HARDWARE IMPLEMENTATION OF OVERLAP SAVE METHOD BASED FADING CHANNEL EMULATOR

BY

MUHAMMAD NAUMAN

01-244172-045

SUPERVISED BY

DR. ATIF RAZA JAFRI

Session-2017-2019

A Report submitted to the Department of Electrical Engineering

Bahria University, Islamabad

in partial fulfilment of the requirement for the degree of MS(EE)

CERTIFICATE

We accept the work contained in this report as a confirmation to the required standard for the partial fulfilment of the degree of MS (EE).

Head of Department	Supervisor
Internal Examiner	External Examiner

DEDICATION

I would like to dedicate this thesis to my parents, siblings, friends, supervisor and teachers for their love, endless support, and strong motivation which helped me in achieving my goals.

DECLARATION OF AUTHORSHIP

I hereby declare that content of this thesis is my own work and that it is the result of work done during the period of registration. To the best of my knowledge, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

(Student Signature)

ACKNOWLEDGEMENTS

First I like to thank Allah, without his grace I could not have achieved anything. I like to specially thank Dr. Atif Raza Jafri, my supervisor and teacher, for his advice, supervision, great support and guidance. I would also thank my Electrical Engineering department for their support and guidance.

ABSTRACT

Wireless communication systems require extensive and thorough evaluations before field trials and actual deployment. Difficulty in real time testing makes it the least favourable option for field trials prior thorough evaluation in the laboratory. It is therefore, channel emulators are used to evaluate a waveform at real time data rate while emulating a channel. Among various channel emulation techniques, Overlap Save (OLS) method used along with inverse Fast Fourier Transform (IFFT) technique provides the best computational efficiency. Hardware solutions based upon OLS method based technique are not available. Implementation of OLS method based technique on FPGA is novel and provides high throughputs as compared to software solutions. In this thesis an FPGA based channel emulator is implemented which emulates a Rayleigh Fading channel along with Doppler effects. An OLS-based fading variates generator and a time domain OLS-based interpolator are implemented which shown reduction in complexity when compared with originally proposed OLS method. A high throughput of 34 Mega Samples per Second (MSPS) was targeted, keeping in view maximum sampling frequency in LTE which is 30.72MHz for a 20 MHz configuration. Design with minimal resources was implemented and required throughput was successfully achieved on Virtex-7 FPGA. Designing of system was done in such a way that system can be used with multiple antenna systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

OLS	Overlap Save
IFFT	Inverse Fast Fourier Transform
MSPS	Mega Samples per Second
FPGA	Field Programmable Gate Array
LTE	Long-Term Evolution
OTA	Over-the-air
IP	Intellectual Property
MIMO	Multiple-Input and Multiple-Output
SOS	Sum of Sinusoids
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
GNG	Gaussian Noise Generator
BRAM	Block RAM

Chapter 1

Introduction

CHAPTER 1. INTRODUCTION

1.1. Thesis Background/Overview

With the advancement of technology and its underlying flexibility, digital signal processing has found its application in almost every field. Most of the old analog systems are replaced by new and specialized digital systems. Technological advancement is the key reason why new and unique applications are added every day. Internet, wireless communication and automation all are few applications of digital system.

Among all mobile communication generations, only 1st generation was based upon analog system, and only supported voice. With each upcoming generation, new and better service was provided. Within 5th generation, three communication services are being addressed: *Extreme Mobile Broadband* (xMBB), *Massive Machine-Type Communication (M-MTC)* and *Ultra-reliable/Critical Machine-Type Communication (U-MTC)* [1].

Before deployment of a wireless communication system, an evaluation is required for successful communication. Ideally real time testing is the best solution for practically testing systems with various scenarios, but it requires high cost, more effort and is time consuming. With already available channel models, there is no or minimum requirement of field measurements. Field measurements are only needed when channel model is not available, and in this case real time field tests in different environments are needed.

A channel emulator imitates the real time channel by emulating it on hardware/software. Channel emulation is performed for evaluation of a system performance in real scenarios before commercial deployment, but commercially available channel emulators are expensive and not flexible when configuring various channel parameters [2].

Figure 1.1 illustrates a typical radio environment, with signal following various paths. Such kind of propagation is known as multipath propagation. Multipath effect can cause signal interference, as a previous symbol can merge in next symbol if proper delay has not been given between symbols.

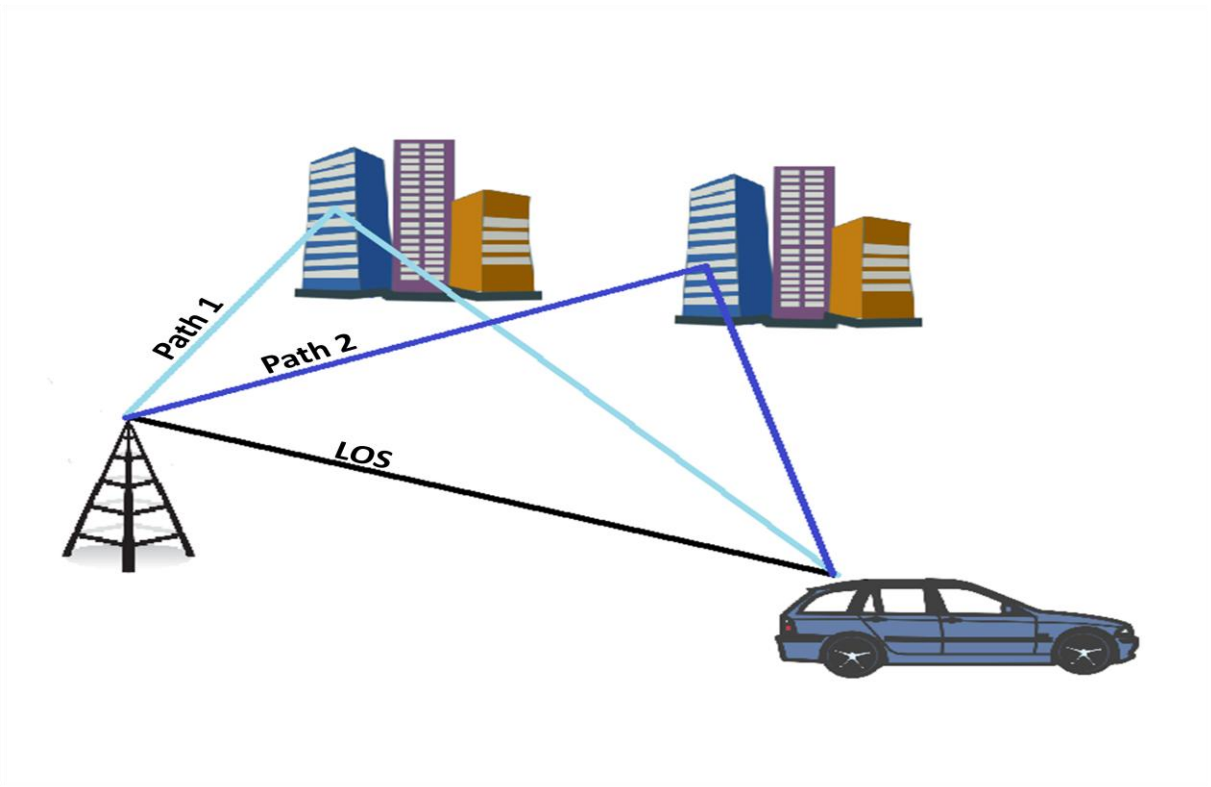


Figure 1.1: Typical Radio Environment

Channel emulator can ‘emulate’ the environment or channel present between both transmitter and receiver. Figure 1.2 displays testing in real environment, also called over-the-air (OTA) field trials [3]. Figure 1.3 is the case in which devices are under test with channel being emulated by channel emulator.

Transmitter For Testing
Receiver For Testing

Figure 1.2: OTA Field Trials

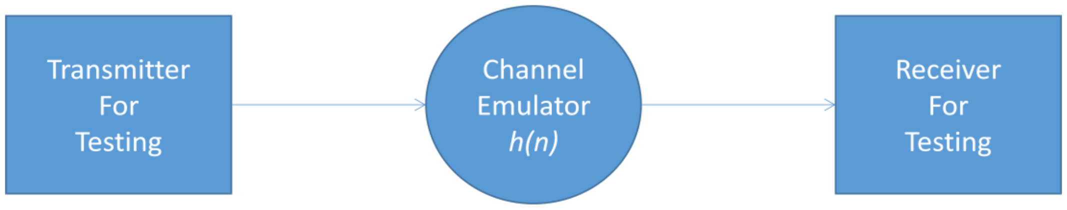


Figure 1.3: Channel Emulation Model

In Figure 1.3 ‘*h (n)’* is the composite channel response generated by channel emulator, using various channel models.

While implementing a channel emulator, certain factors are required to be considered. One of them is fading. Fading is the attenuation in a signal due to various factors. These factors can be environmental conditions, multiple paths or many others. Multiple copies arrive at receiver known as *multipath waves*, and add up to form a signal which varies widely in phase and amplitude [4]. Fading can also be defined as rapid fluctuations in a signal, over a short time period or distance [4]. Figure 1.4 shows types of fading [5].

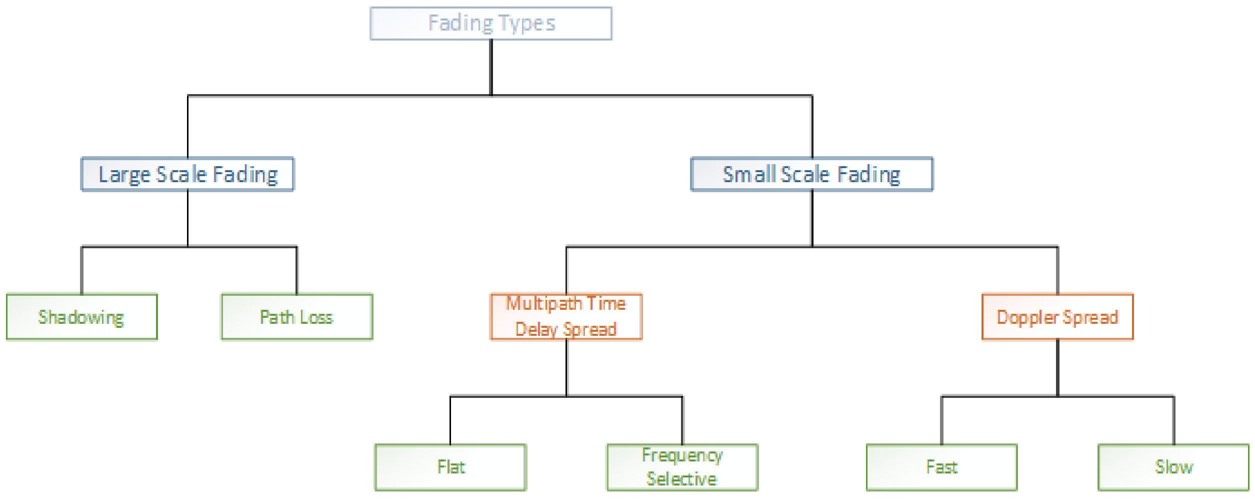


Figure 1.4: Fading Types

1.1.1. Large Scale Fading

Large scale fading is generally simpler to model; it is due to large distance and large objects between transmitter and receiver. It is further divided into *Path Loss* (*Free Space Path*

Loss) and *Shadowing.* Path Loss is proportional to squared distance between transmitter and receiver.

$$FSPL = \left(\frac{4\pi d}{\lambda}\right)^2$$

FSPL is linear and is simpler to model; *Shadowing* is due to the large obstacles present in between transmission medium [5].

1.1.2. Small Scale Fading

Small Scale Fading is more complex to model; large power variations are caused by small changes. It is also called as Rayleigh Fading (if no Line-of-Sight (LOS) is present), as the received signal's envelope is described by probability distribution function (pdf) of Rayleigh distribution [5]. If a dominant LOS component is present, received signal's envelope is described by Rician Fading [5].

Delay Spread and *Coherence Bandwidth B_c* defines how signal is dispersed in time [4]. *Delay Spread* can be defined as the difference between first and last received copy. *Delay Spread* is an important parameter, as for a channel without ISI; symbol duration should be greater than it [6]. Power Delay Profile (PDP) is a function of time delay $A_c(\tau)$, with which *Delay Spread* can be found. Most common *Delay Spread* is *rms Delay Spread* σ_τ. *Mean Delay Spread* $\bar{\tau}$ *and rms Delay Spread* is defined as [6]:

$$\bar{\tau} = \frac{\int_0^\infty \tau A_c(\tau) d\tau}{\int_0^\infty A_c(\tau) d\tau}$$

$$\sigma_\tau = \sqrt{\frac{\int_0^\infty (\tau - \bar{\tau})^2 A_c(\tau) d\tau}{\int_0^\infty A_c(\tau) d\tau}}$$

Coherence Bandwidth B_c defines how much channel is 'flat' which means spectrum range in which all of the channel response is same. *Coherence Bandwidth B_c* is approximated by the following equations, depending upon [4]:

$$B_c = \frac{1}{50\sigma_\tau}$$

$$B_c = \frac{1}{5\sigma_\tau}$$

First equation is the case when frequency correlation function is above 0.9, and second equation is the case when frequency correlation function is above 0.9 [4].

Fading due to the relative motion of receiver, transmitter or due to the movement of objects in the channel cannot be described by *Delay Spread* or by *Coherence Bandwidth*; however it is described by *Doppler Spread* and *Coherence Time* [4]. *Doppler Spread* denoted by B_D is a measure of how much motion of receiver or transmitter caused broadening of spectrum in received signal. *Doppler Spread* causes received signal's bandwidth to increase by $2f_d$, where f_d is the *Doppler Shift* [4].

Doppler Shift depends upon the relative velocity of transmitter or receiver, and also the angle Θ between them. *Doppler Shift* is defined by [4]:

$$f_D = \frac{v}{\lambda}\cos\theta$$

λ denotes wavelength of carrier frequency and v denotes the speed of mobile. For maximum *Doppler Shift*, f_m:

$$f_m = \frac{v}{\lambda}$$

A channel is known as *slow fading* if baseband signal's bandwidth is much greater than *Doppler Spread* [4]. In this case effect of *Doppler Spread* can be ignored.

Coherence Time T_C and *Doppler Spread* are inversely related. *Coherence Time* shows the effects of frequency dispersion in time domain, maximum *Doppler Spread* is also known as *fading rate* of the channel [5].

$$T_C \approx \frac{1}{f_m}$$

One way to define *Doppler Spread* is [5]:

$$D(f) = \frac{1}{\sqrt{1 - \left(\frac{f}{f_m}\right)^2}}$$

For a carrier frequency of 600MHz, Figure 1.5 shows the *Doppler Spectrum* for 30 Km/hour and 60 Km/hour speeds.

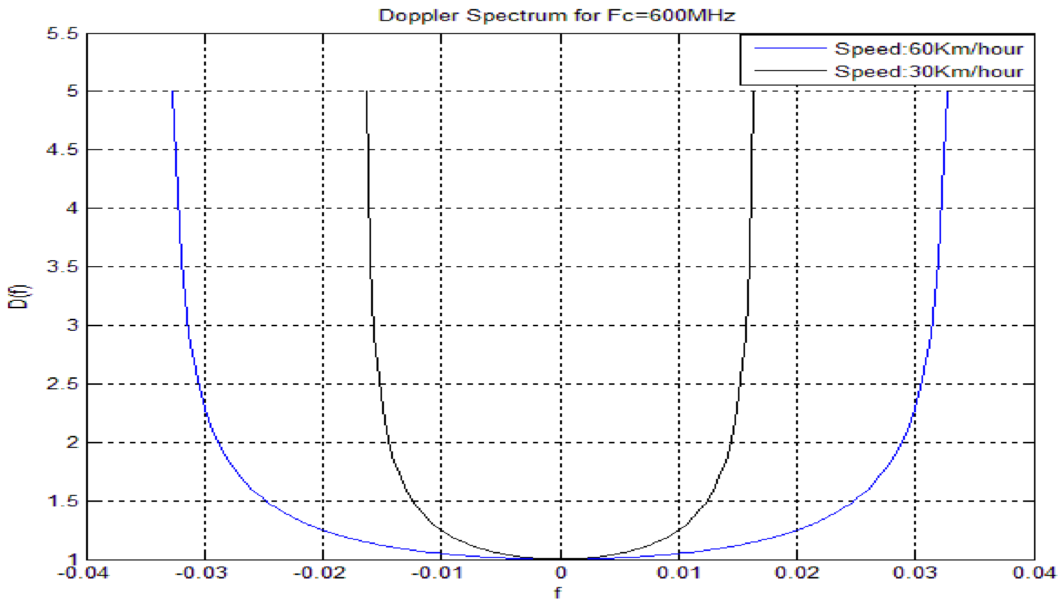


Figure 1.5: Doppler Spectrum

1.1.2.1. Flat & Frequency Selective Fading

When bandwidth of channel is greater than baseband signal's bandwidth, signal undergoes *flat fading*. All frequency components of signal experience same channel effects. The following equation summarizes whether a signal is affected by *flat fading* channel or not:

$$B_s \ll B_c$$

B_s and B_c are the bandwidth of signal and channel. *Flat Fading* channels can also be called as *narrowband channels* [4]. Transmitting more power helps in reducing bit error rates caused by *flat fading*.

Frequency selective fading occurs when the bandwidth of transmitted signal is greater than channel's bandwidth. In such scenario, different frequency components are affected differently. Received signal contains multiple instances of transmitted signal. *Frequency selective fading* is difficult to model and mitigate as compared to *Flat Fading* [4]. The following equation summarizes whether *Frequency Selective Fading* will distort a signal or not:

$$B_s > B_c$$

Figure 1.6 differentiates between *Flat and Frequency Selective Fading.*

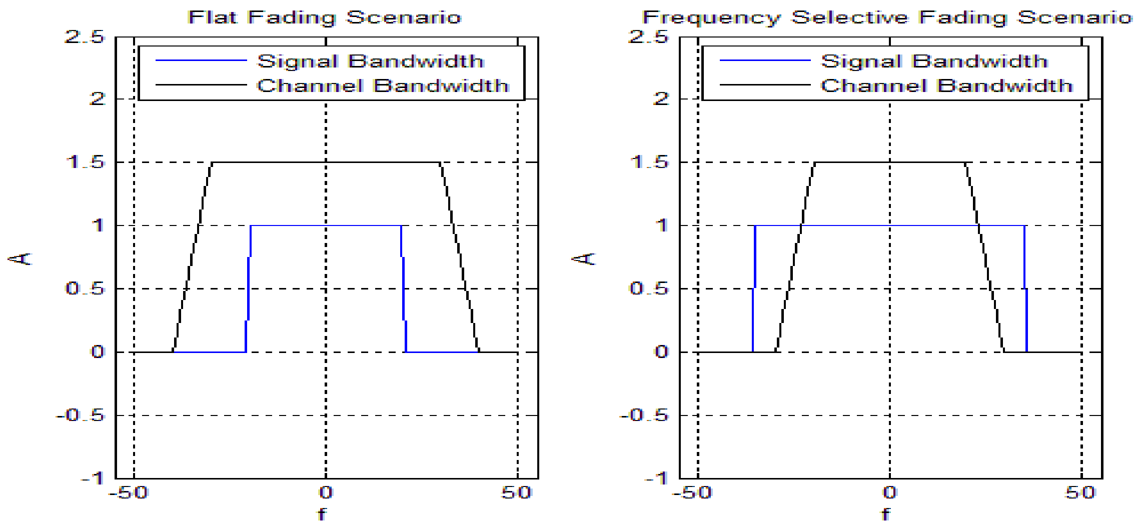


Figure 1.6: Frequency Selective vs. Flat Fading

1.1.2.2. Fast & Slow Fading

Fast and *slow fading* are classified on the basis of *Doppler Spread*. In *Fast Fading*, symbol duration of transmitted signal is greater than *Coherence Time* [4]. *Time Selective Fading* is the frequency dispersion, caused by *Doppler Spread*. *Fast fading* occurs when [4]:

$$T_s > T_c$$

$$B_s < B_D$$

B$_s$ and B$_D$ are the bandwidth of signal and *Doppler Spread*. T$_s$ and T$_c$ are symbol duration and coherence time. *Fast fading* usually occurs for very low data rates [4].

Slow fading channel is the one in which bandwidth of transmitted signal is much larger than *Doppler Spread*. In this case channel is almost static, as symbol duration is much lesser than *Coherence Time* [4]. Again summarizing *slow fading*:

$$T_s \ll T_c$$

$$B_s \gg B_D$$

Figure 1.7 summarizes all the flat, fast, frequency selective and slow fading.

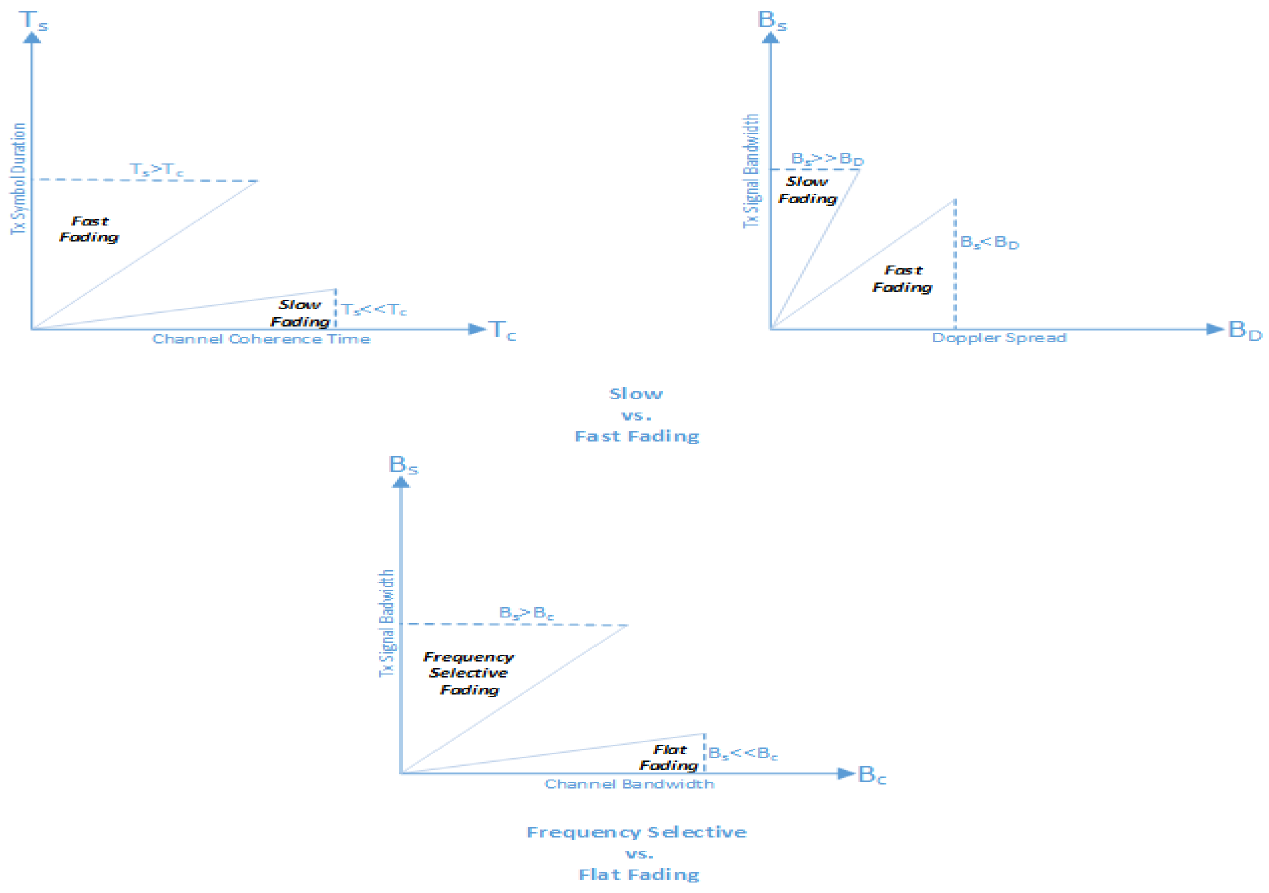


Figure 1.7: Small Scale Fading

1.1.3. Rayleigh Fading

Rayleigh distribution is evident in envelope of mobile radio signal's amplitude [7]. *Rayleigh Fading* is suitable in an environment, where Line of Sight (LoS) component is absent (practical scenarios). Rayleigh Distribution is the sum of two uncorrelated Gaussian Noise signals. A Gaussian Noise generator, fulfilling properties for Gaussian distribution can help in generation of *Rayleigh Fading*. Figure 1.8 illustrates pdf of *Rayleigh Distribution*.

