... & 2 \\ +\,111.b_{-1}b_{-2}... & -\,d'' \\ =\,001.b_{-1}b_{-2}... & \text{result.} \end{array} \tag{50}$$

After the $2 - d''$ is performed, multiplying the numerator by the GS factor gives the result of the division.

4.3 Internal Arctangent

The internal arctangent design is shown in Figure 46. The results are obtained by performing a LUT indexed by the values of the incoming result from the division.

Any of the approximations described in [9] could also be used. The logic, sign, and addition performed after the LUT access are used to differentiate between the case $1 \geq \frac{\text{Im}\{x\}}{\text{Re}\{x\}}$ and $1 < \frac{\text{Im}\{x\}}{\text{Re}\{x\}}$ of Equation 42.

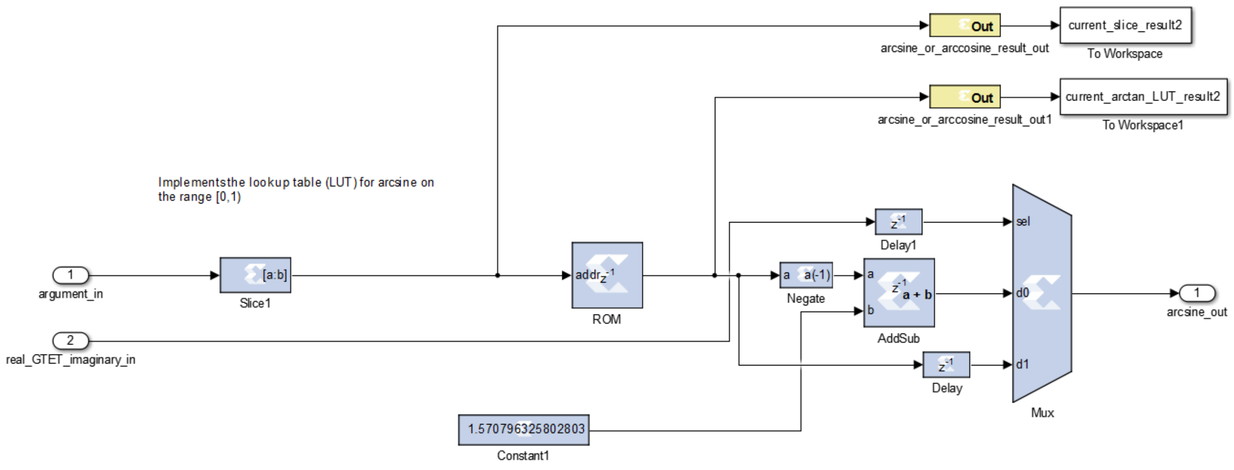


Figure 46: The arctangent index is obtained from the incoming number representation, and then applied to a LUT to get the result. The remaining logic decides how to handle the data for according to the cases in Equation 42.

4.4 Smoothing Filter

As can be seen from the simulation results in Figure 35, there is variance in the estimated phase offset. In the simulation, the underlying phase offset remains the same, and so the variance seen in the estimated phase offset is due purely to the effect of the data and noise acting as a random process. Depending on the required phase offset correction precision, there is a need to filter the predicted phase offset and obtain a more reliable estimate.

A simple auto-regressive (AR) filter of order 1 was created to provide this smoothing to the output of the phase offset estimator. The AR filter is implemented as a FSMD similarly to Frame Detect Controller. The reason for this implementation is to initialize the filter with the first phase offset estimation value so that multiple OFDM frames are not required before an appropriate phase result is provided to the phase offset compensator. Without initialization, the AR filter would begin with an assumed state such as 0 and require multiple frames to approach the actual phase offset parameter.

The AR filter controller FSM is created in VHDL, and the datapath is created in System Generator. An overview of their connections is shown in Figure 47. The internal components of the datapath are shown in Figure 48. The latency of the AR Filter is four clock cycles.

Using the 8 dB SNR case as an example, the variance of the recovered phase estimate was calculated to be about 0.0015 rad^2, or a standard deviation of about 0.04 rad. At this level or phase offset estimation error, Equation 37 shows for $N = 1024$, $\Delta\hat{\theta} = 0.04$ rad, and a sampling rate of 122.88 MHz, the RMS frequency error will be around 750 Hz. Using an AR filter with coefficient 0.1, the variance is reduced to about 8×10^{-5} rad^2, or a standard deviation of about 0.008 rad. At this deviation the result produces a frequency offset around 150Hz after about 15 OFDM frames have occurred. Since the variance decreases proportionally with the number of samples used in a moving average (MA), the standard deviation decreases to the same level of 0.008 rad with about 25 OFDM frames if the phase offset is approximately constant over this time.

The pilot data can be used for finer-tuning the carrier frequency offset and Doppler spread after the estimated coarse carrier frequency offset has completed. For a particular system, the maximum Doppler spread-handling capability of the remaining calculations and the expected channel effects can be taken into account to balance the available correction at each stage of the receiver.