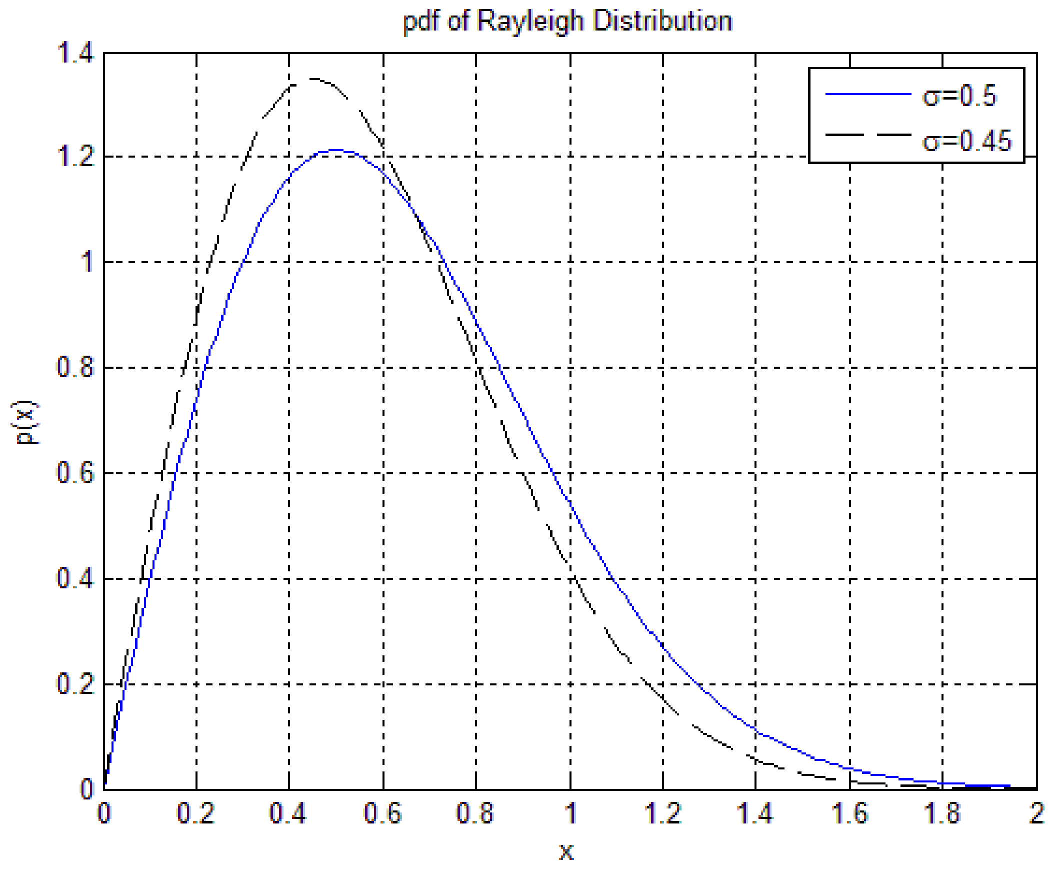


Figure 1.8: Rayleigh Distribution

Rayleigh Fading is very important as it is a practical model for signals affected heavily by urban environments [5]. Emulation of *Rayleigh Fading* with compensating *Doppler Effect* needs a random process generation [8]. With current telecommunication scenario, high throughput is necessary, which requires high speed and specialized hardware. *Field Programmable Gate Array (FPGA)* is one of the solutions for high throughput demanding applications.

1.1.4. Overlap Save Method

Overlap Save (OLS) method is generally used to perform convolution efficiently between a long sequence and a shorter one. This method splits long sequence into smaller blocks and computes convolution. Time domain convolution can be done in frequency domain by simple multiplication and this is the advantage which OLS method utilizes. Generation of fading variates can require convolution with a specific filter, which can be implemented using this technique [8]. Figure 1.9 shows the OLS method, which shows how long sequence is split into parts by following equation:

$$N = L + M - 1$$

Where N is the Fast Fourier Transform (FFT) size and L is selected filter length. Also L is the output valid value to be used. N can be fixed according to the requirements, and M or L can also be chosen. Without OLS method a very large number of FFT size is required, and this increases the computation complexity and memory requirements drastically.

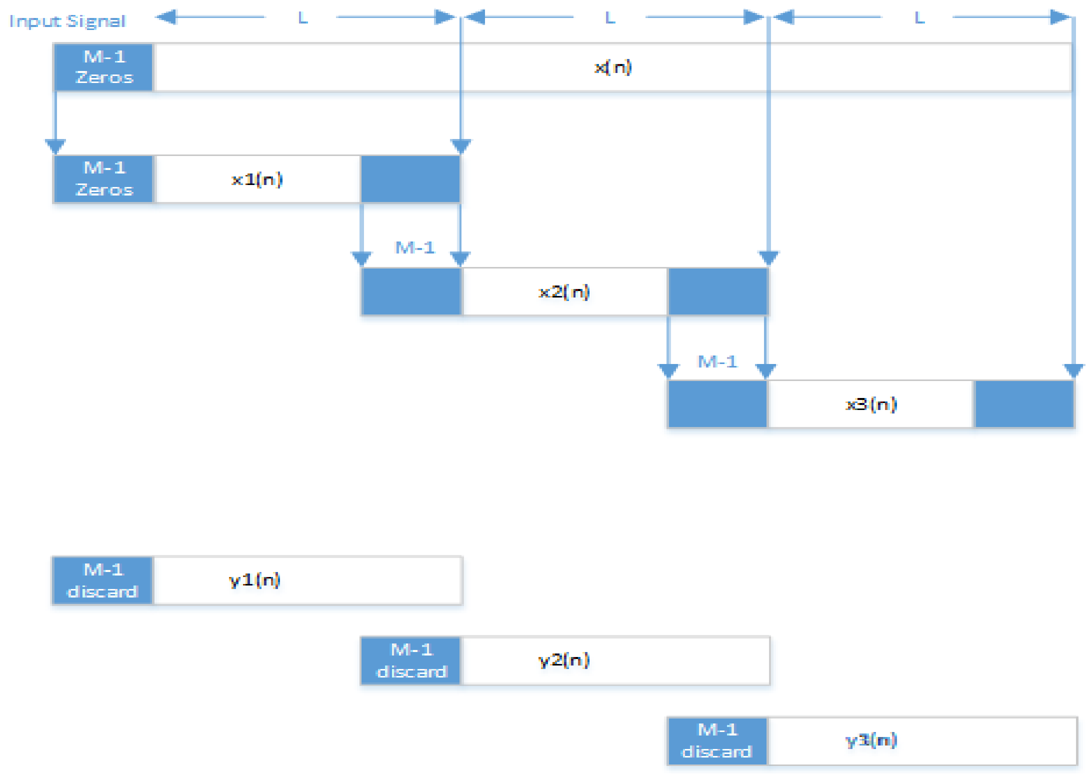


Figure 1.9: OLS Method

1.1.5. Multiple-Input and Multiple-Output (MIMO) System

Multiple antennas at transmitter and receivers can be used to increase overall system data rate and performances [6]. Incorporating multiple antennas is the key principle of MIMO. Using multiple antennas at both transmitter and receiver has proven to provide more efficiency as compared to systems incorporating multiple antennas only at either transmitter or receiver [6]. A typical 2×2 MIMO system is shown in Figure 1.10.

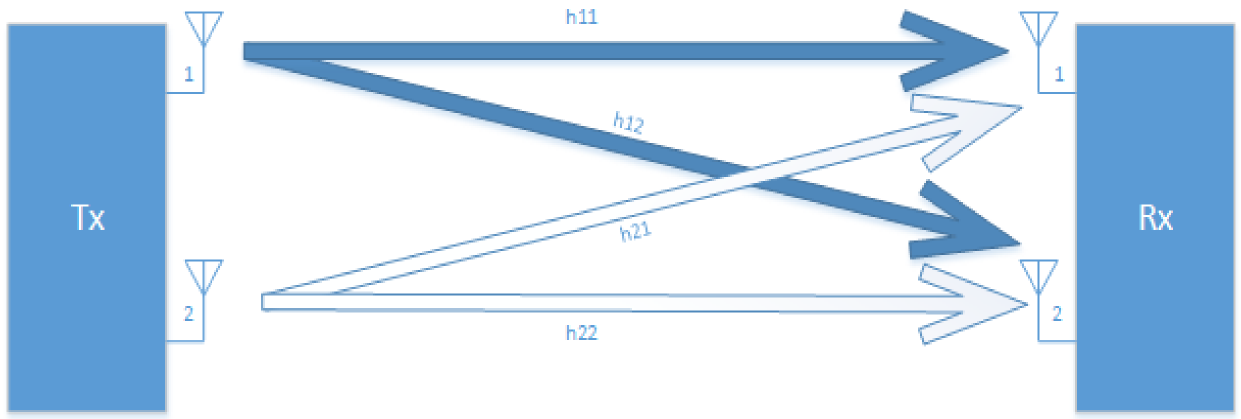


Figure 1.10: MIMO System

1.1.6. Digital System Design and FPGA

With the advancement in digital system design, fast and efficient implementation of various algorithms can be done. An FPGA can be configured in order to perform a specific task, however if a different task is required, FPGA can be reconfigured to perform a new and different task. Up gradation and flexibility in FPGAs are more than ASICs, DSP processors and Microcontrollers. FPGAs are usually used where there is a necessity of a single device performing a task (not good for bulk production), however FPGAs can be used for ASIC prototyping before production, and ASIC testing can be done on FPGA kits. Selection of FPGA for Channel Emulator implementation is adequate for achieving higher output sample rate, good performance and accurate results, when compared to other implementations.

1.2. Problem Description

Over-the-air (OTA) field trials are expensive and require manpower to test a device in real time scenario. With the arrival of 5G technology in 2020, high throughput intensive applications will soon be introduced. These high throughput intensive applications' hardware cannot be merely tested on the basis of simulation results. Also channel emulators based upon microcontroller or Digital Signal Processing (DSP) processor might not be able to achieve required high throughput. For targeted 20MHz LTE channelization system (the most demanding case), an emulator providing 34 Mega Symbols per Second throughput per transmitter receiver path is required. This high throughput is impossible to be achieved through software using a single instance of a DSP or other processor. The situation worsens in case of MIMO scenario where a total of 36 paths are required for a 2×2 MIMO system.

1.3. Thesis Objectives

As mentioned above, software solutions do not fulfil the throughputs of a channel emulator for 5G applications and hence a hardware based solution is required. For those applications where less quantity user specific ICs are required, FPGA provide the ideal platform for implementation. Hence, the aim of this thesis is to propose and implement OLS method [8] based channel emulator on FPGA. A basic hardware IP will be developed to emulate on path between transmitter and receiver. The design will be able to achieve a maximum throughput of 34 MSPS using minimum hardware resources. The IP will be designed while keeping the scalability into account and design will enable usage of multiple instances of IP for 2×2 MIMO configuration and 9 total paths for multipath effects (i.e. 36 instances of IP).

1.4. Thesis Organization

Thesis is organized as follows: Chapter 2 is dedicated for literature review. State of the art channel emulators are discussed and associated techniques that are used in generation of fading variates are analyzed. Since FFT/IFFT is one of the fundamental operations used in channel emulators. FFT techniques are reviewed, along with the time domain and frequency domain interpolation techniques.

Chapter 3 describes the used methodology in the thesis. Proposed system, various methods for implementation, timing requirements and efficient resource utilization, are discussed in this chapter.

Chapter 4 discusses the synthesis and implementation results on FPGA devices. Resource utilization along with a comparison among different implementation solutions discussed in Chapter 3 is also presented.

Finally, Chapter 5 presents the conclusion and future directions in this domain.

Chapter 2

Literature Review

CHAPTER 2. LITERATURE REVIEW

2.1. Channel Emulation

Many channel emulators are commercially available [9, 10, 11]. The commercially available solutions are efficient, provide excellent channel emulation and are scalable. Commercial products targets many different channels and user scenarios however these are too much expensive as simplest of these products starts from $6000. A cost effective solution is very suitable for anyone which intends to perform channel emulation. An FPGA based solution can target a specific mobile communication channel, and can be reconfigured to a new channel when needed.

There are mainly two methods in generation of Rayleigh Fading samples:

- Sum of Sinusoids (SOS) [12] or Jake's Model [7].
- Filtering Gaussian Noise (FIR based [13] or IIR based [14])

Sum of Sinusoids (SOS) method is based upon the idea that received signal is made up of various sinusoids. Received signal $R_D(t)$ is a superposition of waves, and is expressed as [7]:

$$R_D(t) = E_0 \sum_{n=1}^{N} C_n \cos(\omega_c t + \omega_m t \cos A_n + \Phi_n)$$

N is the number of paths. Amplitude of transmitted wave is denoted by E_0, attenuation of the nth path is represented by C_n which is a random variable. Carrier and maximum Doppler frequencies are denoted by ω_c and ω_m. Phase shift is denoted by Φ_n. Jakes fading channel simulator is depicted in Figure 2.1.

SOS method in real time requires large computational complexity and sinusoidal function has inherent property which introduces correlation in between samples [8].

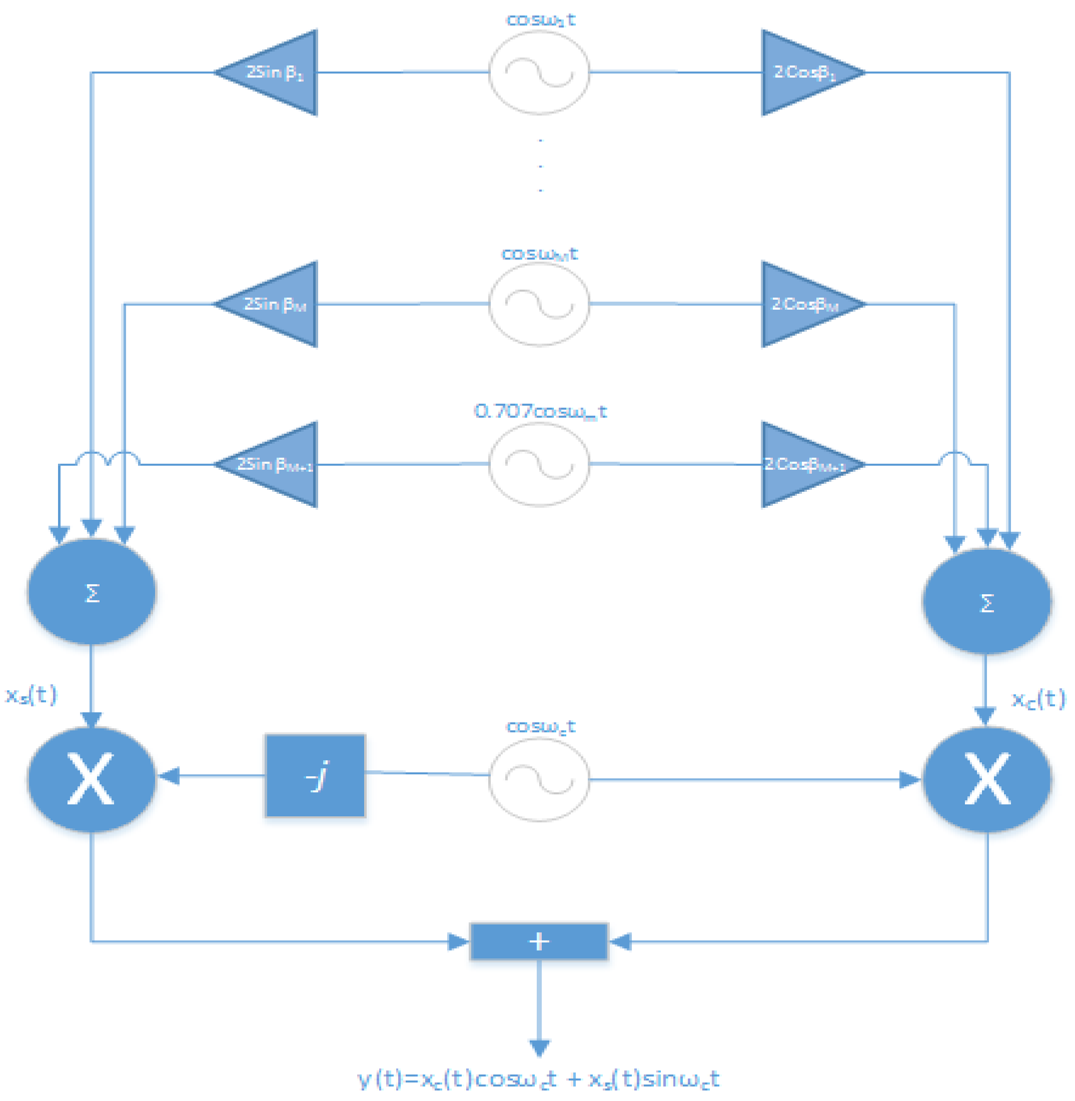


Figure 2.1: SOS Method for Generation

Second method i.e. filtering the Gaussian noise requires FIR [13] or IIR filtering [14], in which FIR based filtering requires large number of taps, and IIR suffers from stability problem. Both the methods require values or filter related to Doppler spectrum as filtering is done with Doppler filter.

Another method exists, which is in actual implementation of second method, is to use FFT and IFFT operation to perform filtering [15, 16]. First time this method was proposed in a computer program written in FORTRAN [15]. In FFT/IFFT method Gaussian noise is multiplied with square root of frequency response of Doppler filter, and then performs IFFT to get final Rayleigh Fading variates. Modification to the original filter coefficients can reduce FFT/IFFT operations to half of the original, and reduces memory requirements [16]. Original Doppler filter sequence is U-shaped [7]:

$$S(f) = \begin{cases} \dfrac{1}{\pi f_d \sqrt{1 - \left(\frac{f}{f_d}\right)^2}}, & if |f| \le f_d, \\ 0, & else \end{cases}$$

Where f_d is the maximum Doppler frequency. Modified Doppler filter coefficients are defined by [16]:

$$F_D(\omega) = \begin{cases} \sqrt{\dfrac{w_m \pi}{4} + \dfrac{w_m}{2} \tan^{-1}\left(\dfrac{w_m - 1}{\sqrt{2w_m - 1}}\right)}, & if \omega = w_m, (N - w_m), \\ \dfrac{1}{\sqrt{2}}\left(1 - \left(\dfrac{\omega}{N f_m}\right)^2\right)^{-\frac{1}{4}}, & if 1 \le \omega < w_m, or\ (N - w_m) < \omega < N, \\ 0, & else \end{cases}$$

Normalized Doppler frequency is represented by$f_m = f_d/f_s$, N is the FFT size and $w_m = f_m N$. Figure 2.2 shows the IFFT based implementation [16].

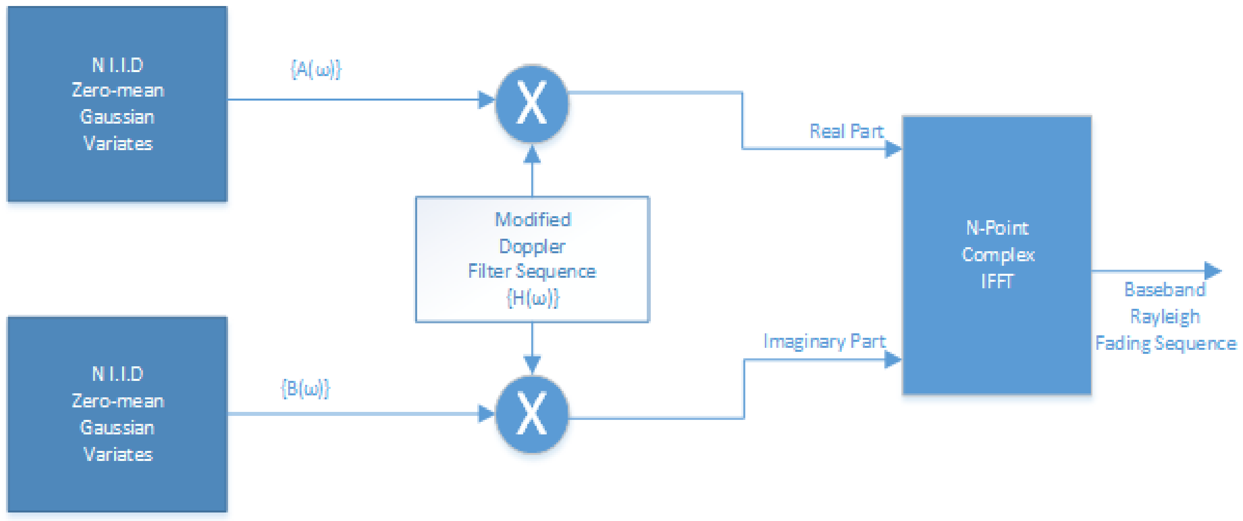

N I.I.D Zero-mean Gaussian Variates
{A(ω)}
X
Real Part
Modified Doppler Filter Sequence {H(ω)}
N-Point Complex IFFT
Baseband Rayleigh Fading Sequence
Imaginary Part
N I.I.D Zero-mean Gaussian Variates
{B(ω)}
X

Figure 2.2: IFFT Method for Generation

FFT/IFFT method is best, compared to the above two methods (SOS and filtering) in terms of computation complexity. Flaw of this method is the discontinuity issue in IFFT, and Doppler variates are required to be stored in advance, due to block-oriented nature of IFFT [8]. Also for high throughput based variates generation, a large size of FFT is required. Further MIMO based system will require multiple instances of single module, and this will make implementation with this method impossible.

With the above mentioned issue, FFT/IFFT method cannot find its application in real time systems with high throughput and long sequences. However there exists a method which can be utilized to perform FFT/IFFT method in real time systems [8]. OLS method along with an interpolator can promise low complexity based real time channel emulation. This method helps in generating *Rayleigh Fading* variates, and interpolator (either implemented in Time or Frequency domain) helps in reducing FFT/IFFT size. Originally proposed method for fading variates generator in [8] is shown in Figure 2.3.

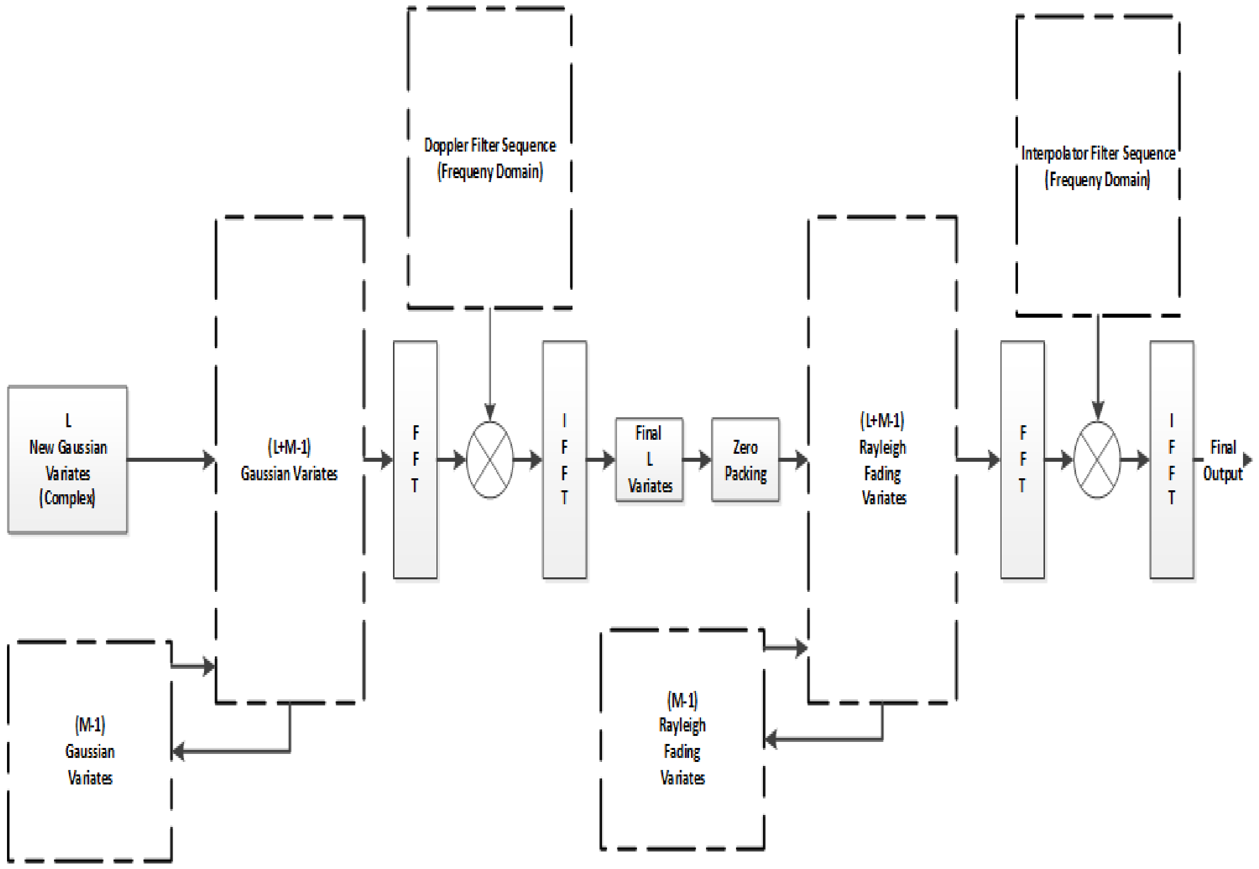

Doppler Filter Sequence (Frequeny Domain)
Interpolator Filter Sequence (Frequeny Domain)
L New Gaussian Variates (Complex)
(L+M-1) Gaussian Variates
FFT
IFFT
Final L Variates
Zero Packing
(L+M-1) Rayleigh Fading Variates
FFT
IFFT
Final Output
(M-1) Gaussian Variates
(M-1) Rayleigh Fading Variates

Figure 2.3: Original OLS-based Channel Emulator

Various FPGA based implementations can be found in literature. Authors in [17] used U- shaped *Doppler Spectrum* and optimized hardware consumption. Single FPGA chip based MIMO implementation with multipath is done in [18]. Elliptic IIR based interpolation on FPGA was implemented in [19]. Authors in [20] generated 25 MSPS, with using IIR based filtering (spectrum filtering and interpolation). Memory consumption is optimized by eight times in [21]. In [22] Sum-of-frequency-modulation (SoFM) method was introduced in a MIMO system, which is a new technique in hardware implementation. An SOS method based fading generator was proposed in [23].

2.2. FFT/IFFT Methods

Various FFT/IFFT techniques can be used. Either Decimation-in-Time (DIT) or Decimation-in-Frequency (DIF) can be used [24]. DIT and DIF implementation in radix-2 FFT is shown in Figure 2.3.

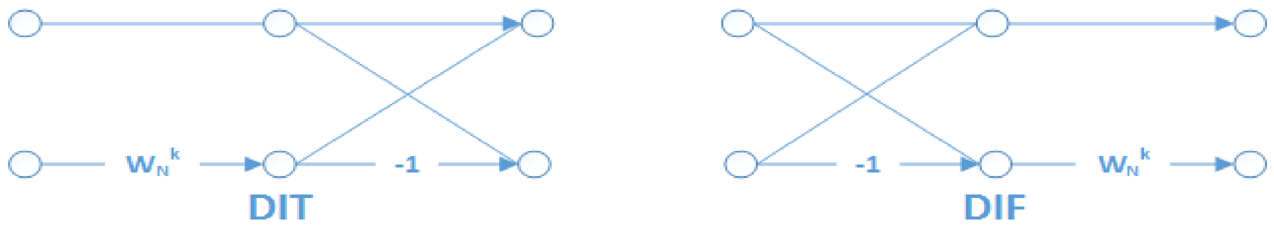


Figure 2.4: Decimation in Time and Decimation in Frequency

Either one of radix-2, radix-4, mix radix or radix-2^2 [25] can be used in FFT/IFFT block. Radix-2^2 utilizes radix-2 structure but has complexity of radix-4. Radix-2 can be used to implement any size of N, whereas radix-4 is useful where N is in power of 4. Computational complexity is reduced in radix-4, however its structure and implementation is difficult.

Various FFT architectures are summarized in [26]. There are three main techniques for hardware implementation of FFT:

- **Direct Implementation**: Data Flow Graph (DFG) is implemented on hardware as it is. This technique is efficient for small size of N; however it is not practically suitable for systems with large FFT size.

- **Memory-Based FFT Architectures**: Reutilizes one or many butterflies, and aim of this approach is to increase butterfly optimization [26]. It is further divided into two subdivisions:
    - **Single Memory-Based**: Utilizes single memory for inputs and intermediate outputs of FFT, data is fed into Processing Element (PE) which is made of a single or multiple butterflies (BF) from memory and results are fed back. Several iterations of the process generate results. This approach cannot be used when input data is streaming, however if high speed is not required this architecture is feasible. Single Memory-Based architecture is displayed in Figure 2.4 [26].

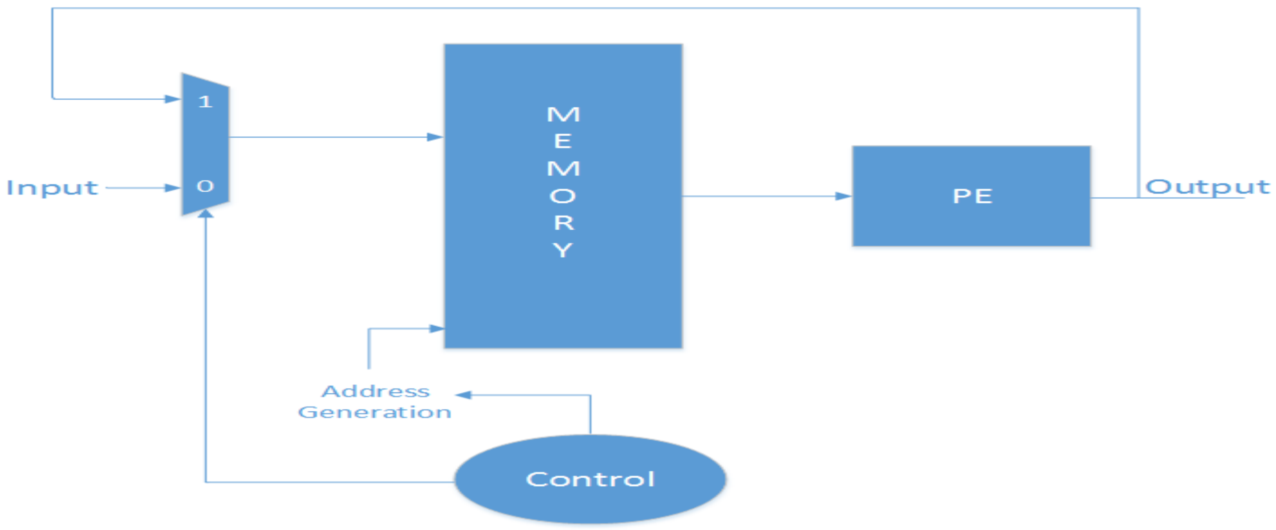


Figure 2.5: Single Memory-based FFT Architecture

    - **Dual Memory-Based**: Utilizes two separate memories and a PE. Both memory switch roles as input and output, after each FFT stage switching of input and output is done. This approach also cannot be used with continuous stream of data. Dual Memory-Based architecture is displayed in Figure 2.5 [26].

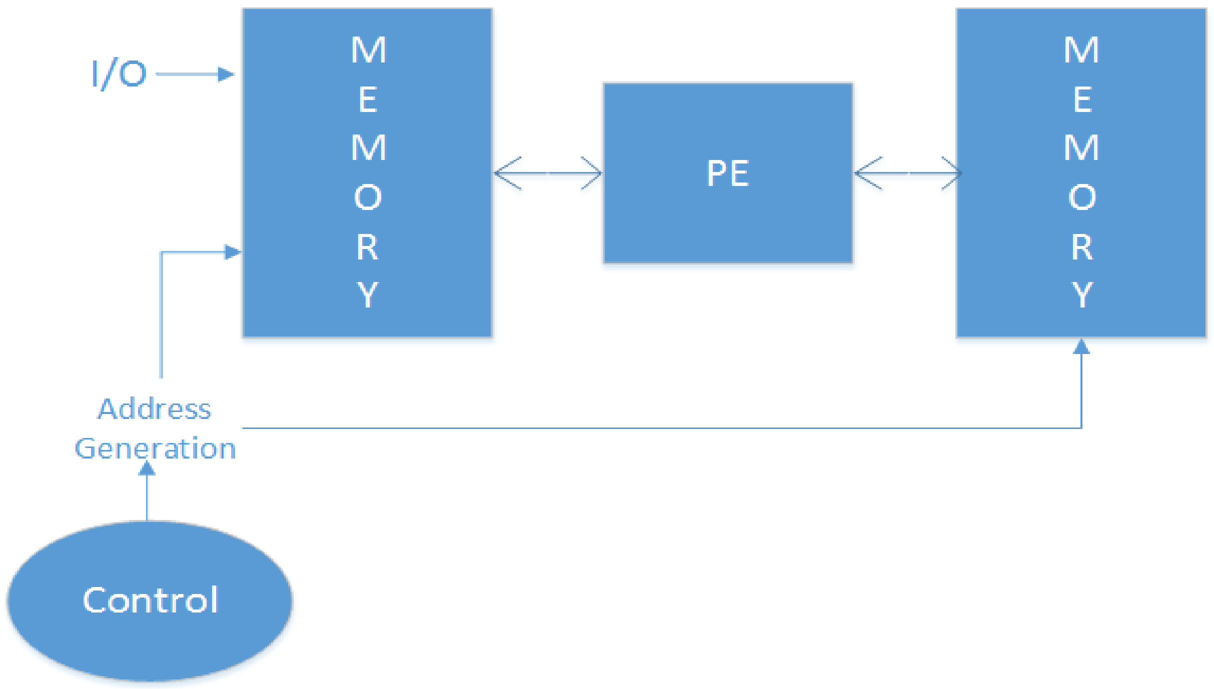


Figure 2.6: Dual Memory-based FFT Architecture

- **Pipelined FFT Architectures**: A high resource consuming approach, but the only feasible approach which incorporates streaming data and provides best throughput. It is also called streaming architecture [26] and is highly sophisticated in terms of its utilization of structure and simple control. It utilizes parallelism between multiple stages. Incorporating pipeline registers between each stage can increase throughput even more. A BF operation is done in this architecture and then twiddle factors are multiplied. A single BF is dedicated to a specific stage of DFG, and log_2N BFs are needed. It takes (N/2+1) cycles to start the process and latency of it is 2N-1. It is further divided into ***Feedback*** and ***Feedforward*** systems. Figure 2.6 displays a generic diagram for Pipelined FFT architectures [26].

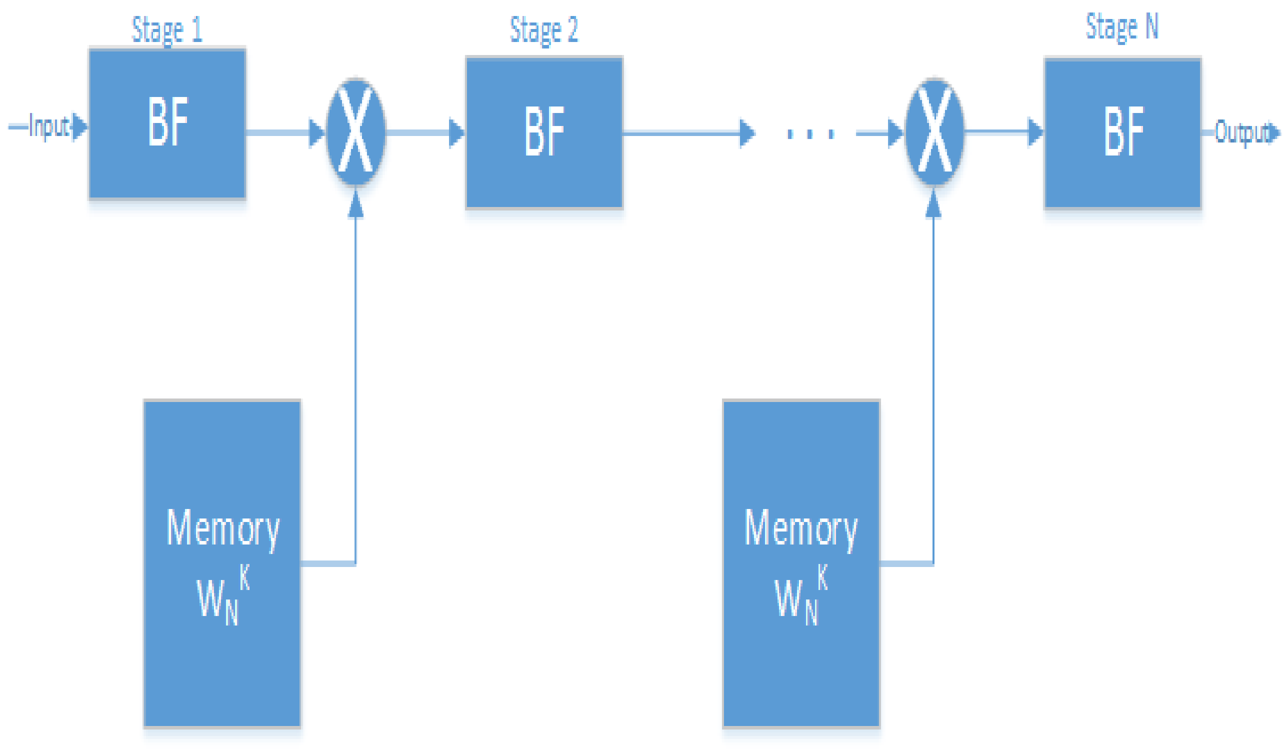
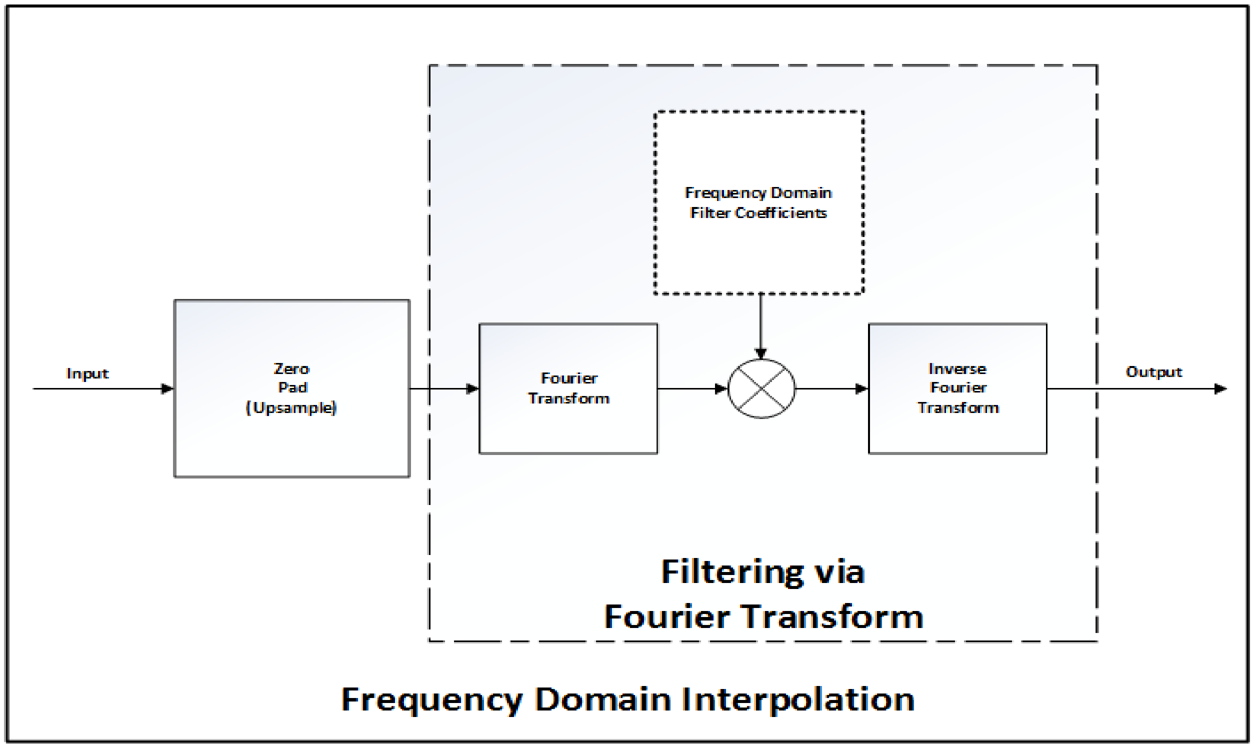


Figure 2.7: Pipelined FFT Architecture

2.3. Time Domain vs. Frequency Domain Interpolation

With all above FFT architectures with their advantages and disadvantages, interpolation is necessary due to the fact that in practical scenarios Doppler frequency is very low as compared to sampling frequency. Interpolation process requires upsampling (zero insertion) followed by filtering. Filtering can be performed in either frequency domain or time domain. Time domain filtering suffers with zero multiplication, most of the taps contain zero values and in such a case frequency domain based filtering is preferred. Interpolation in frequency domain requires additional FFT block and additional memories. FFT involves various multiplications and storage in between stages so additional hardware cost occurs in FFT (in terms of complexity). Figure 2.8 displays the frequency domain and time domain interpolation.



Frequency Domain Filter Coefficients
Input
Zero Pad (Upsample)
Fourier Transform
Inverse Fourier Transform
Output
Filtering via Fourier Transform
Frequency Domain Interpolation

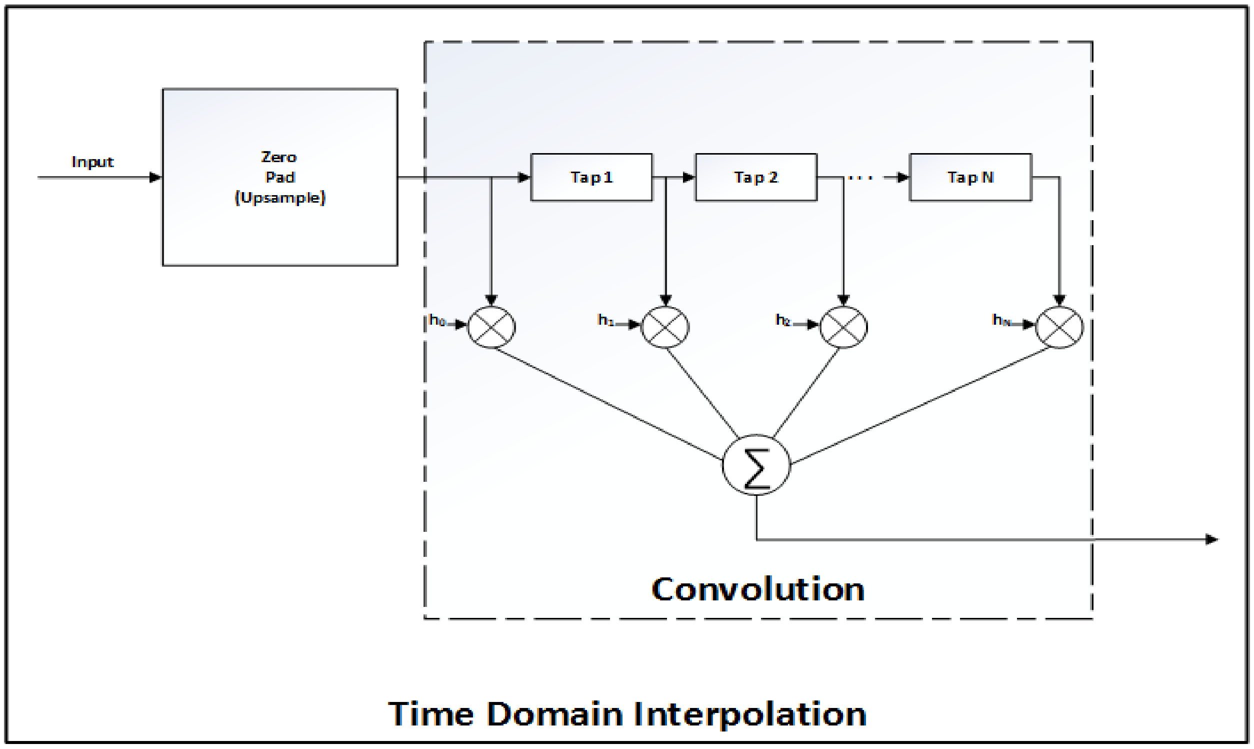


Figure 2.8: Interpolation in Time and Frequency Domain

An efficient time domain filtering on FPGA can reduce cost of additional FFT/IFFT hardware as fewer multipliers can be utilized by allowing only non-zero multiplications [27]. Filtering technique in [27] is shown in Figure 2.9. OLS based interpolator in frequency domain [8] can be done using time domain technique [27] with slight modification to achieve high throughput.

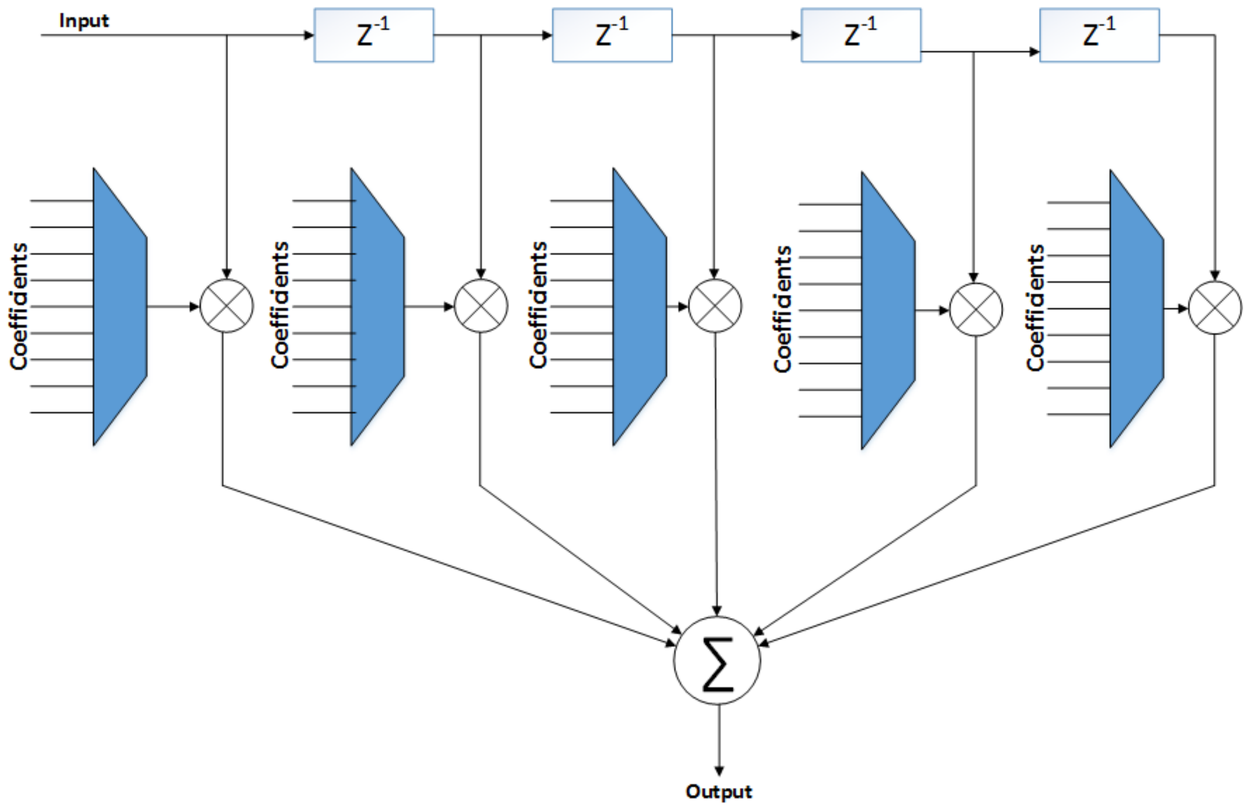


Figure 2.9: Efficient Time Domain Filtering

In this thesis fading variate generation is done with using technique in [8] with many modifications. A high throughput was targeted and MIMO system had to be incorporated. For multiple antenna systems and incorporating multiple paths, one needs multiple instances of a single module with different input parameters. A need of time domain interpolation was seen in [8]. Zero padding, memory storage for complex interpolator sequence, additional memory for OLS method and additional FFT/IFFT hardware consumes more resources. Resources required by system increases a lot, so a time domain interpolation with real valued coefficients of interpolator sequence can ensure significant reduction in hardware complexity. Time domain interpolation is done with modification to the technique in [27].

Chapter 3

Methodology

CHAPTER 3. METHODOLOGY

Generation of *Rayleigh Fading Variates* with *Doppler* effects and along with MIMO path can be implemented with various techniques. FPGA based implementation requires an in depth analysis before implementation, as FPGA provides fast processing, but it has limited number of resources. For this thesis a MIMO 2×2 based channel emulator was targeted. In each of the antenna's path nine different paths were considered. So for this system 36 modules are required which will generate fading variates.

Top level diagram of [8] is illustrated in Figure 3.1.

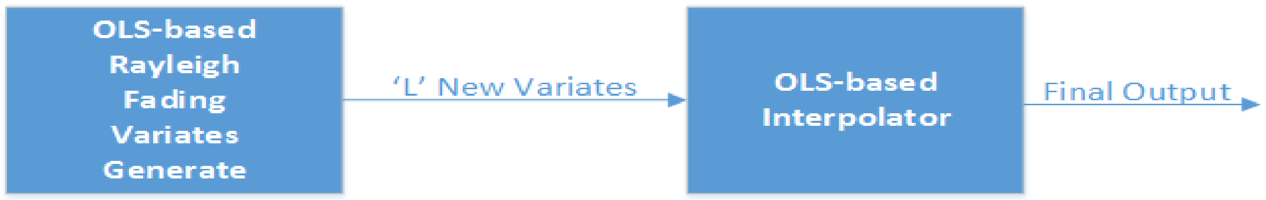


Figure 3.1: Original OLS method based Interpolator's Top Diagram

OLS-based generator and interpolator proposed in [8] are illustrated in Figure 3.2 and 3.3.