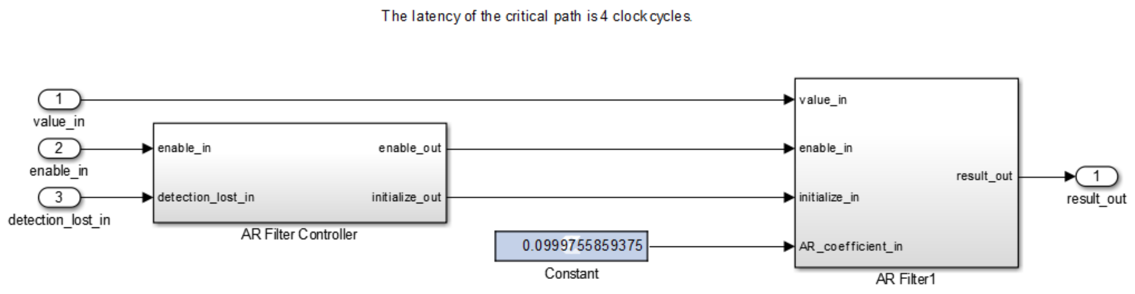
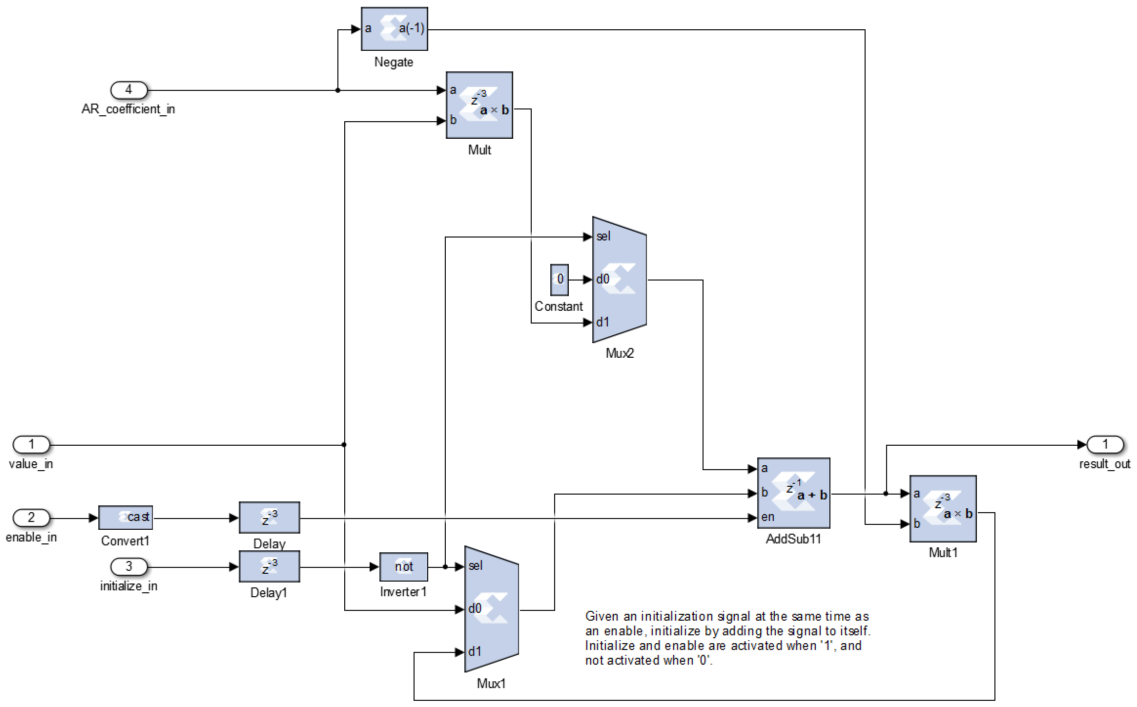


Figure 47: Overview of the AR filter in System Generator.



Given an initialization signal at the same time as an enable, initialize by adding the signal to itself. Initialize and enable are activated when '1', and not activated when '0'.

Figure 48: The System Generator block diagram of the AR filter datapath.

Chapter 5 Conclusion

The digital logic designs presented in this thesis synchronize a receiver to an OFDM frame and estimate a received signal's carrier frequency offset using only the cyclic prefix size and position in the frame. The correlation of cyclic prefix samples with their corresponding tail data samples produces a peak value which provides an effective synchronization time. The synchronization locks onto the signal as each correlation peak lies within a known window of time after the previous peak, and falls out-of-lock when the correlation peaks are not detected during this window. A Markov-chain model was created to predict the portion of time the receiver will be locked-on to an incoming OFDM frame at different SNRs. The real and imaginary parts of the correlation signal at the correlation peak time are used to estimate the phase offset between the tail data and cyclic prefix. This phase offset provides the coarse carrier frequency offset using the time difference between the samples. The estimated phase offset is smoothed by an AR filter and provided to later stages in the receiver. The design runs on the UMDCC's radio testbed with expected operation during a loop-back test and a board-to-board test. This design provides a flexible way to synchronize to an OFDM frame, estimate its coarse carrier frequency offset using only the cyclic prefix, and the design runs in practice on a radio testbed.