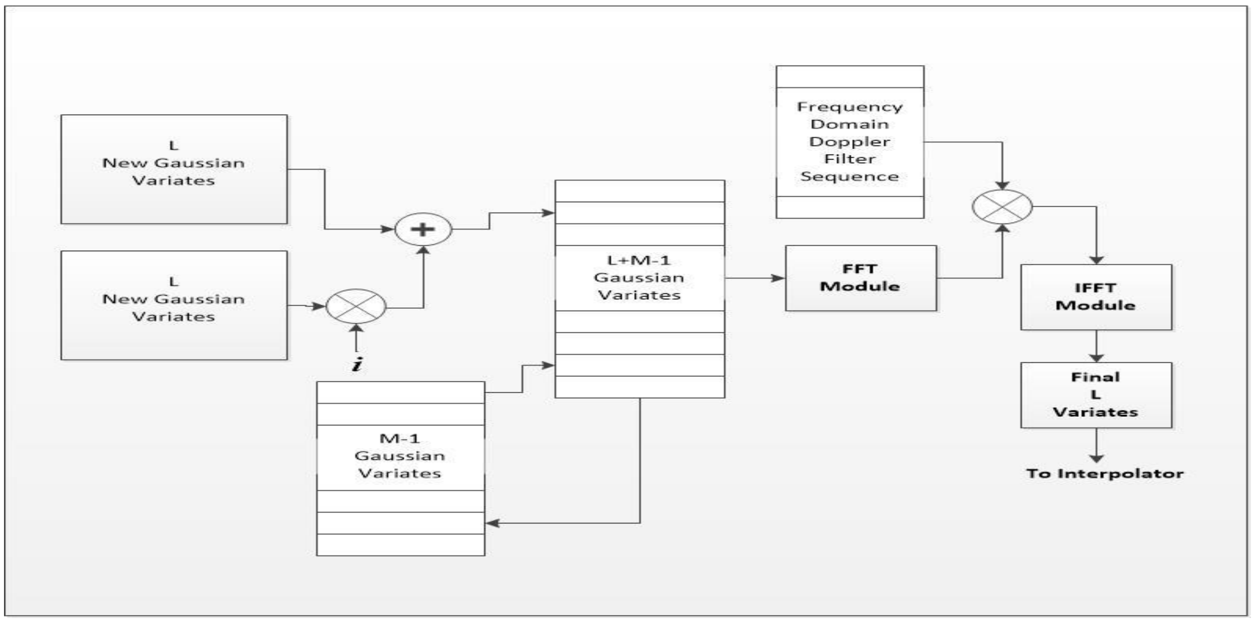


Figure 3.2: Original OLS-based Generator

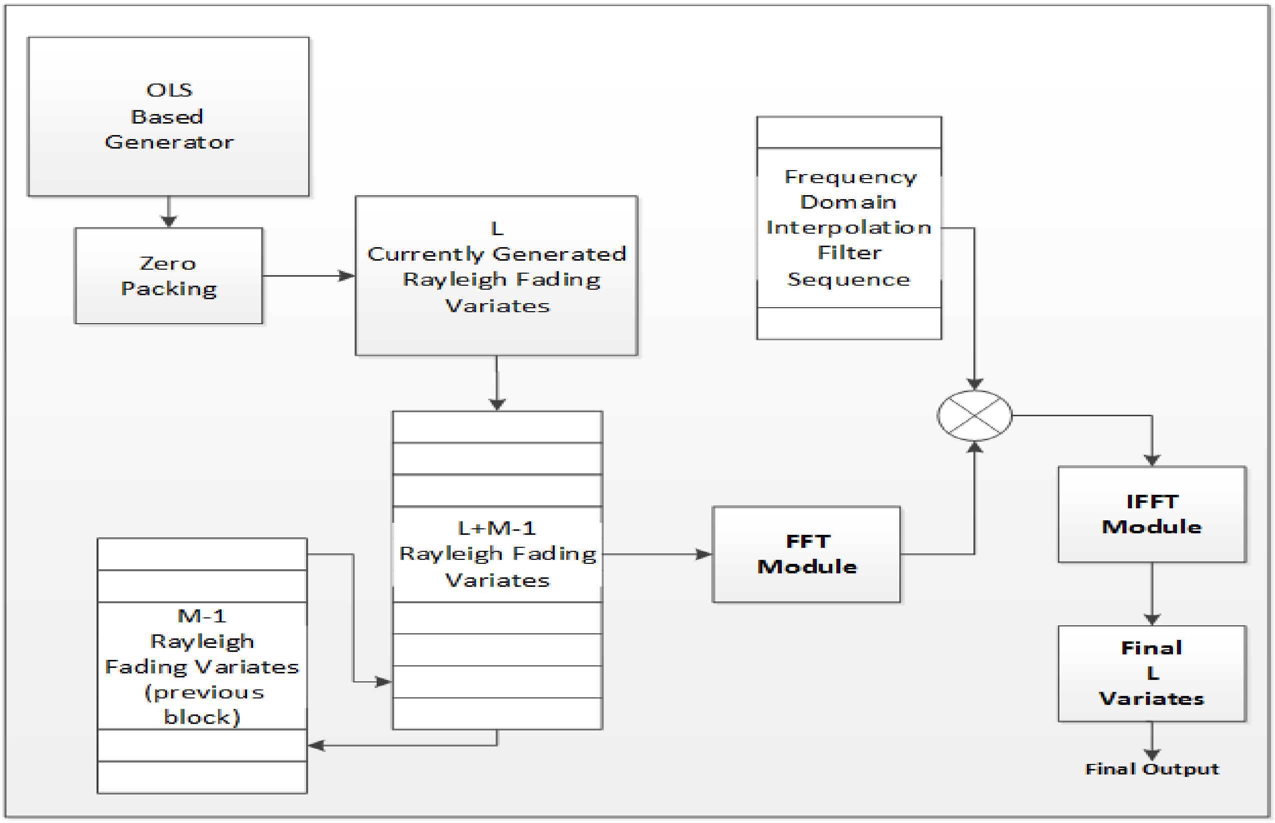

OLS Based Generator
Zero Packing
L Currently Generated Rayleigh Fading Variates
Frequency Domain Interpolation Filter Sequence
L+M-1 Rayleigh Fading Variates
M-1 Rayleigh Fading Variates (previous block)
FFT Module
IFFT Module
Final L Variates
Final Output

Figure 3.3: Original OLS-based Interpolator

The original OLS-based scheme proposed in [8] was efficient, however implementing it directly on FPGAs for MIMO based system requires too many resources. Also the frequency domain interpolation is efficient if upsampling rate is low, however it requires an additional FFT/IFFT module, complex multipliers and additional memory for storage of results. The time domain interpolation can act as a better solution where upsampling rates are high; also it does not require additional memories and FFT/IFFT modules. With slight modification to the technique of [27], an efficient time domain interpolator can be implemented. Only non-zero multiplication is required, and for high upsampling rate, there exists a lot of zeros multiplication, which can be neglected.

Proposed architecture for OLS-based Generator in this thesis is some modification to the original one proposed in [8]. So for a single module, following sub modules are required:

- Gaussian noise generation
- FFT/IFFT module

- Complex Multiplier
- Block Memories

3.1. Sub-Modules

3.1.1. Gaussian Noise Generation

One method in Gaussian noise generation is too study different methods in generating it, and implementing it from scratch in FPGA. However there exist some open source IP (Intellectual Property) cores which can be used in this thesis for efficient Gaussian Noise Generation. One of the open source cores for Gaussian noise generation is Gaussian Noise Generator Core (GNG) [28]. GNG core is efficient in resource consumption, has long period 2^{176}, provides high throughput. Also in [28] it is specified that GNG core can be used in accurate emulation of an Additive White Gaussian Noise (AWGN) channel. Output of GNG core is in Q (5, 11) format, in which 5 bits are of integer and 11 bits for fraction. GNG core's required resources on Virtex-7 FPGA is illustrated in Table 1.

Table 1: Implementation Results of GNG

Number of Utilized Slices	363
Number of RAM36E1	1
Number of DSP48E1	2
Maximum Frequency	361 MHz

Two instances of a single GNG core are required for emulation, one indicating real part and second one indicating imaginary part. The schematic module for the core is shown in Figure 3.4.

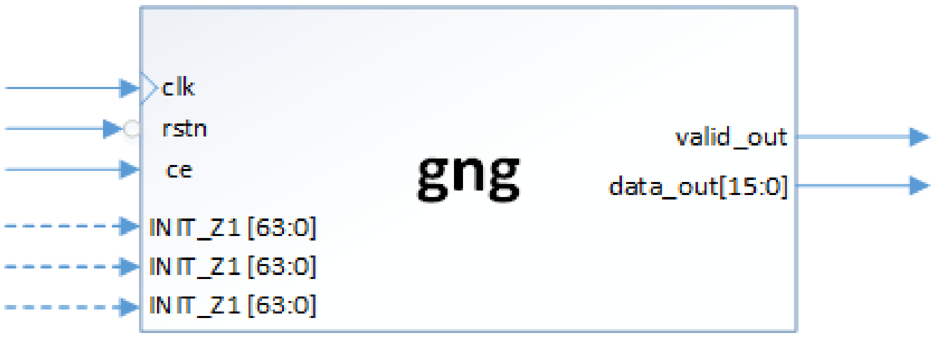


Figure 3.4: GNG Schematic

Init_Z1-3 are initial seeds, 'ce' is clock enable, which is kept high while generating new Gaussian variates. 'Valid_out' signal is tied high when GNG core is ready to output data. There is 11 cycle latency between 'ce' and 'valid_out' signal. Figure 3.5 displays the timing diagram of GNG core.

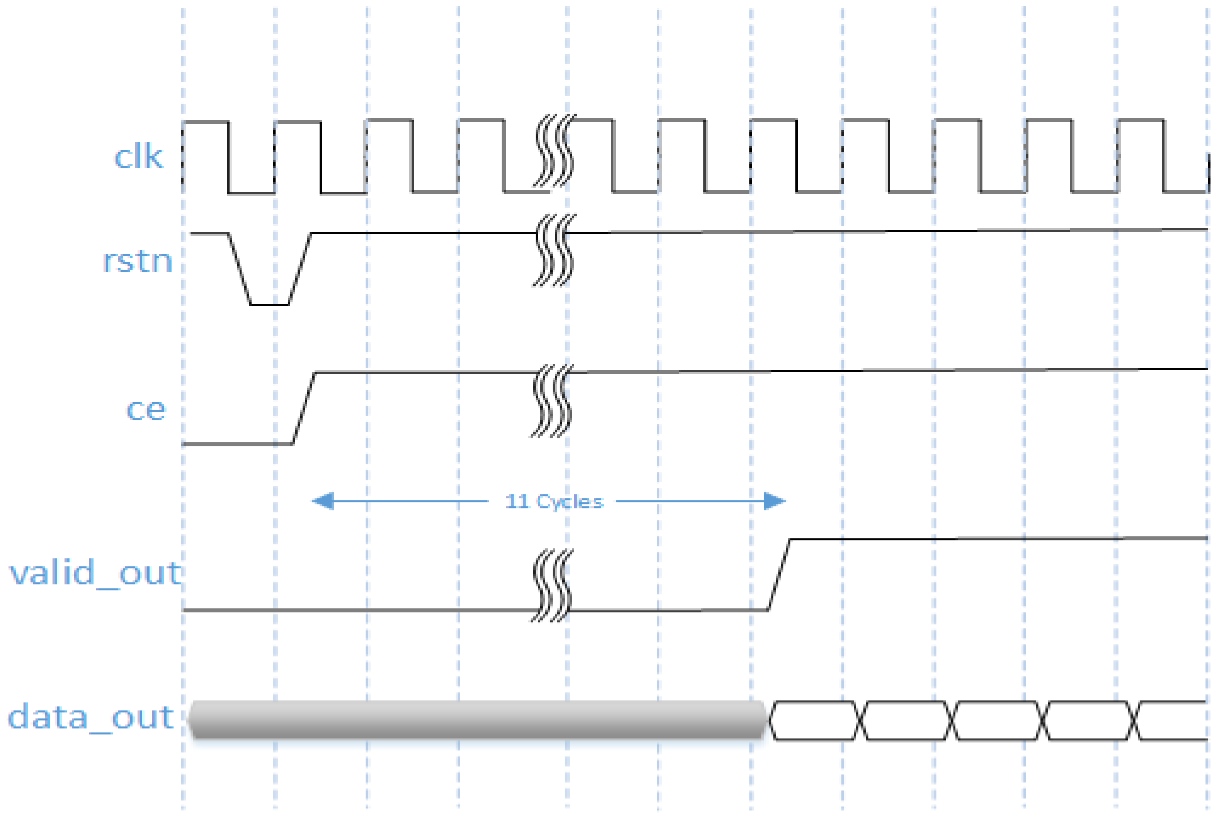


Figure 3.5: GNG Timing Diagram

3.1.2. FFT/IFFT Module

Like GNG core is available for Gaussian noise, Xilinx LogiCORE™ IP Fast Fourier Transform v8.0 is available for FFT/IFFT operation [29]. AXI4-Stream is the communication protocol used by the core. FFT core can take FFT of sizes from 2^3 to 2^{16}. Input data can range from 8 bits to 34 bits. Maximum operable frequency for this core is 550 MHz, and it can be used with floating or fixed point (full-precision or scaled). FFT size and FFT/IFFT mode can also be configured on run-time. Among four architectures, anyone can be selected based upon requirement. Four architectures provided by FFT core are:

- **Pipelined Streaming I/O**: useful for high throughput and continuous data streaming. Most resource consuming architecture, but best for high throughput sensitive applications such as the one in thesis.
- **Radix-4 Burst I/O**: less resource consuming but throughput is very low as compared to Pipelined Streaming I/O architecture. Loads and process data separately.
- **Radix-2 Burst I/O**: Same as Radix-4 architecture, but smaller butterfly size. It is slower than Radix-4 architecture, but consumes less resource.
- **Radix-2 Lite Burst I/O**: Longer transform time, but uses least resources.

There is a trade off between transform time and resource consumption. Large transform times requires lesser resources and vice versa. Figure 3.6 displays the throughput vs. resources [29].

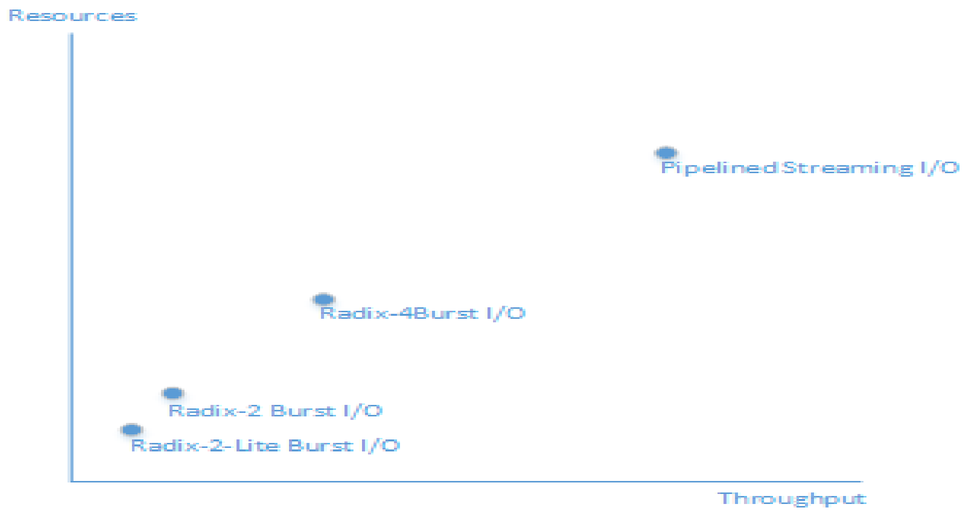


Figure 3.6: Resources vs. Throughput in FFT Core

Required throughput can only be achieved by first two architectures. With Pipelined Streaming I/O, a single FFT core can be used for two modules instead of two FFT cores due to its faster transform timings. FFT/IFFT size of 8192 has been selected in the thesis and after interpolation 4096 samples are generated in each process (see Appendix-A). Pipelined Streaming I/O architecture is shown in Figure 3.7.

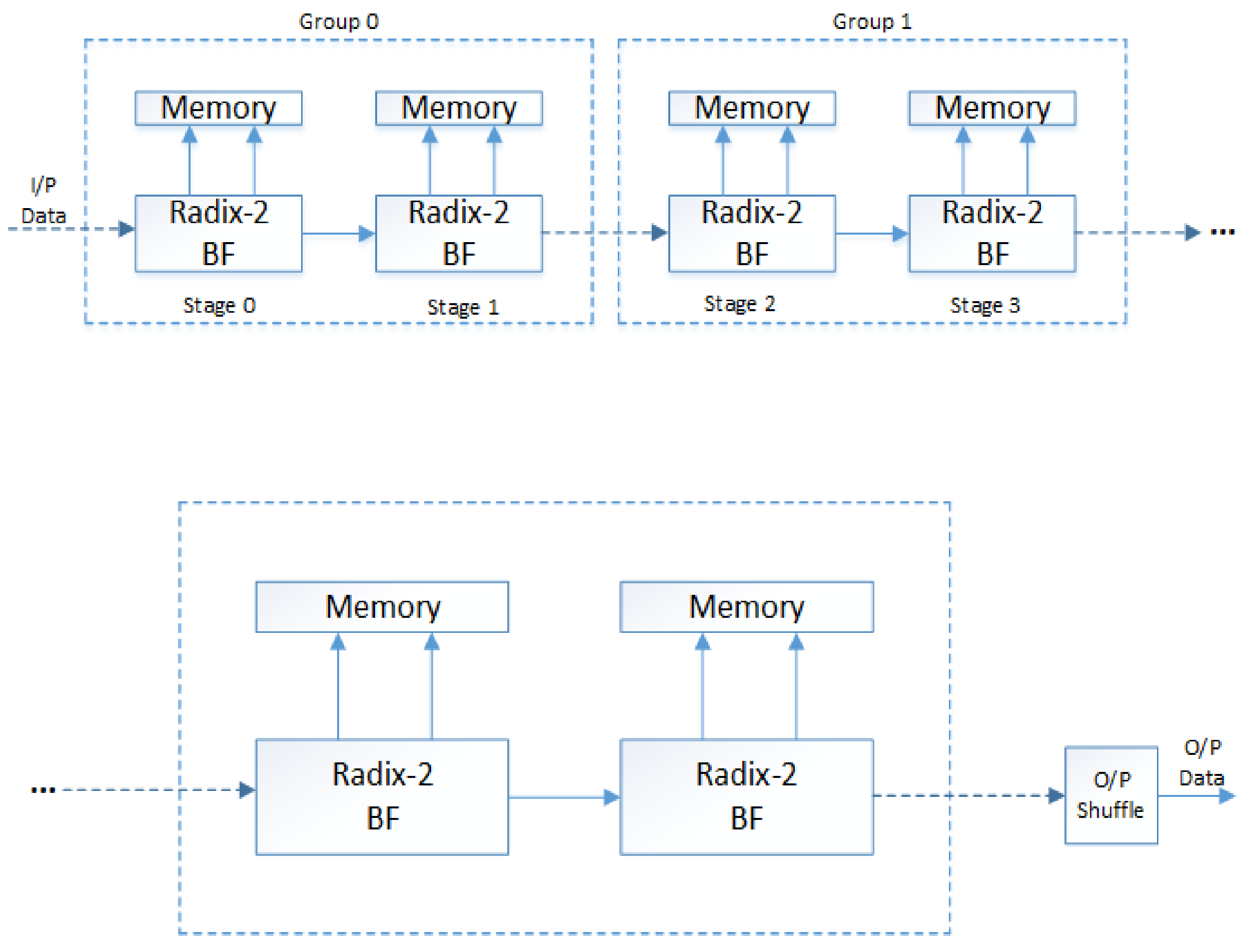


Figure 3.7: Pipelined Streaming I/O Architecture

On Xilinx ISE software or Vivado Design Suite software, different options are given to configure FFT core before generating it into project (ISE or Vivado Design Suite). These options include static or dynamic FFT sizes, selection of architecture, natural or bit-reverse output, and many more. Also latency and resources' consumption is also displayed. For an FFT size of 8192, with input as 16 bits (from GNG core), architecture as Pipelined Streaming I/O with full-precision fixed point and output as natural order, resources consumption by FFT core is illustrated in Table 2.

Table 2: FFT Core's Resource Utilization

Operable Frequency (Maximum)	550 MHz
Block Rams	52
DSP48E1	25

Before discussing the schematic diagram of FFT core, first AXI4-Stream protocol needs to be discussed. There exist 'TVALID' and 'TREADY' signals on FFT core. Loading data into FFT Core is controlled by 's_axis' signals and output data is controlled by 'm_axis' signals. Figure 3.8 describes the AXI4-Stream protocol.

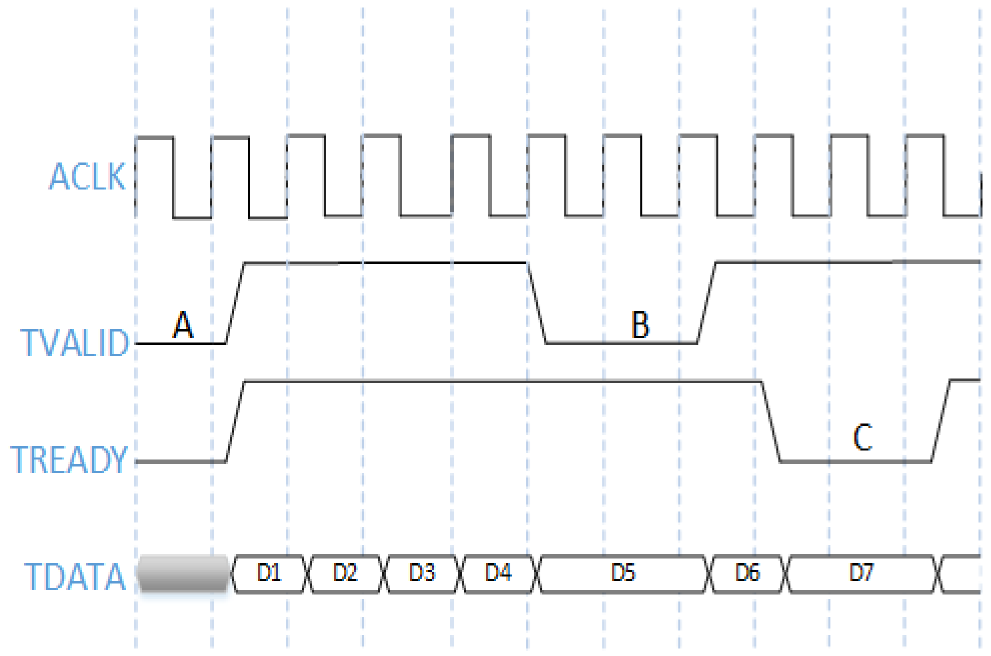


Figure 3.8: AXI Protocol

Points **A**, **B** and **C** are where no transfer of data takes place, **A** and **B** are master's wait state and **C** is slave's wait state. Master asserts signal 'TVALID' and Slave asserts 'TREADY', whenever both are high data transfer takes place. When loading a frame signal 's_axis_data_tvalid' is asserted, and kept high until no more data has to be sent. Same is the case when FFT core unloads data; it asserts ‘m_axis_data_tready’ and keep it high until all data is unloaded.

Schematic Symbol of FFT Core is shown in Figure 3.9. ‘Config_tdata’ signal’s LSB indicates whether an FFT or an IFFT has to be taken, ‘s_axis_data_tlast’ can be asserted to indicate last sample of transform. FFT output signals are defined by ‘m_axis’, ‘m_axis_data_tlast’ is asserted to indicate last output of frame. Various event signals from FFT core indicate different things, such as start of the FFT process. Event signal indicating start of FFT process is important, as if new configuration has to be applied; it is applied after this signal is asserted.

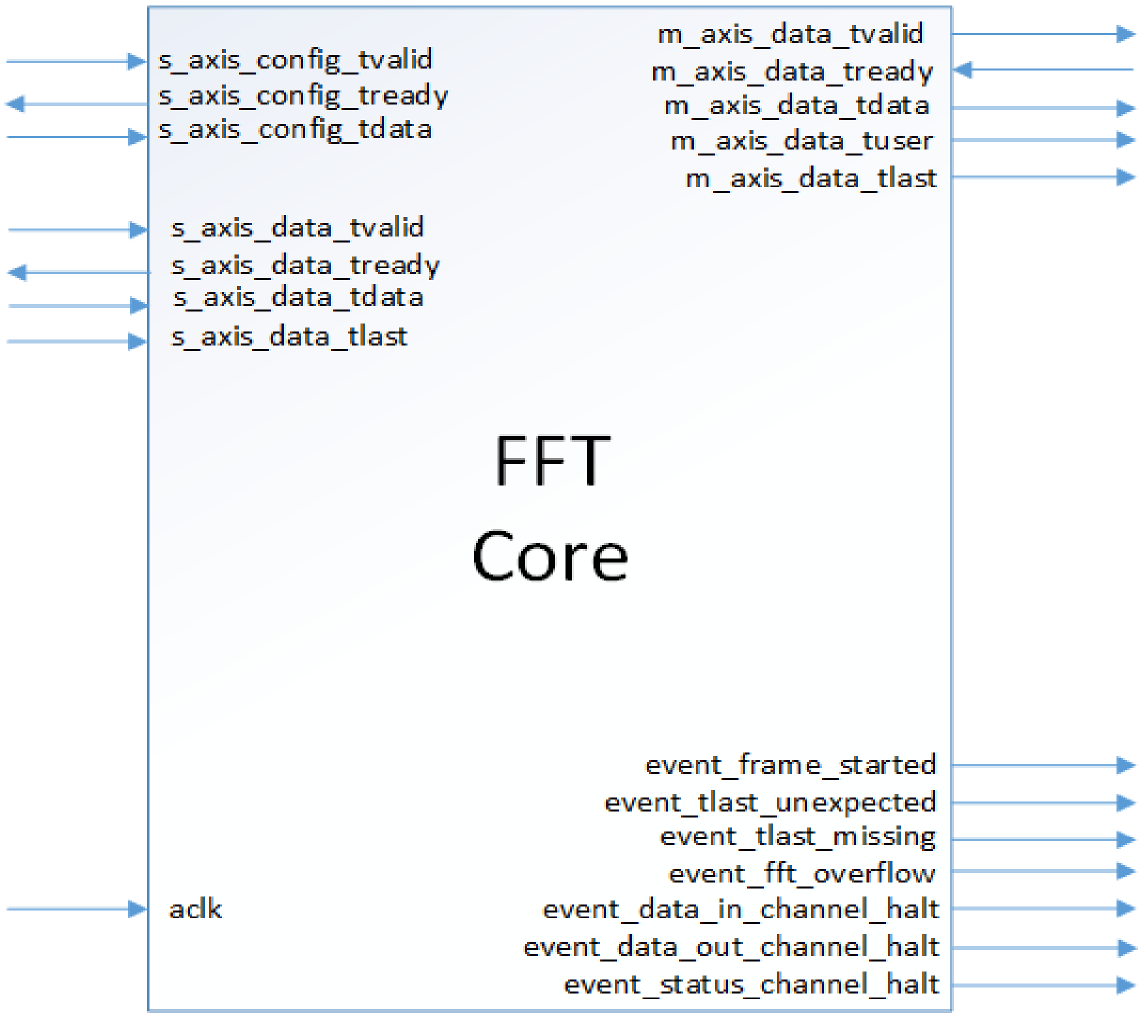


Figure 3.9: FFT Core Schematic

In Pipelined Streaming I/O architecture, latency is approximately 3N cycles with natural output, if data is continuously loaded into FFT core, after initial latency, each frame is output after N cycles. If a wait state is inserted in between each frame, this wait state is also seen in between output frames, for example if ‘K’ number of cycles are consumed by wait state in between loading two frames, this ‘K’ wait state cycles will also occur in between output of these two frames. Figure 3.10 shows timing diagram of Pipelined Streaming I/O [29].

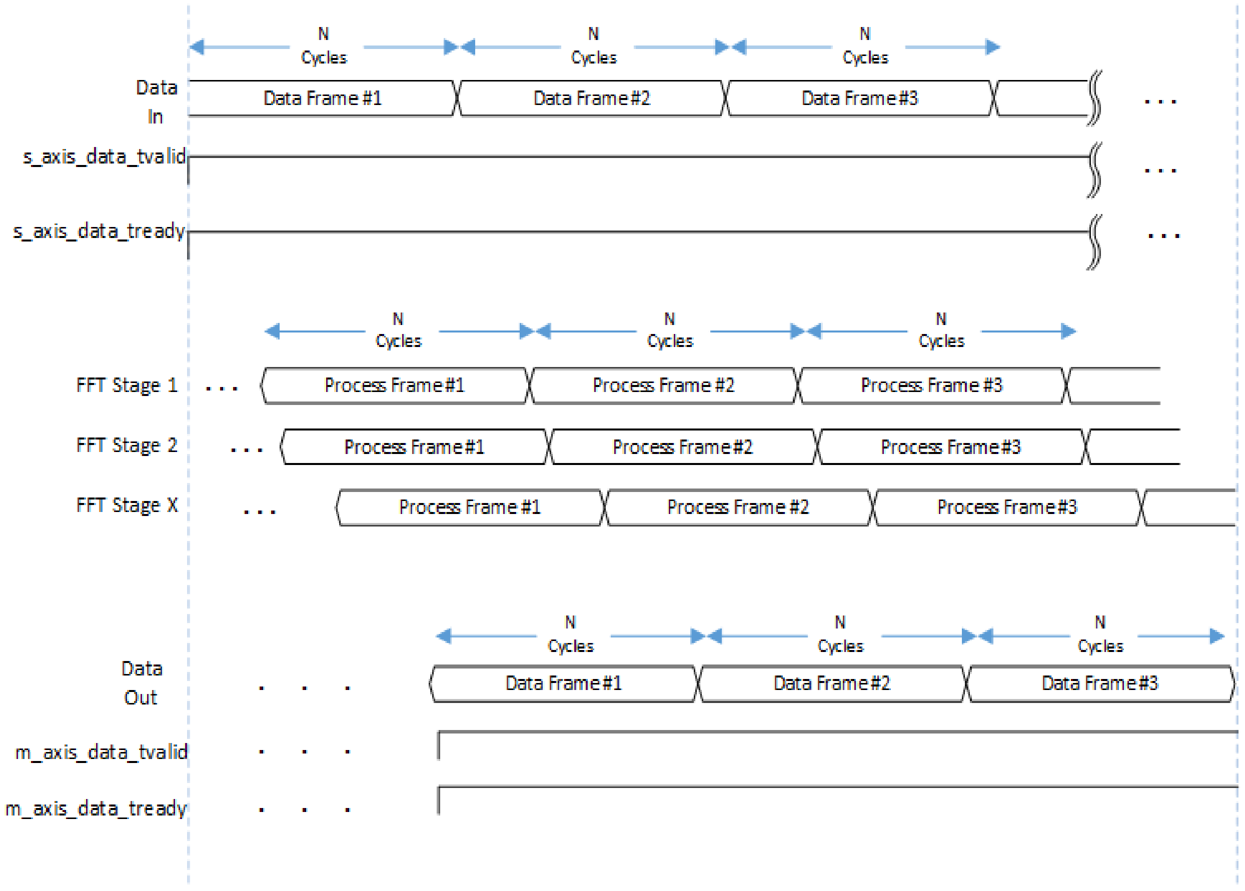


Figure 3.10: Frame by Frame Processing and Timings in Pipelined Streaming I/O Architecture

3.1.3. Complex Multiplier

For complex multiplication there is also an IP core available [30], however with the usage of DSP slices [31] it is simple to implement a complex multiplier using four DSP48E1s/multipliers. Also inside DSP slice, pipeline registers can be added which increases throughput to a greater number. Figure 3.11 shows a complex multiplier with pipeline registers added in each stage. Pipeline registers reduces the critical path, which in turn increases the maximum operable frequency of whole system. Multipliers in Figure 3.11 is made up of DSP slices, a single DSP slice can multiply two real numbers of size 25 and 18 bits (maximum size). Equation for Figure 3.11 is:

$$(A + Bi) \times (C + Di) = (AC - BD) + i(AD + BC)$$

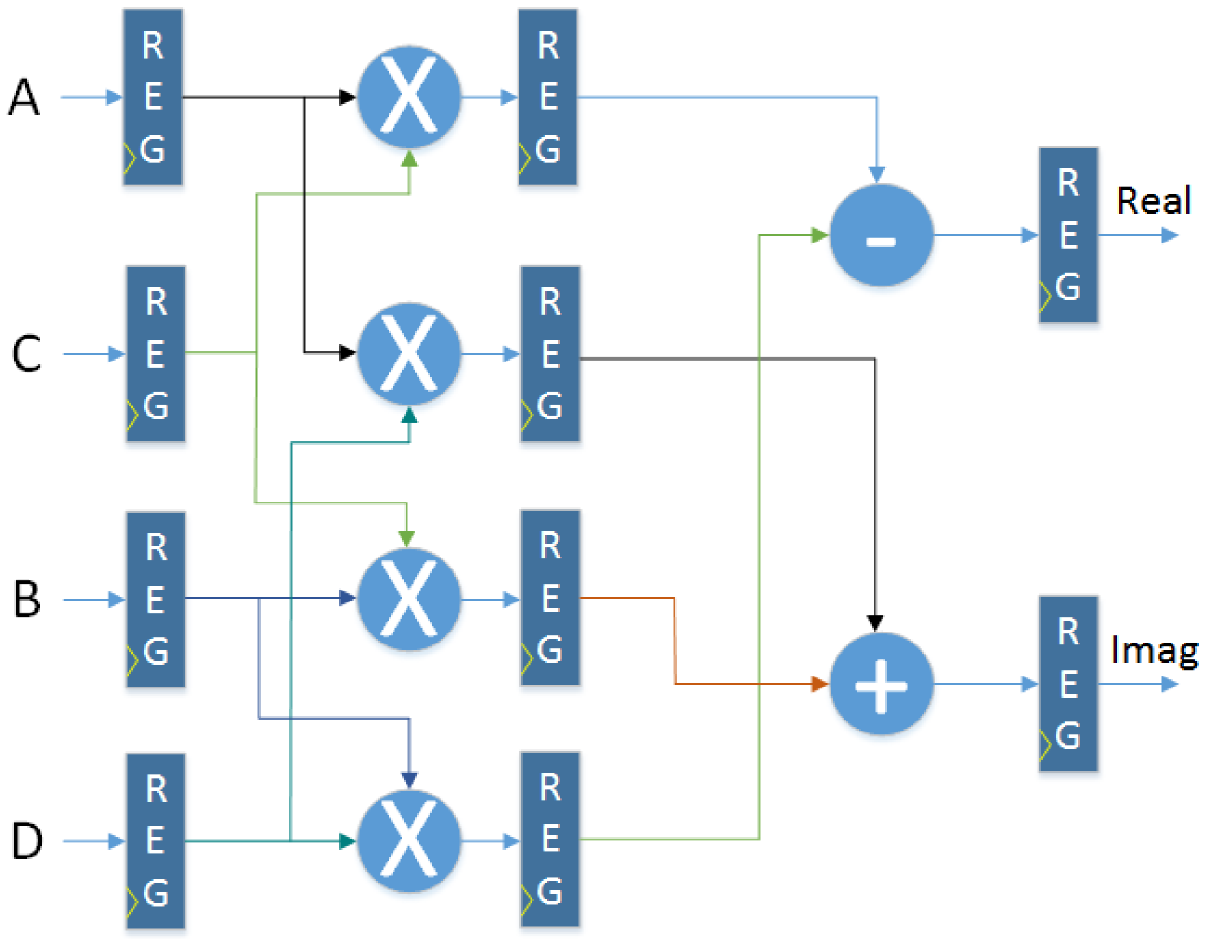


Figure 3.11: Complex Multiplier

Output of FFT core when full-precision is selected is equal to input size + $log_2 (N)$ + 1. With 16 bits as input output is in 30 bits (8192 FFT length). For utilizing a single multiplier for this purpose needs truncation of data. Last 5 bits from LSB are chopped and results are stored in 25 bits (real and imaginary part separately). This core output is multiplied with Doppler Filter Sequence in Frequency Domain, Doppler Filter Sequence are stored as Q (1, 15) format in a memory. Hence a single DSP48E1/multiplier can be used.

3.1.4. Block Memories

Block Memory Generator [32] is a GUI based memory constructor, available in Vivado Design Suite and Xilinx ISE. ROMs, RAMs and Block RAMs can be constructed (single, dual or triple port) with it. In this thesis various block RAMs are required. One block RAM is required to store Doppler Filter Sequence (Frequency Domain). As already mentioned 32 bits are required (real and imaginary 16 bits each) for Doppler Filter Sequence, a total of 8192x32 spaced memory is required. Also this memory needs to be single port as filter sequence is written one time only and a simple counter for address can control it.

Second required memory is for Interpolator's coefficients. As 4096 outputs are generated (see Appendix-A) in each process, 4096 real coefficients are required to be stored in advance. Analysis shown (see Appendix-B) Q (1, 24) is enough for storing time domain filter coefficients, so a memory of 4096x25 is required (single port interface). If a frequency domain interpolation filter sequence needs to be stored, it will require 8192x50 memory locations, as both real and imaginary part needs to be stored. So in terms of Block RAMs, time domain interpolation also has an edge over frequency domain interpolation.

Third memory required is for GNG core's outputs, output of GNG core is stored in a dual port BRAM. Dual port BRAM is required because OLS method requires some old values and new values, as FFT core is accessing data from a single memory at a time, it is required that new values are also written in it. Depending upon the value of 'L' in OLS method, new variates are generated. A dual port memory of 8192x32 is enough for GNG core as FFT size is 8192. After 8192 variates are loaded into FFT core, an offset is added to address register of memory, which skips first 'L' variates, meanwhile process for 'L' new variate generation is started and new variates are written at start of memory. So a counter executes 8192 times, however an offset of ‘*L*’ is always added in address register. With this scheme there is no requirement of shifting memory after loading one frame.

Fourth memory required is for storing outputs of OLS-generator, a memory of 64x32 (see Appendix-A) is required for it, as for the system’s requirement maximum value of ‘*L*’ in OLS-method is ‘64’. Also if a frequency domain interpolation is intended, memory requirements increase drastically. Zeros are needed to be inserted between non-zero Rayleigh fading variates. A total of 8192x32 is the minimum requirement; also more DSP slices will be needed for complex multiplication (25x30 bit multiplication needed). So it is convenient to use a time domain solution, where upsampling rates are high. Also a 64x60 bits memory is required in each OLS-based generator for synchronizing all interpolators.

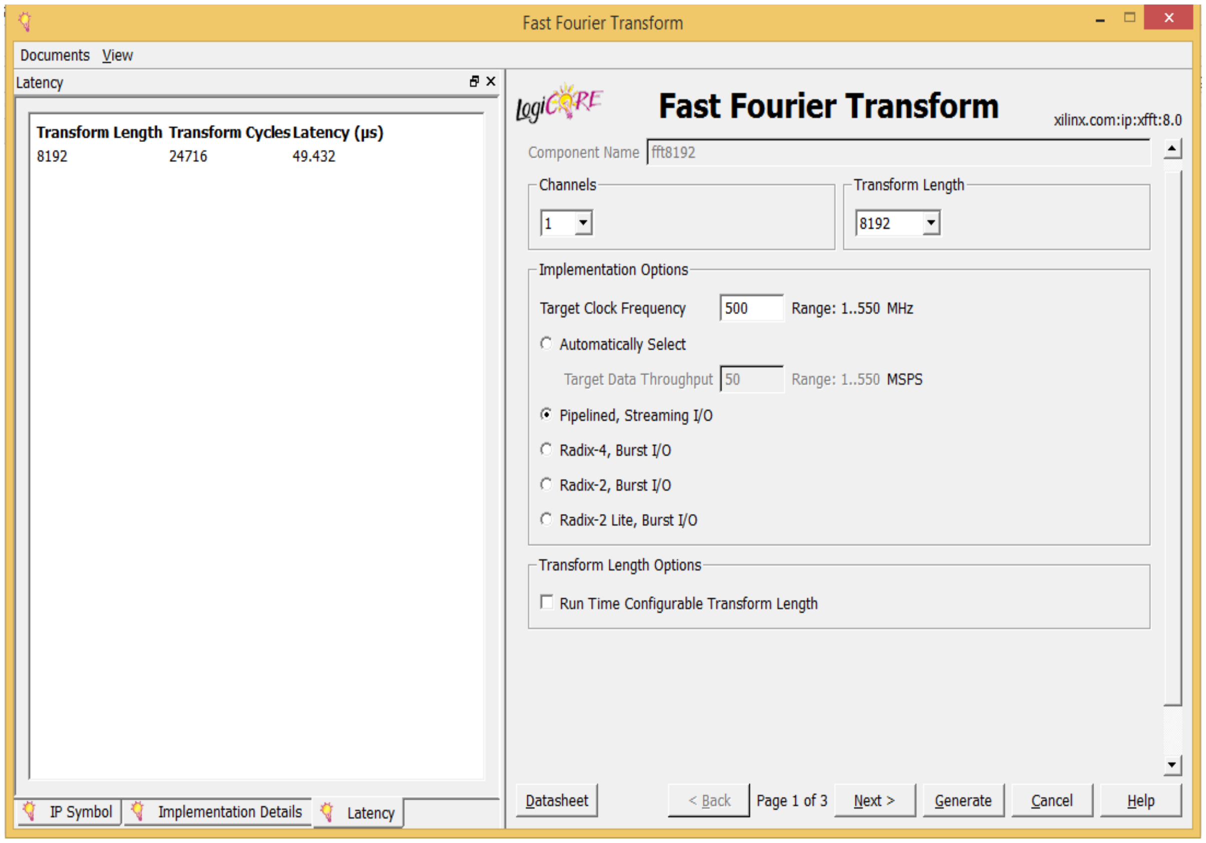
3.2. Timing Requirements

For achieving required throughput, a single process needs to be executed multiple times, with that information total number of cycles can be calculated and a frequency can be targeted. If maximum frequency (critical path’s inverse) of FPGA based design is greater than required frequency, timing requirements are achieved.

In this thesis a target throughput of 34 Mega Symbols per Second was intended. This rate is chosen by keeping in view maximum sampling frequency in LTE is 30.72MHz for a 20 MHz configuration [33]. At least 30.72 Mega Symbols per Second is needed, but in this thesis 34 MSPS was adjusted, as in this case system scalability can also be judged (for higher throughputs how this system can work).

Each time the process is executed, ‘*L*’ Rayleigh fading variates are generated and interpolator produces 4096 outputs (see Appendix-A). For 34 MSPS, a total of 8301 times the process needs to be executed in one second. So a single process which generates 4096 outputs cannot exceed 120 micro seconds time.

Initially 8192 cycles are required to first time fill up the memory for Gaussian variates, which is needed for FFT core; however this initialization can be done in a state of a Finite State Machine (FSM). FFT core for 8192 size, working with Pipelined Streaming architecture requires 24716 cycles. Figure 3.12 displays the FFT Core generation screen.



Transform Length	Transform Cycles	Latency (µs)
8192	24716	49.432

Fast Fourier Transform

Component Name: fft8192

Channels: 1

Transform Length: 8192

Implementation Options

Target Clock Frequency: 500 Range: 1..550 MHz

( ) Automatically Select

Target Data Throughput: 50 Range: 1..550 MSPS

[✓ Pipelined, Streaming I/O]

( ) Radix-4, Burst I/O

( ) Radix-2, Burst I/O

( ) Radix-2 Lite, Burst I/O

Transform Length Options

☐ Run Time Configurable Transform Length

Figure 3.12: FFT Core Generation in Xilinx ISE Design Suite 14.7

There can be various schemes for achieving targeted system throughput some of them are:

3.2.1. Scheme 1: Separate FFT/IFFT Module and Interpolator

If an IFFT module is separately instantiated, Interpolator utilizes equal or lesser cycles than the latency, and simultaneously after one frame's complete output second frame is loaded, $24716 \times 8301 = 205167516$ cycles are needed. This system can work on 205.16MHz. Timing diagram of this system is displayed in Figure 3.13 (pipeline registers and DSP slices' initial delays are neglected).

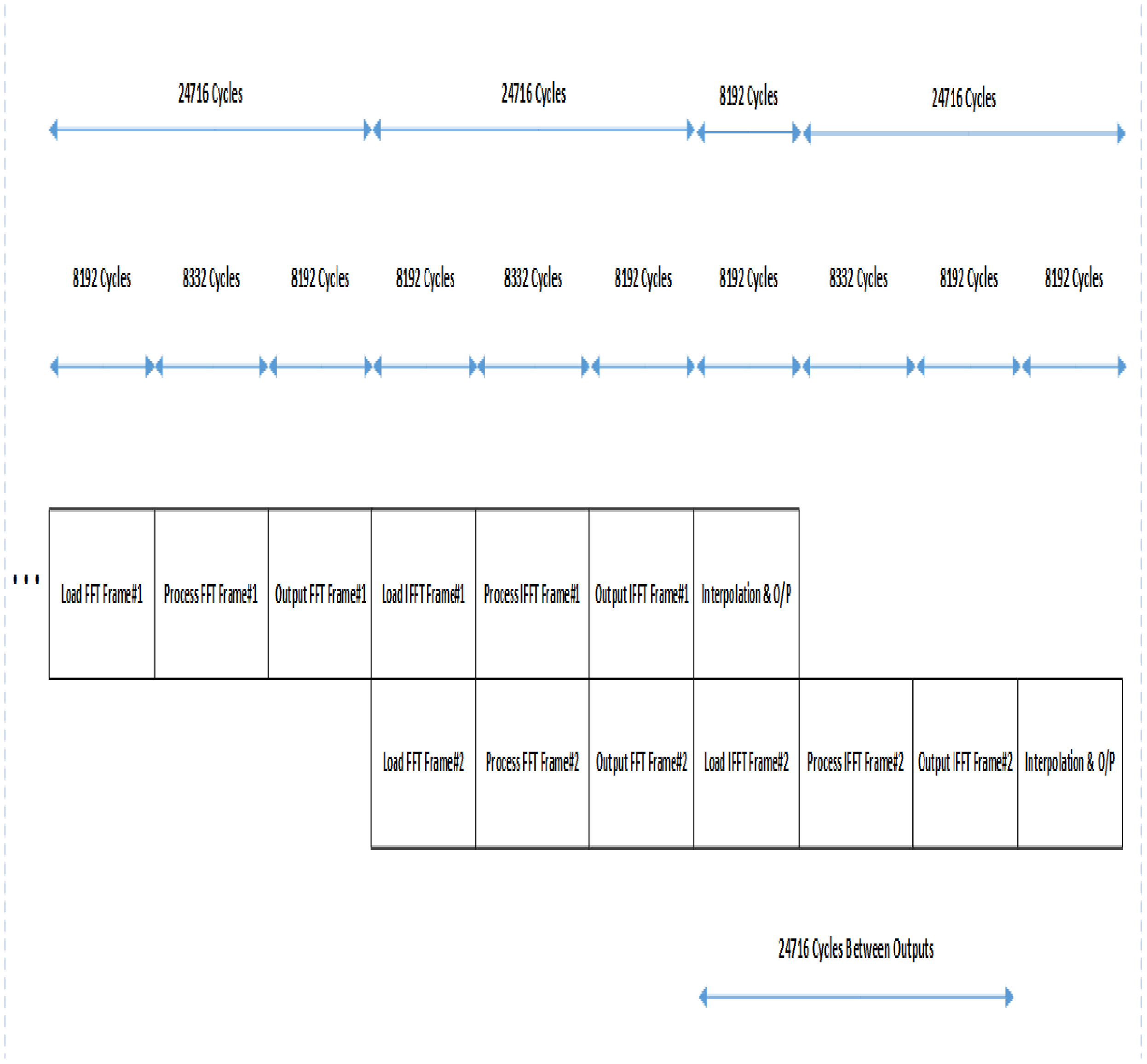


Figure 3.13: Scheme 1 Timing Requirements (without Pipelining)

This scheme can provide required results, however this system is poorly pipelined and better pipelining can be achieved if frames are continuously loaded and unloaded. This will generate output after every 8192 cycles, which mean $8192 \times 8301 = 68001792$ cycles are required, this system can operate on 68 MHz, and such a low frequency indicate that it is very easy to implement. Figure 3.14 displays timing requirement of this scheme.

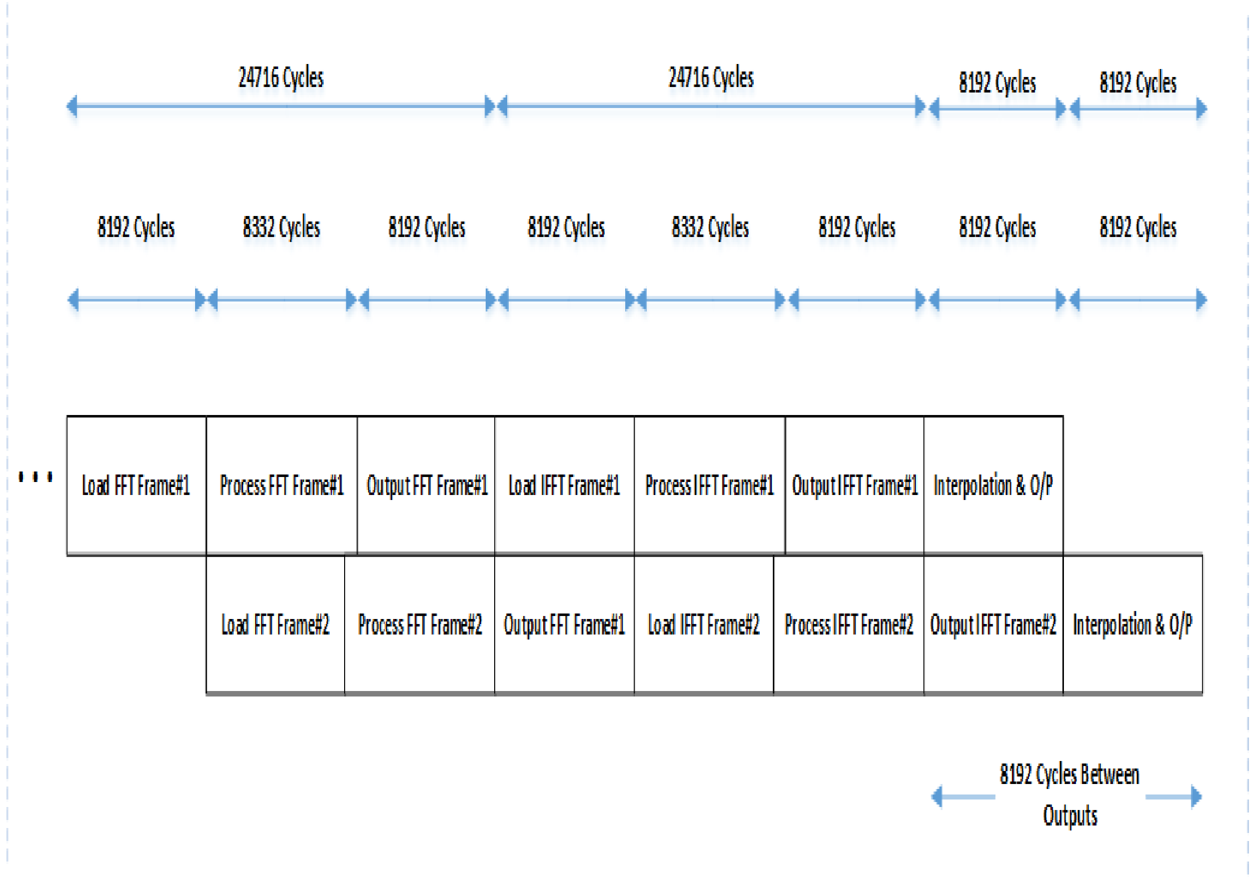


Figure 3.14: Scheme 1 Timing Requirements (with Pipelining)

Scheme of using two instances of FFT core (one as FFT other as IFFT) is a simple technique, and it provides very high throughputs. Flaw of this method lies in its large resource consumption. Each instance of FFT core consumes a lot of resources and interpolator module has just 8192 cycles in which it has to generate outputs, with fast processing a large amount of complexity comes. For this system, time domain interpolator requires '*L*' complex multiplications, in which Rayleigh fading variate (complex) is multiplied with real interpolation filter coefficients. Complexity increases along with size of '*L*'. Even with 64 as maximum size of '*L*', MIMO based solution cannot be implemented on targeted FPGA device. Only way to efficiently use this method is to utilize frames of different types (different GNG memories) by a single module and incorporate block memories at the output of interpolator. Also this method can be efficient where single antenna systems needs to be analyzed.