The next step to work with these ideas is to create the phase offset compensator, which uses the estimated phase to undo the carrier offset frequency's effect on the received signal. This can be done in an open loop or closed loop fashion. With the compensator completed, the radio design should be able to handle board-to-board OFDM transmission in a test environment.

References

[1] C. Schlegel et al., ”UMDCC Report 2014-2015”, Halifax, Canada, 2015, pp. 10-12.

[2] C. E. Shannon, ”A Mathematical Theory of Communication,” *The Bell System Technical Journal.*, vol. 27, pp. 37923, 62356, July, October, 1948.

[3] J. G. Proakis, M. Salehi, ”Fading Channels I:Characterization and Signaling,” *Digital Communications.*, 5th Edition, ch. 13, sec. 13.1-2, pp. 831.

[4] J. G. Proakis, M. Salehi, ”Fading Channels I:Characterization and Signaling,” *Digital Communications.*, 5th Edition, ch. 13, sec. 13.1-2, pp. 843.

[5] J. G. Proakis, M. Salehi, ”Characterization of Fading Multipath Channels,” *Digital Communications.*, 5th Edition, pp. 831-833.

[6] J. G. Proakis, M. Salehi, ”Characterization of Fading Multipath Channels,” *Digital Communications.*, 5th Edition, ch. 4, sec. 4.10-2, pp. 262.

[7] J. G. Proakis, M. Salehi, ”Optimum Receivers for AWGN Channels,” *Digital Communications.*, 5th Edition, ch. 11, sec. 11.2-8, pp. 757-758.

[8] J. G. Proakis, M. Salehi, ”Adaptive Equalization,” *Digital Communications.*, 5th Edition, ch. 10, sec. 10.5, pp. 721.

[9] C. Schlegel, ”OFDM Equalization and Channel Tracking,” Halifax, Canada, 2015, rev.1.2, pp. 5.

[10] J. G. Proakis, M. Salehi, ”Multiple-Antenna Systems,” *Digital Communications.*, 5th Edition, ch. 15, sec. 15.4-3, pp. 1014.

[11] J. G. Proakis, M. Salehi, ”Optimum Receivers for AWGN Channels,” *Digital Communications.*, 5th Edition, ch. 11, sec. 11.2-5, pp. 751.

[12] M. Jar, ”Frame Synchronization,” Halifax, Canada, 2015, rev.1.0, pp. 3.

[13] J. G. Proakis, M. Salehi, ”Carrier and Symbol Synchronization,” *Digital Communications.*, 5th Edition, ch. 5, sec. 5.2, pp. 295-313.

[14] J. G. Proakis, M. Salehi, ”Spread Spectrum Signals for Digital Communications,” *Digital Communications.*, 5th Edition, ch. 12, sec. 12.5, pp. 816-817.

[15] C. Marsh, ”Introduction to Continuous Entropy,” Dept. CSC., Princeton Univ., Princeton, NJ, U.S.A., 2013.

[16] S. Hauck and A. DeHon. ”Implementing Applications with FPGAs,” *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*, 1st ed. Burlington, MA, Morgan Kaufmann Publishers, 2008, ch. 21, sec. 21.4.2, pp. 449-452.

[17] E. Nash, ”Correcting Imperfections in IQ Modulators to Improve RF Signal Fidelity,” ADI, Norwood, MA, U.S.A., AN-1039, 2009.

[18] J. G. Proakis, M. Salehi, ”Carrier and Symbol Synchronization,” *Digital Communications.*, 5th Edition, ch. 5, sec. 5.3-1, pp. 318.

[19] L. Null and J. Lobur. *The Essentials of Computer Organization and Architecture*, 2nd ed. Sudbury, MA, Jones and Bartlett Publishers, 2006, ch. 4, sec. 4.2.1, pp. 178.

[20] S. Hauck and A. DeHon. ”Compute Models and System Architectures,” *Reconfigurable Computing: The Theory and Practive of FPGA-Based Computation*, 1st ed. Burlington, MA, Morgan Kaufmann Publishers, 2008, ch. 5, sec. 5.2.1, pp. 112.

[21] J. G. Proakis, M. Salehi, ”Deterministic and Random Signal Analysis,” *Digital Communications.*, 5th Edition, ch. 2, sec. 2.7-4, pp. 71-74.

[22] J.C.I. Chuang and N.R. Sollenberger, ”Burst coherent demodulation with combined symbol timing, frequency offset estimation, and diversity selection,” *IEEE Trans. Commun.* vol. 39, no. 7, pp. 1157-1164, Jul. 1991.

[23] S.M. Kay, ”Sinusoidal Parameter Estimation,” *Modern Spectral Estimation: Theory and Application*, 1st ed. Englewood Cliffs, N.J., Prentice Hall, 1988, ch. 13, sec. 2, pp. 408-411.

[24] S. Rajan, S. Wang, R. Inkol, and A. Joyal, ”Efficient approximations for the arctangent function,” *IEEE Signal Processing Mag.*, vol. 23, pp. 108-111, May, 2006.

[25] R.E. Goldschmidt, ”Applications of Division by Convergence,” M.S. Thesis, Dept. Elec. Engn., MIT, Cambridge, MA, 1964.