3.2.2. Scheme 2: Single FFT/IFFT Module and Interpolator

With a single FFT/IFFT module, a control mechanism is always needed, as a same module is being used in FFT and IFFT modes. There can be two methods in using single FFT/IFFT module, both of these methods require equal number of interpolators; however FFT/IFFT module instances can be reduced.

First method is to use a single FFT instance, load two frames in it, and stop the FFT core for further inputs. When the output is ready and being sent out, feed it back to the same core after multiplication with Doppler filter sequence. Now this first method of this scheme requires more cycles and more complex control mechanism as a single core is doing both tasks. When FFT core starts outputting IFFT output, again process is restarted and again two frames are loaded. 'L' number of useful IFFT outputs of two frames is sent to interpolator, and now interpolator module has enough time to do its task with reduced number of resources. As already mentioned, latency of a core is 24716 clock cycles, when output of FFT core is started, new data can be loaded in it (due to Pipelined Streaming architecture). So after 16524th clock cycle, output data from FFT core can be multiplied with Doppler filter sequence and can be feedback to FFT core for IFFT operation. Figure 3.15 illustrates the concept, along with information of consumed clock cycles. Halt indicates where FFT core doesn't take input.

Initially some cycles are required, before pipeline is established, initially 49441 clock cycles are consumed by FFT and IFFT process, however at 33058th cycle frame number 3 is loaded into FFT, with this in between outputs, 32918 clock cycles are required. Hence the process requires 32918 clock cycles.

With this scheme two frames are loaded at a time which means this process needs to be executed half times in a second. 4151 times this process needs to be executed in a second, so with this scheme $33058 + 32918 \times 4352 = 136675676$ clock cycles are required. This system can work on 136.7 MHz, and is efficient in terms of resource utilization.

Drawback of this method lies in the structure of interpolator. Interpolator has 16k cycles for each frame (total 32k). With 4096 cycles requiring maximum complex multipliers, 16k cycles will reduce multipliers to four times; however this architecture requires replication of a single module

36 times. This will significantly increase the resource consumption, and scalability of such a system is difficult, however this scheme is scalable and efficient if targeted throughput is increased.

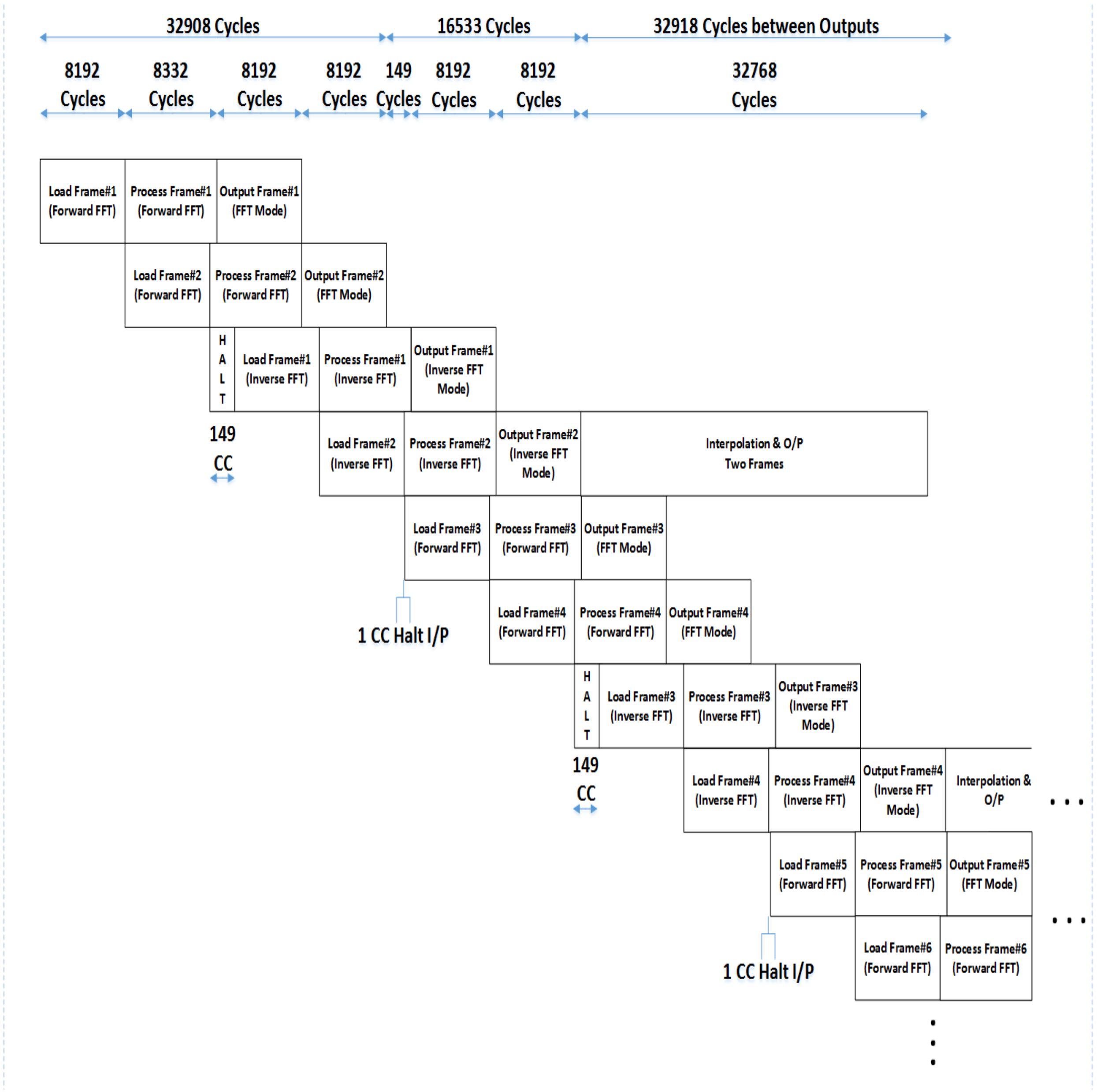

32908 Cycles
16533 Cycles
32918 Cycles between Outputs
8192 Cycles
8332 Cycles
8192 Cycles
8192 Cycles
149 Cycles
8192 Cycles
8192 Cycles
32768 Cycles
Load Frame#1 (Forward FFT)
Process Frame#1 (Forward FFT)
Output Frame#1 (FFT Mode)
Load Frame#2 (Forward FFT)
Process Frame#2 (Forward FFT)
Output Frame#2 (FFT Mode)
HALT
Load Frame#1 (Inverse FFT)
Process Frame#1 (Inverse FFT)
Output Frame#1 (Inverse FFT Mode)
149 CC
Load Frame#2 (Inverse FFT)
Process Frame#2 (Inverse FFT)
Output Frame#2 (Inverse FFT Mode)
Interpolation & O/P Two Frames
Load Frame#3 (Forward FFT)
Process Frame#3 (Forward FFT)
Output Frame#3 (FFT Mode)
1 CC Halt I/P
Load Frame#4 (Forward FFT)
Process Frame#4 (Forward FFT)
Output Frame#4 (FFT Mode)
HALT
Load Frame#3 (Inverse FFT)
Process Frame#3 (Inverse FFT)
Output Frame#3 (Inverse FFT Mode)
149 CC
Load Frame#4 (Inverse FFT)
Process Frame#4 (Inverse FFT)
Output Frame#4 (Inverse FFT Mode)
Interpolation & O/P
Load Frame#5 (Forward FFT)
Process Frame#5 (Forward FFT)
Output Frame#5 (FFT Mode)
1 CC Halt I/P
Load Frame#6 (Forward FFT)
Process Frame#6 (Forward FFT)

Figure 3.15: Scheme 2 Timing Requirement (1st Method)

It can be seen that first method's critical path is average, if half of critical path can be achieved; two paths can be simultaneously emulated with a single FFT core. This second method involves loading frames of two different paths into FFT core, and finally two interpolators are instantiated for each path. This method now requires half number of FFT/IFFT instances and memory requirements can also be reduces as all modules can be driven by single instances of memories. Figure 3.16 shows the timing diagram. Note that now $33058 + 32918 \times 8301 = 273285376$ clock cycles are required. System can work on 273.28 MHz, achieving such frequency is difficult, but incorporating pipelining can make it possible. This system is scalable and efficient if more antennas are incorporated (MIMO system with more antennas needs to be emulated).

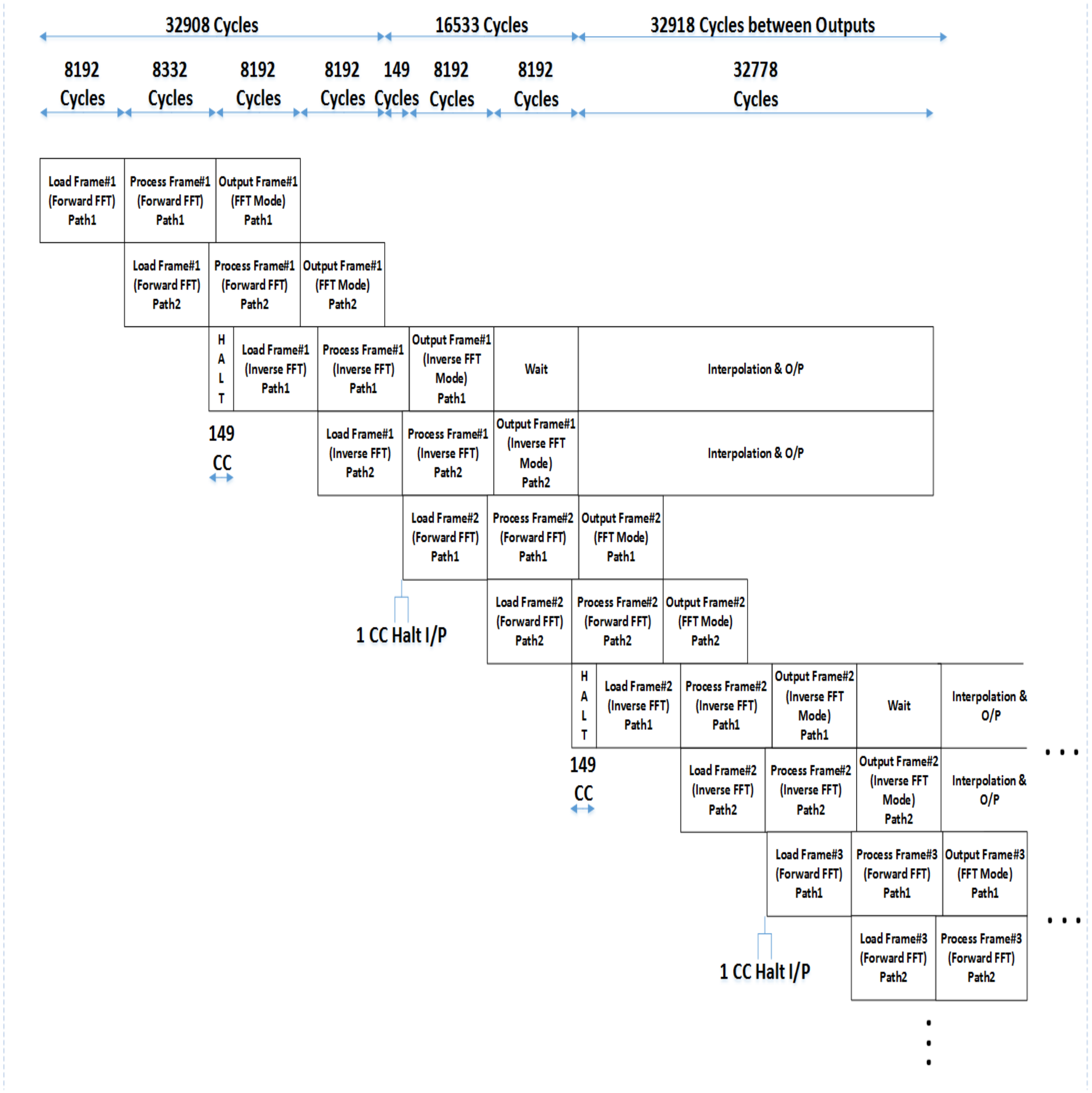

32908 Cycles
16533 Cycles
32918 Cycles between Outputs
8192 Cycles
8332 Cycles
8192 Cycles
8192 Cycles
149 Cycles
8192 Cycles
8192 Cycles
32778 Cycles
Load Frame#1 (Forward FFT) Path1
Process Frame#1 (Forward FFT) Path1
Output Frame#1 (FFT Mode) Path1
Load Frame#1 (Forward FFT) Path2
Process Frame#1 (Forward FFT) Path2
Output Frame#1 (FFT Mode) Path2
HALT
Load Frame#1 (Inverse FFT) Path1
Process Frame#1 (Inverse FFT) Path1
Output Frame#1 (Inverse FFT Mode) Path1
Wait
Interpolation & O/P
149 CC
Load Frame#1 (Inverse FFT) Path2
Process Frame#1 (Inverse FFT) Path2
Output Frame#1 (Inverse FFT Mode) Path2
Interpolation & O/P
Load Frame#2 (Forward FFT) Path1
Process Frame#2 (Forward FFT) Path1
Output Frame#2 (FFT Mode) Path1
1 CC Halt I/P
Load Frame#2 (Forward FFT) Path2
Process Frame#2 (Forward FFT) Path2
Output Frame#2 (FFT Mode) Path2
HALT
Load Frame#2 (Inverse FFT) Path1
Process Frame#2 (Inverse FFT) Path1
Output Frame#2 (Inverse FFT Mode) Path1
Wait
Interpolation & O/P
149 CC
Load Frame#2 (Inverse FFT) Path2
Process Frame#2 (Inverse FFT) Path2
Output Frame#2 (Inverse FFT Mode) Path2
Interpolation & O/P
Load Frame#3 (Forward FFT) Path1
Process Frame#3 (Forward FFT) Path1
Output Frame#3 (FFT Mode) Path1
1 CC Halt I/P
Load Frame#3 (Forward FFT) Path2
Process Frame#3 (Forward FFT) Path2

Figure 3.16: Scheme 2 Timing Requirement (2nd Method)

Due to the scalability in MIMO system, method two of scheme two was chosen as the main architecture for implementation.

3.3. System Architecture

Top level architecture of a single module is displayed in Figure 3.17. Each module incorporates two paths.

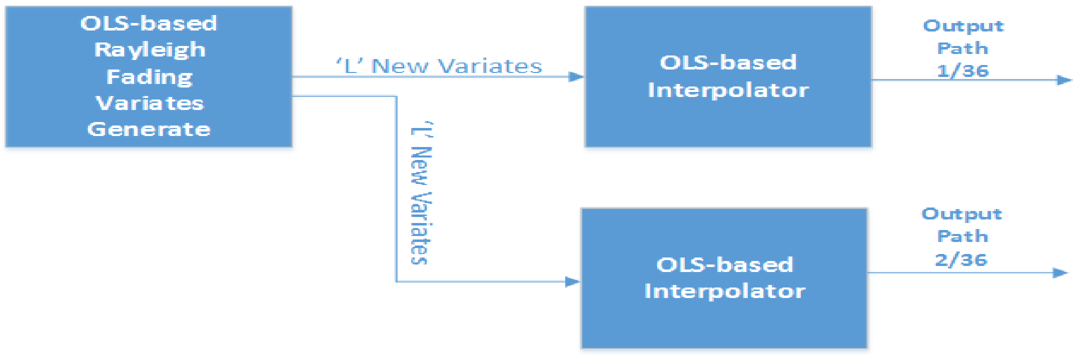


Figure 3.17: Top Level Architecture (Single Module)

For each antenna there are nine paths, this is modelled as Tapped Delay Line Channel Model, for a single antenna system following diagram illustrates concept.

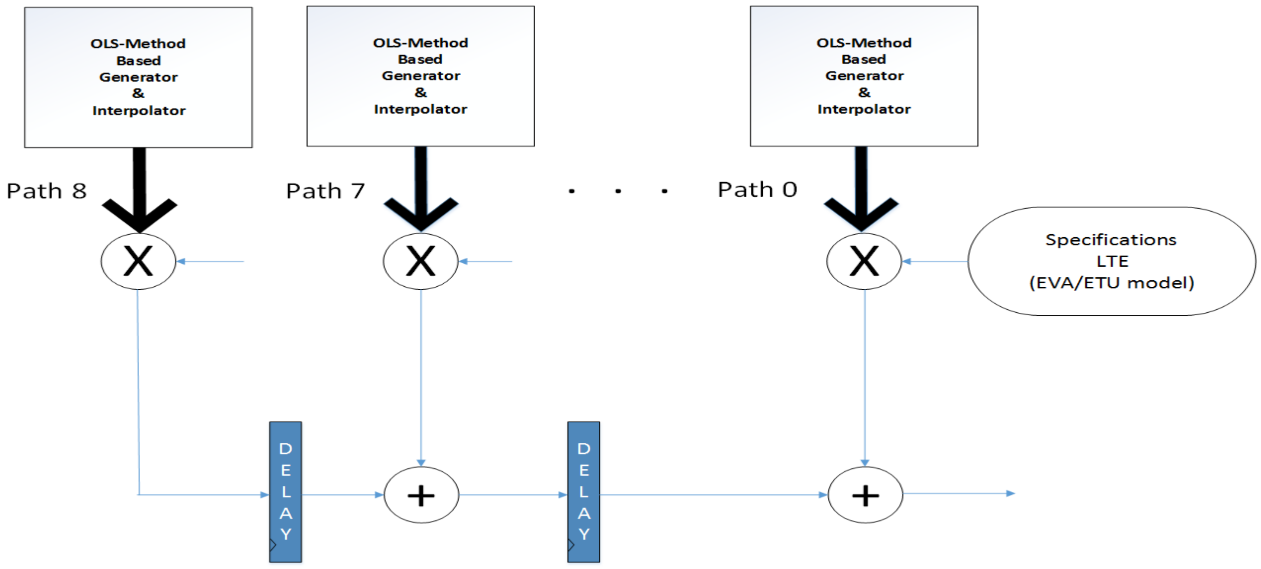

OLS-Method Based Generator & Interpolator
OLS-Method Based Generator & Interpolator
OLS-Method Based Generator & Interpolator
Path 8
Path 7
. . .
Path 0
X
X
X
Specifications LTE (EVA/ETU model)
DELAY
+
DELAY
+

Figure 3.18: Tapped Delay Line Channel Model

For 36 paths, eighteen instances of the above module are required. Overall top level module is shown in Figure 3.19.

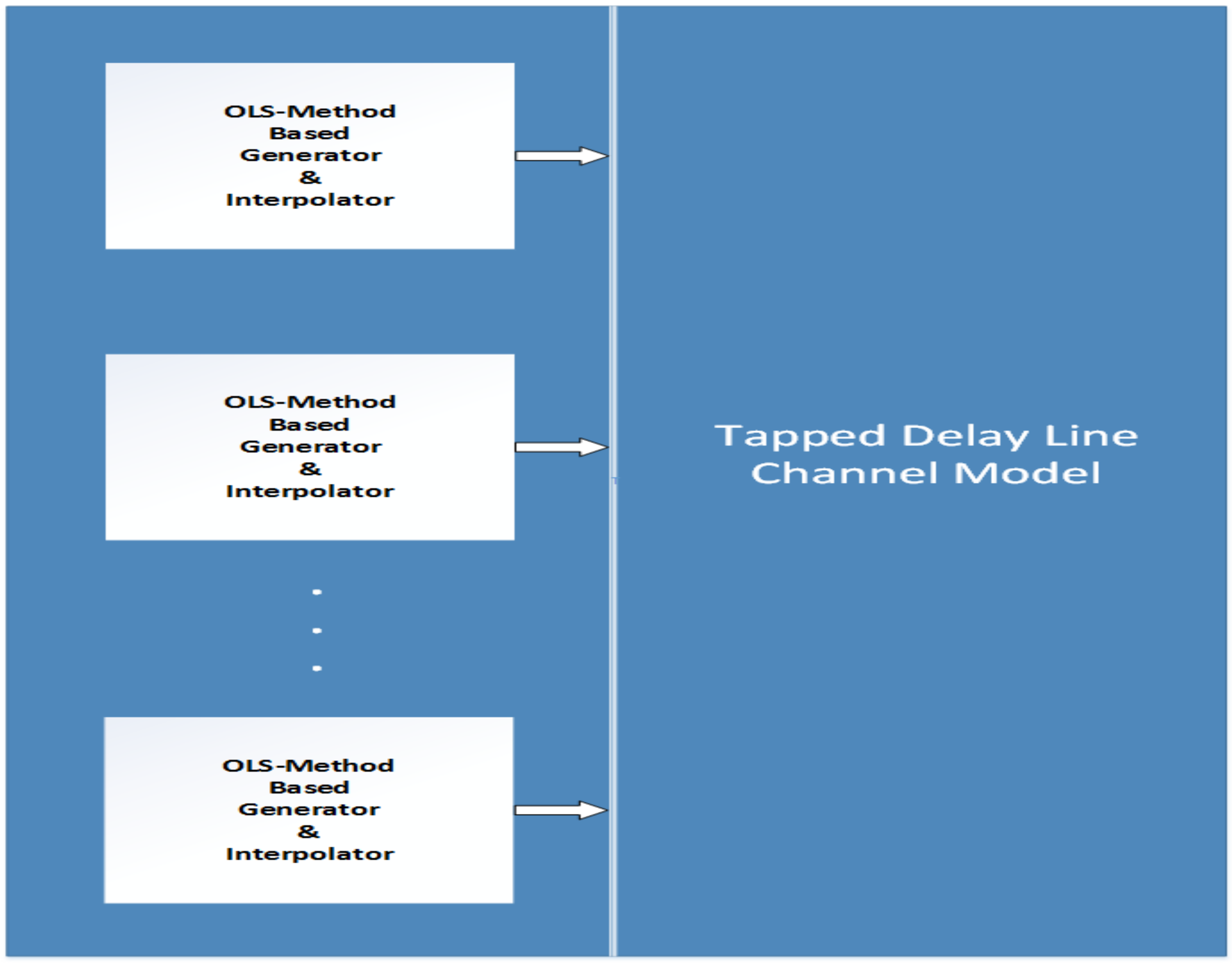

OLS-Method Based Generator & Interpolator
OLS-Method Based Generator & Interpolator
OLS-Method Based Generator & Interpolator
Tapped Delay Line Channel Model

Figure 3.19: Top Level (MIMO System)

Implementing Tapped Delay Line Channel Model is simple, and simply placing registers at the output can implement delay while relative power can be implemented with simple multiplication.

Difficult parts of architecture are OLS- based generator module and Interpolator. OLS-based generator is similar to that of proposed in [8]. However there are some modifications and architecture is illustrated in Figure 3.20. Internal system of control circuits for new Gaussian variates, input selection of FFT/IFFT etc. are not displayed as they're simple Multiplexers depending upon various counters.

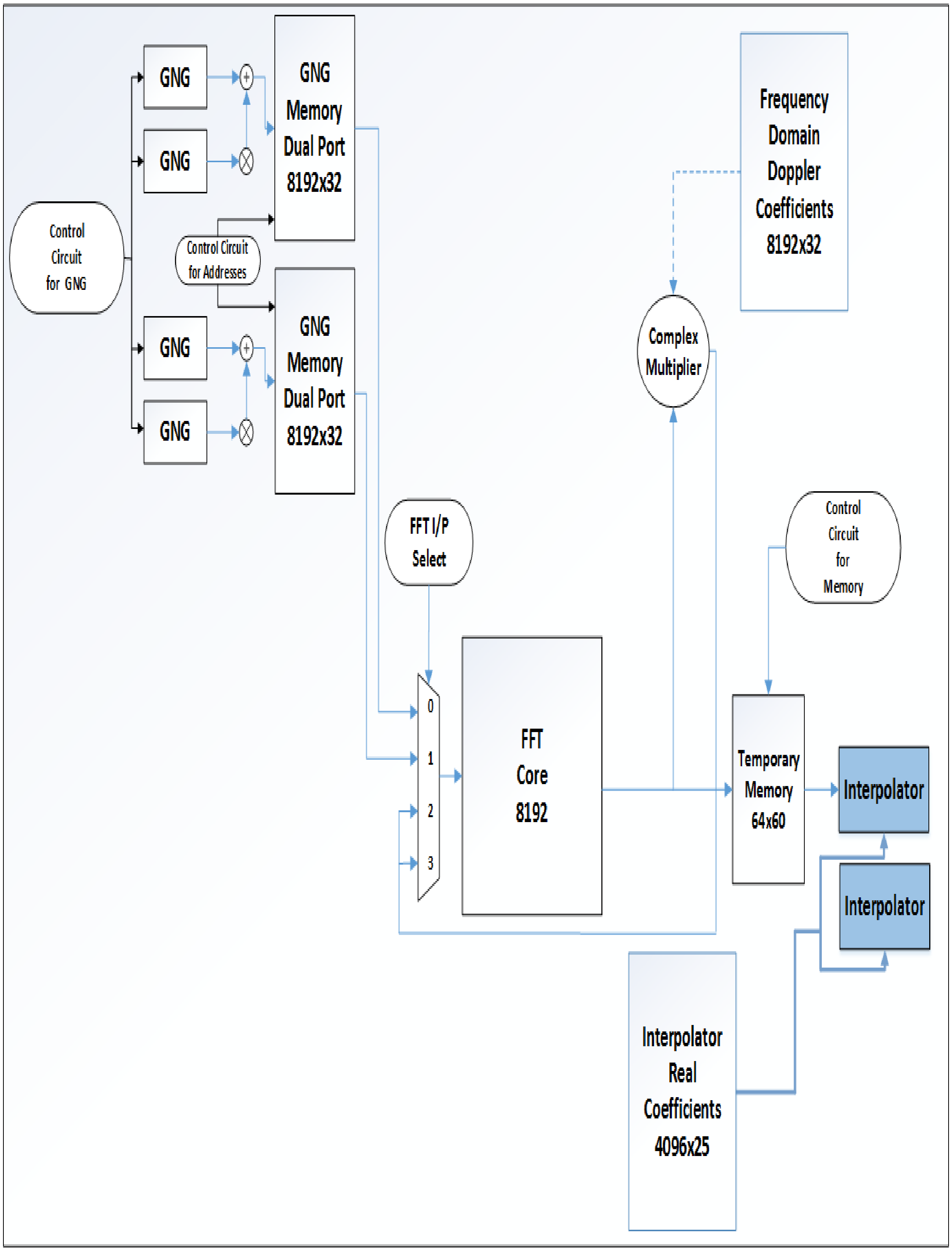

GNG
GNG
GNG
GNG
GNG Memory Dual Port 8192x32
GNG Memory Dual Port 8192x32
Control Circuit for GNG
Control Circuit for Addresses
Frequency Domain Doppler Coefficients 8192x32
Complex Multiplier
FFT I/P Select
Control Circuit for Memory
0
1
2
3
FFT Core 8192
Temporary Memory 64x60
Interpolator
Interpolator
Interpolator Real Coefficients 4096x25

Figure 3.20: Proposed OLS-based Generator Architecture

FSM for OLS-generator is displayed in Figure 3.21; main state is further elaborated in it.

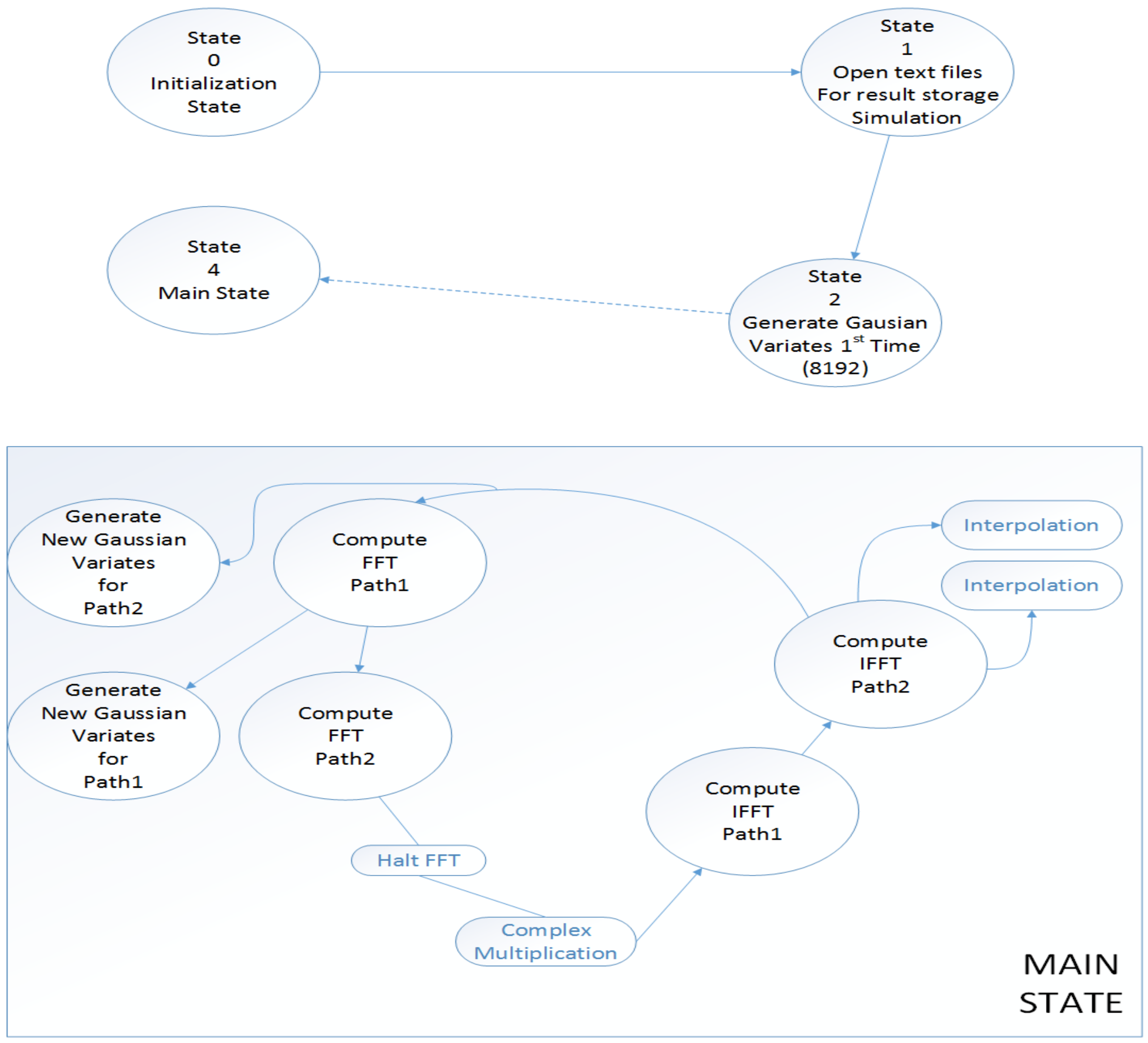

State 0 Initialization State
State 1 Open text files For result storage Simulation
State 4 Main State
State 2 Generate Gausian Variates 1st Time (8192)
Generate New Gaussian Variates for Path2
Compute FFT Path1
Interpolation
Interpolation
Compute IFFT Path2
Generate New Gaussian Variates for Path1
Compute FFT Path2
Compute IFFT Path1
Halt FFT
Complex Multiplication
MAIN STATE

Figure 3.21: FSM of OLS-based Generator

After Rayleigh fading variates are generated, Interpolation is necessary. Architecture of interpolator was finalized by keeping in view total number of necessary multiplications and available clock cycles. With above mentioned system architecture for OLS-based generator, approximately 33000 cycles are available. Maximum multiplications possible in system are 128 (64 ‘*L*’ value meaning 64 non-zero complex multiplications). So if all 128 multipliers are utilized, 4096 coefficients give output in 4096 cycles. With 32k cycles available, comparison is shown in the following table:

Table 3: Available Clock Cycles vs. Required Multipliers

Available Clock Cycles	**Multipliers**
4096	128
8192	64
16384	32
32768	16

With maximum 64 non-zero complex multiplications, there is also a need for specialized hardware for this purpose, and filter coefficients are needed to be stored in a special way. For the minimum available clock cycles condition, there are 64 taps after which a complex multiplier (two real multipliers are present). Now in theory for 4096 coefficients, there is a need of 4096 taps; however 64 are the only non-zero multiplications. Instead of storing 4096 coefficients in a single memory, sixty four memories of 64 memory locations can be used. With each tap multiplication is performed between relevant coefficient and Rayleigh fading variate, hence requirement of 4096 memories and multipliers are not needed. Multiplexer can also be used instead of memories, however for reconfiguration that strategy is not feasible. Also 36 interpolator instances was needed, multiplexers are made up of FPGA internal logic, so it is better to use memories. Pipeline stages are also added to even further reduce critical path, also with this scheme some of pipeline registers can be skipped as this system already provides efficient throughput and lesser clock cycles. 64 complex multiplier based solution is useful with single antenna systems, however for a MIMO based system where a lot of interpolators are needed to emulate all paths, this solution cannot fit in the target FPGA device. So there is a need for a solution with less number of multipliers. Figure 3.22 illustrates the concept of using 64 memories and multipliers.

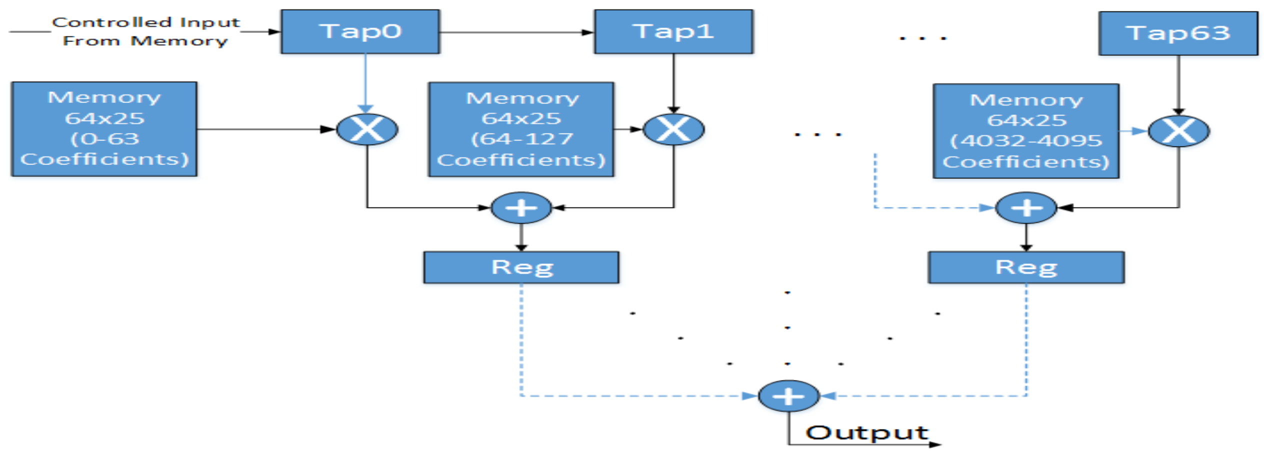

Controlled Input From Memory
Tap0
Tap1
Tap63
Memory 64x25 (0-63 Coefficients)
Memory 64x25 (64-127 Coefficients)
Memory 64x25 (4032-4095 Coefficients)
Reg
Reg
Output

Figure 3.22: Fast Interpolator Architecture having Maximum Resource Consumption

Aforementioned, interpolator has approximately 33k cycles, so a lesser number of multipliers can be used (Table 3). Such implementation is scalable and will consume lesser resources than the targeted board. Now in such a case eight taps has one complex multiplier, now eight memories are required having 512 coefficients. Coefficients are stored in such a way that only relevant coefficient is multiplied with relevant tap. Coefficients are stored in the following way:

Table 4: Interpolator Coefficients Storage Method in Various Memories

Memory	Coefficients
0	0,64,128,192,256,320,384,448,1,65,...,511
1	512,576,640,704,768,832,896,960,512,577,...,1023
2	1024,1088,1152,1216,1280,1344,1408,1472,1025,1089,...,1535
3	1536,1600,1664,1728,1792,1856,1920,1984,1537,1601,...,2047
4	2048,2112,2176,2240,2304,2368,2432,2496,2049,2113,..., 2559
5	2560,2624,2688,2752,2816,2880,2944,3008,3072,2561,2625,...,3071

6	3072,3136,3200,3264,3328,3392,3456,3520,3073,3137,...,3583
7	3584,3648,3712,3776,3840,3904,3968,4032,3585,3649,...,4095

Figure 3.23 displays the concept. Memories are instantiated in OLS-based generator module, so that two interpolators use a single memory.

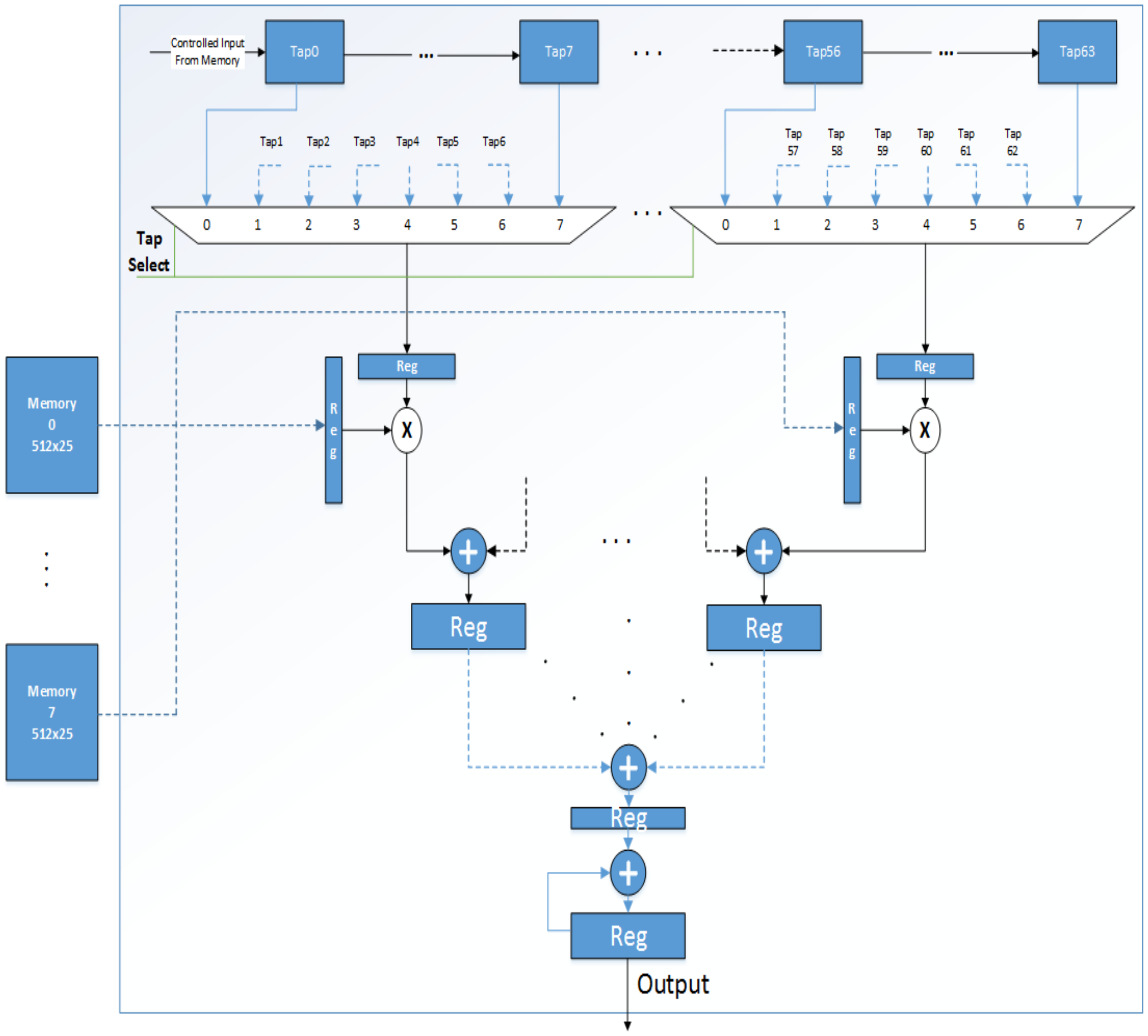


Figure 3.23: Proposed OLS-based Interpolator (Resource Efficient)

Architecture can further be improved by instantiating interpolator coefficients in top module, however critical path is increased and timing requirements are not met due to it. Controlled Input from memory is from a block ram inside interpolator which is 64x32 in size. Simple binary counter selects the multiplier input from taps, and a simple binary counter controls the address of interpolator coefficient memories. Figure 3.24 displays the FSM for interpolator.

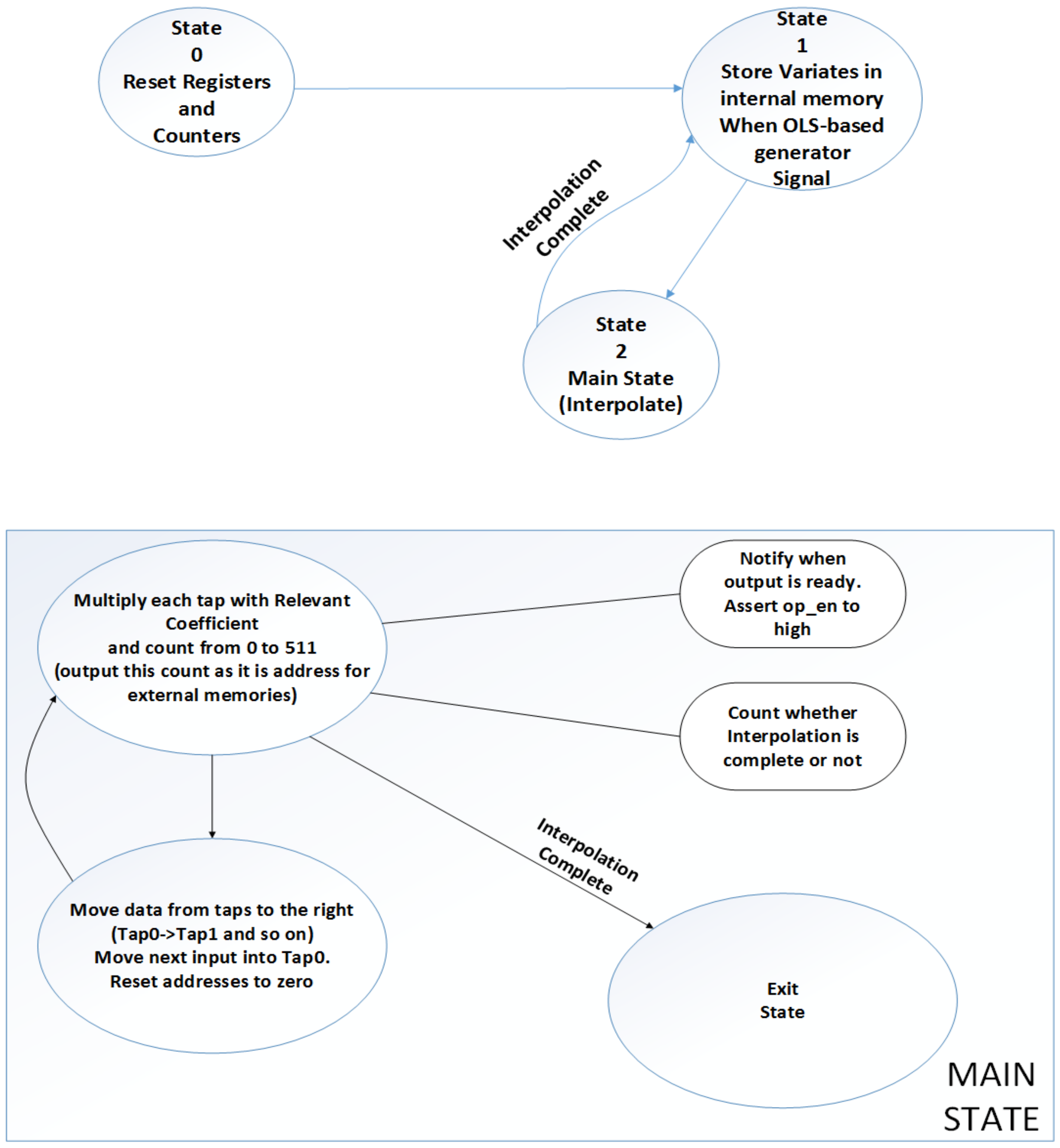


Figure 3.24: FSM of OLS-based Interpolator

Emphasis in this thesis was given on improving single module (OLS-based Generator and OLS-based Interpolator) as MIMO system can be simply made by instantiation. Improvement was made by considering three factors:

- Timing Requirement
- Resource Utilization in a single module
- Resource Utilization for 36 modules (MIMO)

Further Tap delay channel model is simple as various delays are needed to be incorporated and fixed numbers are multiplied with outputs.

In next section i.e. **Evaluation**, various results are shown, and each module’s resource utilization is tabulated.

Chapter 4

Evaluation

CHAPTER 4. EVALUATION

Different FPGA devices offer different resources, for simple designs Spartan-3 series FPGAs can be used. Different family offers different resources, however an FPGA with minimal resources and cost is generally preferred. Also an FPGA evaluation kit is preferred as it is simple to implement it directly on hardware. This thesis requires a device which can accommodate MIMO based system and can provide very high throughputs. For a single antenna system it is very simple to implement it on any FPGA, however for MIMO based systems several considerations are to be sought.

While deciding an FPGA device for the implementation, maximum available Block Memory, DSP slices, Slice LUTs (Look up tables) and flip flops are to be considered. For this thesis Virtex-7 device XC7VX550T was targeted. Some Resources of XC7VX550T are illustrated in Table 5 [34].

Table 5: Some Resources of XC7VX550T

Slices	Block Ram (18K)	Block Ram (36K)	DSP Slices
86600	2360	1180	2880

4.1. System Constraints

34 MSPS throughput was targeted, carrier frequency was set as $f_{carrier}$=600 MHz, moreover available C program of methodology in [8] was used to calculate the Doppler frequencies, and number of new variates per execution is found also by it (value of 'L'). System can support Doppler frequencies from f_d=11.1 Hz to f_d=350.56 Hz. These Doppler frequencies are of speeds from 20 km/h to 631 km/h. Number of new variates per OLS-based Generator's execution range from L= 2^2 to 2^6. Outputs of interpolator was set as 4096, interpolation of 'L' outputs are used to

generate 4096 interpolated outputs. For the simulation purposes, Doppler frequency was set as 16.67 Hz which corresponds to ‘L’ = 4.

4.2. System Verification

System verification is very necessary, as hardware optimizations come after it. A C model of [8] was available so a ‘GNG core’ output based Matlab implementation was done. Firstly the outputs of GNG core and filter coefficients in floating points were stored in a text file (using test bench of GNG core and available C model). Gaussian variates and filter coefficients are then stored in a matrix to be easily accessed by Matlab script. After loading inputs a function which converts floating point into fixed point format (Q (m, n) format) helps to perform fixed point calculations in parallel. Matlab also helped in verifying Time domain solution vs. Frequency domain solution’s result comparison (see Appendix B). Figure 4.1 displays the final normalized outputs (floating point, Matlab based Fixed point and FPGA output).

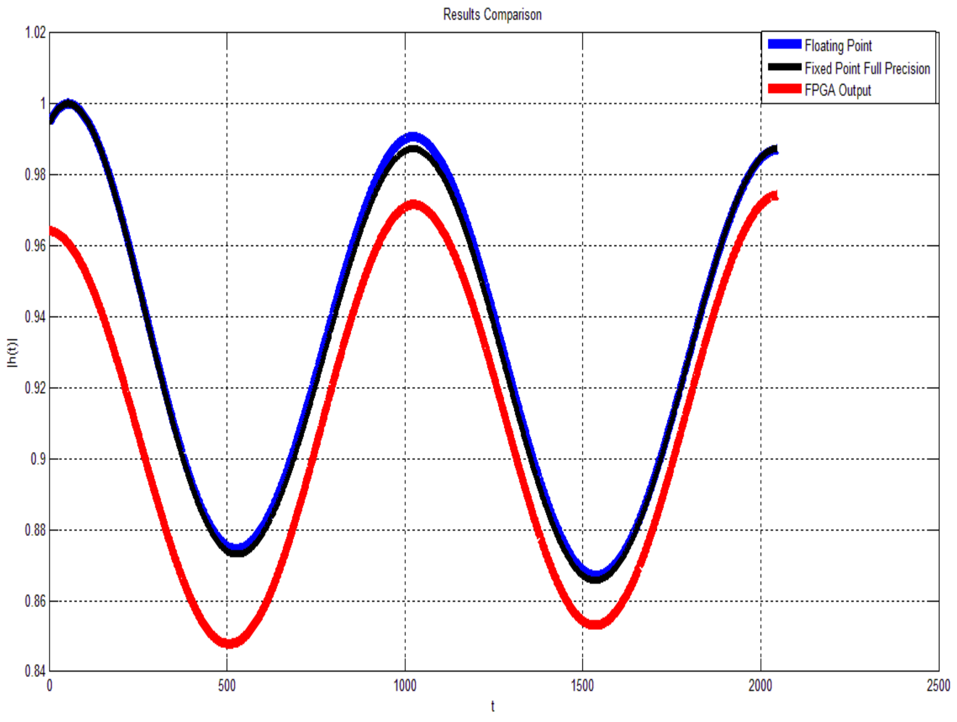


Figure 4.1: Output Comparisons (Matlab Output vs. FPGA)

Difference can be seen in the Matlab based full precision (fixed point) implementation and floating point, however an offset exists between each implementation. In FPGA based implementation after multiplication with filter, result is chopped down into 16 bits, due to which here an offset is visible in the graph, but the output waveform is same.

4.3. System's Simulation Waveform

Second method of scheme 2 discussed in Chapter 3 is simulated first on Xilinx ISE Design Suite 14.7. First time enable signal for interpolator comes after 57645 clock cycles. The signal 'en' when asserted for 'L' number of cycles initiates interpolation. This is displayed in Figure 4.2. Num_Cycles count the number of cycles.

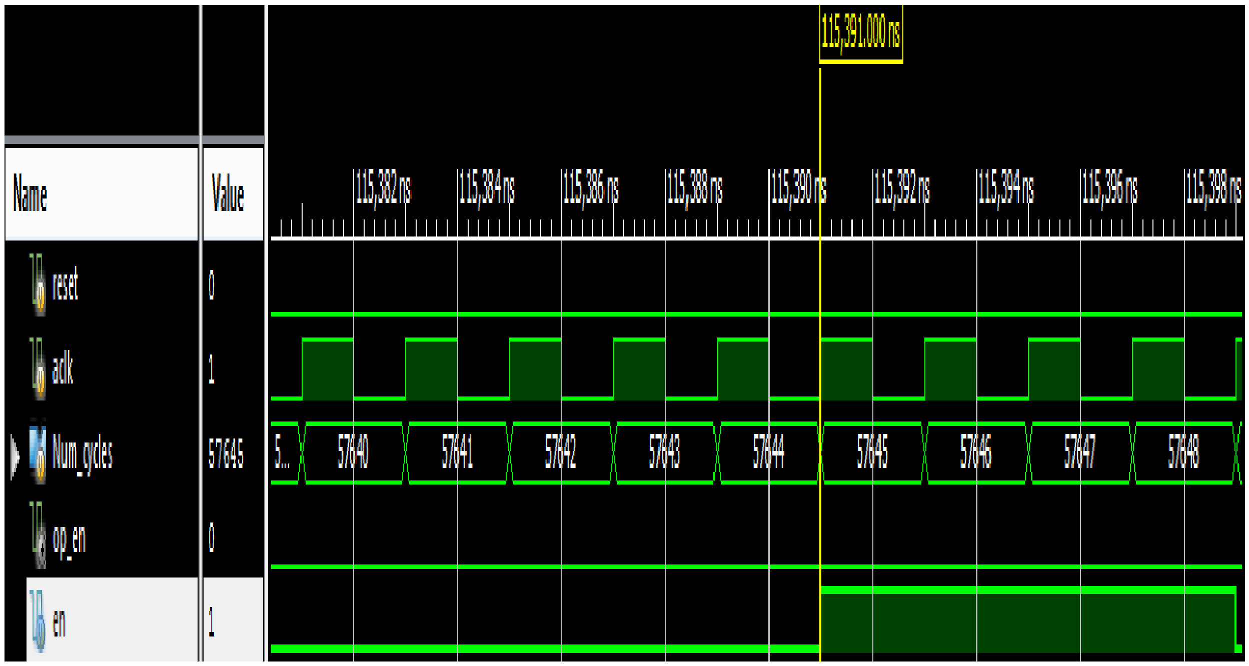


Figure 4.2: Interpolator Initiation Signal

Note that as per simulation parameters 'L' was set as 4, so for four cycles 'en' signal was high. Second time 'en' becomes high when clock cycles count to 90704. This is displayed in Figure 4.3 and Figure 4.4.

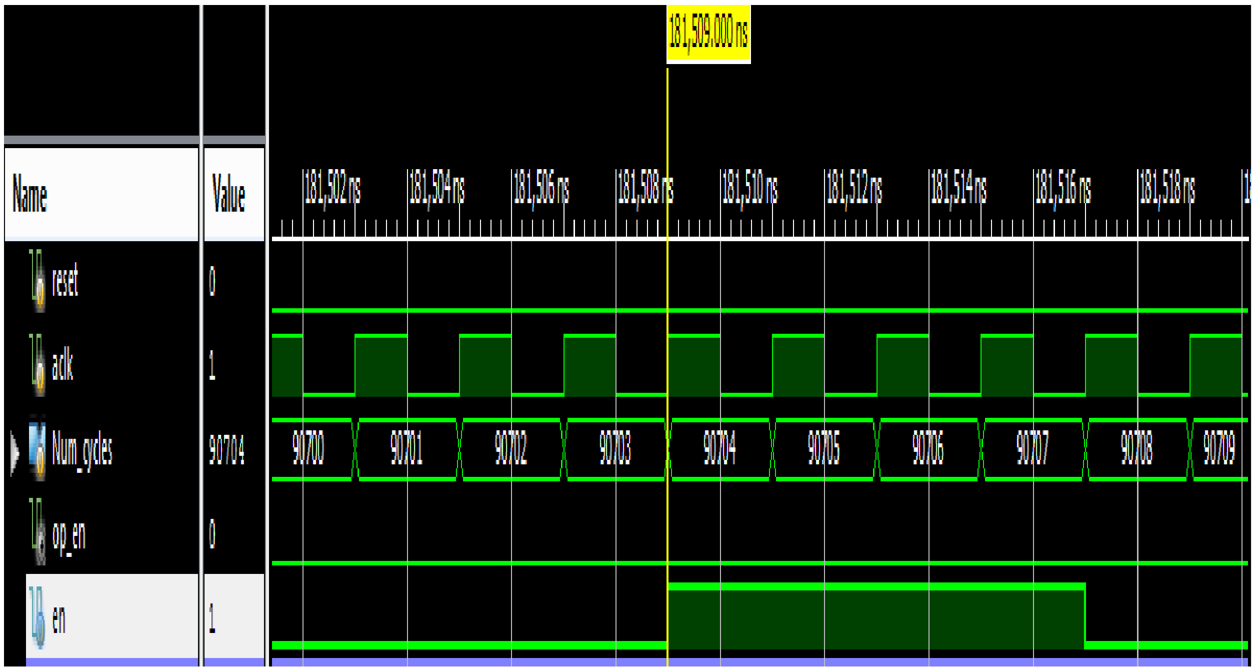


Figure 4.3: Output for 'L' Number of Cycles

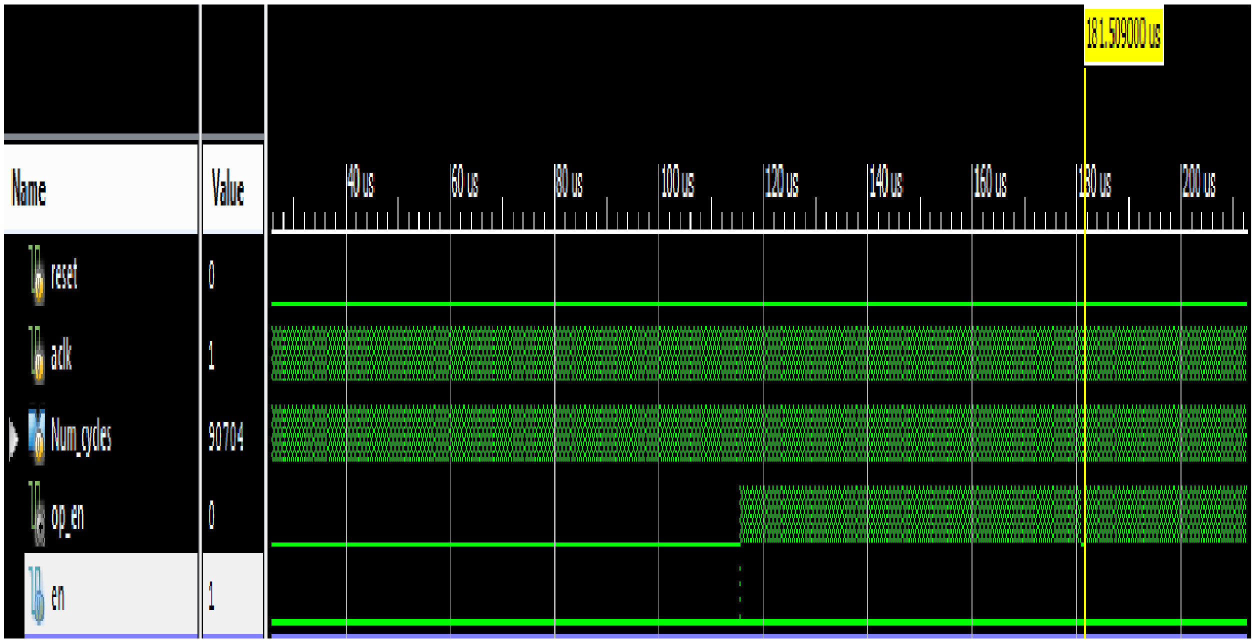


Figure 4.4: Second Time Interpolator Initiation

There are 33059 cycles between outputs, so for 34 MSPS, there is a need of 274422759 cycles, or a frequency of approximately 274 MHz (3.64ns critical path), as already mentioned in System

Performance, both interpolator and OLS-based generator has a critical path less than required (can work on larger frequency), so timing requirements are met in it.

For the outputs there are three signals, two containing data and one showing output data is valid. After initiating the interpolation few cycles are consumed inside the interpolator to store the input data in memory. Signal 'op_en' is asserted when output data is mature, and is raised after every 7 cycles. Figure 4.5 shows the situation in which two samples are output.

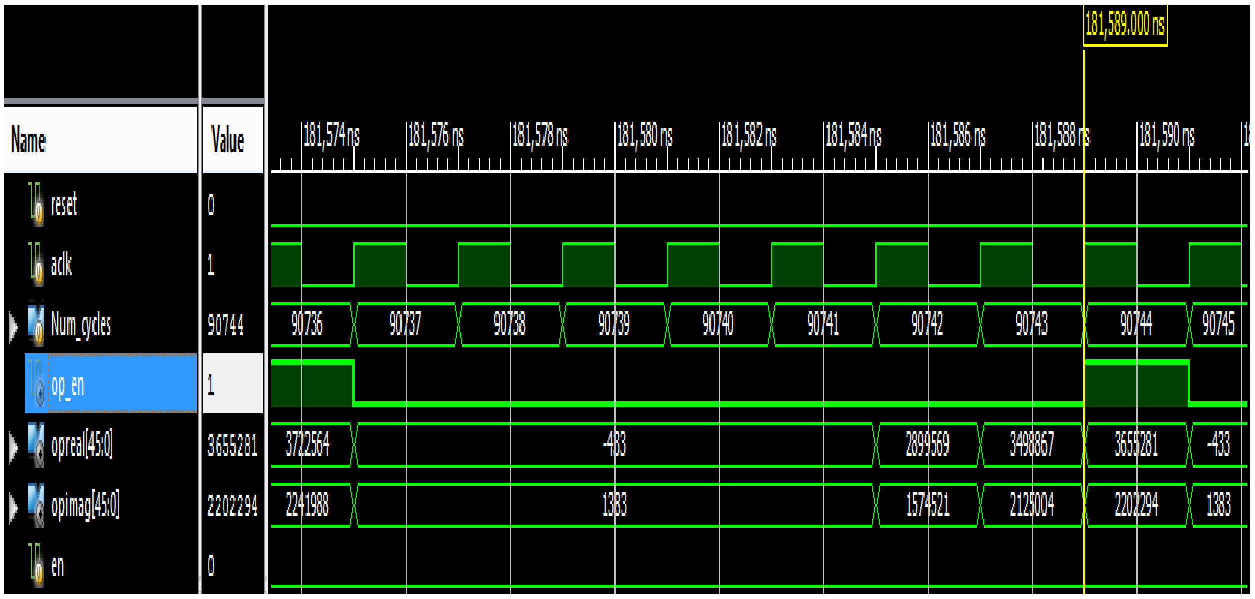


Figure 4.5: Output Samples

Targeted throughput of 34 MSPS is achieved on the targeted device for MIMO system. A single path was also tested using Zynq ZC-702, and results were compared with simulation results.

4.4. System Performance

Emphasis was given on creating a single module, as other modules are instantiated from it. Before estimation of actual resources, following Table 6 show the resource used by some common sub-modules.

Table 6: Resources Used by Common Sub-modules

Sub-Module	Resources	
FFT	52 Block RAMs (Mostly 18k)	25 DSP Slices
GNG Core	1 Block RAM (36k)	2 DSP Slices
Memory for Storing GNG Outputs	8 Block RAMs (1x18k & 7x36k)	
Complex Multiplier	4 DSP Slices	
Doppler Filter Coefficients Memory	8 Block RAMs (36k)	
Interpolator Coefficients Memory	8 Block RAMs (8x18k)	

For the schemes discussed in chapter 3 for various timing requirements, scheme 1 having separate FFT/IFFT, estimated required resources for a single module are shown in Table 7.

Table 7: Estimated Resources for Scheme 1

Module	Resource Type	Expected Utilization
Interpolator	Block RAMs	64
	Multipliers	128
OLS-based Generator	Block RAMs	118
	Multipliers	62

Among 112 Block RAMs, 104 are of FFT module (52 each), 50 multipliers are used also by it (25 each). 64 Block RAMs inside Interpolator holds coefficients data. 64 taps are used so 128 multipliers are needed.

For this scheme 36 modules need to be instantiated. This exceeds the available resources in targeted device, also in the Virtex-7 FPGA's largest device i.e. XC7VX980T, this scheme cannot be implemented. This scheme can work on a system having critical path of 14.7ns (68 MHz).

In the first method of scheme two, in which a single FFT/IFFT module is used. Table 8 show the estimated required resources of this scheme.

Table 8: Estimated Resources for first method in Scheme 2

Module	Resource Type	Expected Utilization
Interpolator	Block RAMs	16
	Multipliers	32
OLS-based Generator	Block RAMs	118
	Multipliers	37

This design can fit in targeted device; however slice LUTs can increase as 36 modules need to be instantiated. This scheme can work on a system having critical path of 7.31ns (136.7 MHz).

Second method in scheme two is feasible, as OLS-based generator can compensate two paths. A very low critical path is required (3.64ns, approximately 274 MHz frequency). This critical path is difficult to achieve however incorporating pipeling between each process and before and after each operation can achieve such a low critical path.

For targeted system 18 OLS-based generators and 36 interpolators were required to be instantiated. One OLS-based generator connects with two interpolators. Resource utilization and operational frequency for a single interpolator are indicated in Table 9.

Table 9: Interpolator Resources

Synthesis Results					Place and Route Results				
Occupied Slices Count	Block RAM (18K)	Block RAM (36K)	DSP Slices	Max Operational Frequency	Occupied Slices Count	Block RAM (18K)	Block RAM (36K)	DSP Slices	Max Operational Frequency
-	1	0	16	400.986 MHz (2.494 ns)	1,351	1	0	16	343.406 MHz (2.912 ns)

These results are for a single module, however for a 2×2 MIMO system, Table 10 show the required resources:

Table 10: Interpolator for MIMO System

Resource Type	Total	1 Module	36 Modules
Slices	86600	1,351	48636
Block RAM (18 K)	2,360	1	36
Block RAM (36 K)	1180	0	0
DSP Slices	2880	16	576

Now for OLS-based generator, 18 modules need to be instantiated. Resource utilization and operational frequency for a single OLS-based Generator are indicated in Table 11.

Table 11: OLS-based Generator Resources

Synthesis Results					Place and Route Results				
Occupied Slices Count	**Block RAM (18K)**	**Block RAM (36K)**	**DSP Slices**	**Max Operational Frequency**	**Occupied Slices Count**	**Block RAM (18K)**	**Block RAM (36K)**	**DSP Slices**	**Max Operational Frequency**
-	62	28	69	388.086 MHz (2.577 ns)	4457	62	28	69	275.41 MHz (3.631 ns)

These results are for a single module, which incorporates two paths, Table 12 show the required resources for a 2×2 MIMO based system:

Table 12: OLS-based Generator Resources for MIMO system

Resource Type	Total Resources	1 Module	18 Modules (36 Paths)
Slices	86600	4457	80226
Block RAM (18 K)	2,360	62	1116
Block RAM (36 K)	1180	28	504
DSP Slices	2880	69	1242

From the results in above table, it can be seen that thirty six modules can be instantiated on the targeted device. Hence the required task is achieved on a single device. Now for the path delays and power multiplication, few registers and multipliers are needed, which can be simply added inside the device, or another device can be connected with it (having fewer resources).

Chapter 5

Conclusions and Future Work

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

Over-the-air (OTA) field trials are difficult, require expensive equipment and are tedious. OTA should be used only where it is the only option left, as already available channel models can simplify the task and are very efficient in channel emulation.

In this thesis, a high throughput based channel emulator has been implemented on FPGA, which emulates a Rayleigh Fading Channel with Doppler effects. The objective of the thesis was to implement an IP implementing OLS method providing a throughput of 34 MSPS while utilizing minimum hardware. In order to reduce the hardware used, at first FFT/IFFT core is utilized maximally in variates generator block and secondly in place of interpolation in frequency domain (as proposed in [8]), time domain based interpolator incorporating time domain convolution is proposed to exploit the presence of zero padding at the input of interpolator which enabled to reduce the hardware cost of interpolator.

For a 2×2 MIMO system with multi path effects (nine taps/paths), 36 instances of a SISO system without multipath are required, channel emulator was implemented keeping in view this requirement. Main focus was set to achieve timing requirement with least resources. Timing requirement was set as 34 MSPS, which was successfully achieved.

Before implementation of system on FPGA, thorough study was done on different methods in channel emulation, different FFT computation methods, fixed vs. floating point implementation, and also on available resources. After it various implementations were done in order to achieve the best design. Actual results from FPGA were stored and compared with Matlab results (floating and fixed point) for on chip validation.

The work under the scope of this thesis motivates author to further pursue the work in future and select a different technique in channel emulation. For higher Doppler frequencies than the targeted ones, implemented OLS-based generator can be used for implementation and time domain interpolator is scalable as well. Frequency domain interpolation can also be studied and implemented in future to even make system more scalable; however higher consumption of resources in this technique requires a thorough study first to reduce complexity. Single FFT core based system (with same FFT core for interpolation) can be realized, however achieving targeted throughput is a long challenging task. Further, MIMO systems greater than 2×2 can also be targeted, which is also a challenging task. Massive MIMO concept in 5G technology also requires channel emulation and new techniques and methods can be studied for such implementation.

REFERENCES

[1] H. Tullberg, H. Droste, M. Fallgren, P. Fertl, D. Gozalvez-Serrano, E. Mohyeldin, O. Queseth and Y. Selén, "METIS research and standardization: A path towards a 5G system," in *2014 IEEE Globecom Workshops (GC Wkshps)*, Austin, TX, 2014.

[2] T. M. Fernández-Caramés, M. González-López and L. Castedo, "FPGA-based vehicular channel emulator for evaluation of IEEE 802.11p transceivers," in *2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST)*, Lille, October 2009.

[3] C. DeMartino, "New Technology Redefines Channel Emulation for 5G Millimeter-Wave Systems," Microwaves & RF, 20 10 2017. [Online]. Available: https://www.mwrf.com/test-measurement/new-technology-redefines-channel-emulation-5g-millimeter-wave-systems.

[4] T. Rappaport, Wireless Communications: Principles and Practice, Prentice Hall PTR, 1996.

[5] B. Sklar, Rayleigh fading channels in mobile digital communication systems .I. Characterization, IEEE, 1997.

[6] A. Goldsmith, WIRELESS COMMUNICATIONS, Cambridge University Press, 2005.

[7] W. C. Jakes, Microwave Mobile Communications, IEEE Press, 1993.

[8] J. Yang, C. A. Nour and C. Langlais, "Correlated Fading Channel Simulator," *IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS,* vol. 12, no. 6, pp. 3060-3071, 2012.

[9] Anritsu, "ACE RNX Channel Emulator," 2019.

[10] Gigacomp, MRC, "Wideband Radio Channel Emulator EB Propsim C2," 2015.

[11] Keysight Technologies, "Propsim F8 Channel Emulator," 2016.

[12] Pätzold, Matthias, C.-X. Wang and a. B. O. Hogstand, "Two New Sum-of-Sinusoids-Based Methods for the Efficient Generation of Multiple Uncorrelated Rayleigh Fading Waveforms.," *IEEE Transactions on Wireless Communications*, vol. 8, no. 6, p. 3122–3131, 2009.

[13] C.-D. Iskander, "A MATLAB-based object-oriented approach to multipath fading channel simulation," Feb.2018. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/18869-a-matlabbased-objectoriented-approach-to-multipath-fading-channel-simulation.

[14] C. Komninakis and J. F. Kirshman, "Fast Rayleigh fading simulation with an IIR filter and polyphase interpolation," *RF Design,* pp. 24-34, July 2004.

[15] D. Young and N. Beaulieu, "JOHN I. SMITH," *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY,* vol. 24, no. 3, pp. 39-40, August 1975.

[16] D. Young and N. Beaulieu, "The generation of correlated Rayleigh random variates by inverse discrete Fourier transform," *IEEE Transactions on Communications,* vol. 48, no. 7, pp. 1114-1127, July 2000.

[17] P. Huang, Y. Du and Y. Li, "Stability Analysis and Hardware Resource Optimization in Channel Emulator Design," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS,* vol. 63, no. 7, pp. 1089-1100, July 2016.

[18] S. Fard, A. Alimohammad and B. Cockburn, "Single-field programmable gate array simulator for geometric multiple-input multiple-output fading channel models," *IET Communications,* vol. 5, no. 9, pp. 1246-1254, June 2011.

[19] A. Alimohammad and S. F. Fard, "FPGA Implementation of Isotropic and Nonisotropic," *IEEE Transactions on Circuits and Systems II,* vol. 60, no. 11, pp. 796-800, Nov. 2013.

[20] A. Alimohammad, S. F. Fard, B. F. Cockburn and C. Schlegel, "A Compact Single-FPGA Fading-Channel Simulator," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS,* vol. 55, no. 1, pp. 84-88, JANUARY 2008.

[21] A. N. S. f. R. C. G. W. C. o. t. I. i. FPGA, "Pengda Huang," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II,* vol. 63, no. 2, pp. 216-220, FEBRUARY 2016.

[22] Q. Zhu, H. Li, Y. Fu, C.-X. Wang, Y. Tan, X. Chen and Q. Wu, "A Novel 3D Non-Stationary Wireless MIMO Channel Simulator and Hardware Emulator," *IEEE TRANSACTIONS ON COMMUNICATIONS,* vol. 66, no. 9, pp. 3865-3878, SEPTEMBER 2018.

[23] Z. Sheng-kui, Z. Qiu-ming, D. Xiu-chao, L. Xing-lin and C. Xue-qiang, "A Novel Method for Generation of Correlated Composite Fading," *JOURNAL OF APPLIED SCIENCES—Electronics and Information Engineering,* vol. 33, no. 5, pp. 470-480, September 2015.

[24] A. V. Oppenheim and R. W. Schafer, Discrete-time Signal Processing, Prentice-Hall, 1999.

[25] S. He and M. Torkelson, "A New Approach to Pipeline FFT Processor," in *Proceedings of International Conference on Parallel Processing*, Honolulu, HI, USA, 1996.

[26] M. ALI, "PIPELINED FAST FOURIER TRANSFORM PROCESSOR," Tampere, Finland, January 2017.

[27] A. R. Jafri, J. Majid, M. A. Shami and M. A. Imran, "Hardware Complexity Reduction in Universal Filtered Multicarrier Transmitter Implementation," *IEEE Access,* vol. 5, pp. 13401-13408, 2017.

[28] G. Liu, *Gaussian Noise Generator Core Specification,* OpenCores, Jan. 29, 2015.

[29] Xilinx, *LogiCORE IP Fast Fourier Transform v8.0,* Xilinx, July 25, 2012.

[30] Xilinx, *Complex Multiplier v6.0,* Xilinx, November 18, 2015.

[31] Xilinx, *DSP48 Macro v3.0,* Xilinx, November 18, 2015.

[32] Xilinx, *Block Memory Generator v8.3,* April 5, 2017: Xilinx.

[33] A. d. l. Oliva, J. A. Hernandez, D. Larrabeiti and A. Azcorra, "An overview of the CPRI specification and its application to C-RAN based LTE scenarios," *IEEE Communications Magazine,* vol. 54, no. 2, pp. 152-159, February 2016.

[34] Xilinx, *7 Series FPGAs Data Sheet: Overview,* February 27, 2018.

APPENDIX A. SELECTION OF VARIOUS PARAMETERS

The maximum sampling frequency in LTE is 30.72 MHz, from which 34 Mega Samples per Second rate was decided to be achieved. Minimum throughput needs to be 30.72 MSPS, but an interface might be required between various systems working on different clocks, so the 34 MSPS output rate was selected.

FFT core requires various DSP slices and Block RAMs, system was to be designed keeping in view that several instances of a single module is required (36). So selection of an FFT size is important, 8192 size was selected. Also available C model used an interpolator’s output of 4096, so an FFT size of its double is feasible.

Depending upon the Doppler frequency, C model calculates the number of ‘L’ variates to be generated. Actually an intermediate sampling frequency $\tilde{f}_s$ is selected which decides how much upsampling is required. This frequency is selected by keeping the ratio $f_s\ /\tilde{f}_s$ to a value which is a power of 2. Ratio decides the number of zeros to be inserted between each output samples, as in original design interpolation was done in frequency domain.

APPENDIX B. COMPARISON OF FILTER COEFFICIENTS

For the Doppler Filter coefficients, Q (1, 15) format was used to store them. 16 bits were selected because FFT output's 25 bits are used. Maximum inputs to DSP slice can be 25 and 18 bits, so FFT output was allocated more bits. Mean Square Error (MSE) between fixed and floating point is in 10^{-12} as most information is stored in fractional part.

For the Interpolator Coefficients, Q (1, 24) format was used to store them. Mean Square Error (MSE) between fixed and floating point is in 10^{-15}. Reducing fractional part reduces the accuracy to a large extent, as most coefficients are truncated to zero.