Design and Implementation of an FPGA-based Adaptive filter Single-User Receiver

Prinya Atiniramit

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Peter M. Athanas, Chair
Mark T. Jones,
Jeffrey H. Reed.

September 17, 1999
Blacksburg, Virginia

Keywords: FPGA, CDMA, CCM, adaptive filter receiver

Design and Implementation of an FPGA-based Adaptive filter Single-User Receiver

Prinya Atiniramit

(ABSTRACT)

During the last decade, the wireless communications industry has grown rapidly. Driven by market demand, service providers are continuously looking for better systems. The main focus of continued research has been to increase the quality of services and system capacity. The Code Division Multiple Access (CDMA) cellular system had been proposed for use as a new standard for cellular telephone systems.

A great deal of research has been conducted to develop receiver structures useful for CDMA systems. Traditional receivers such as the correlation and RAKE receivers are vulnerable to the near-far problem, i.e., the problem encountered when one received signal power is stronger than another. This problem is common in mobile environments.

For single-user receivers, adaptive filtering techniques can be employed to alleviate multiple access interference and the near-far problem. In this thesis, an adaptive filter receiver is implemented on the FPGA-based configurable computing platform called GigaOps G900. By using FPGAs, designers can implement special-purpose signal processing architectures using specialized data paths, optimized sequencing, and pipelining while still providing some flexibility. This results in better overall system performance, resource utilization, and reduced power consumption.

Acknowledgements

I would like to thank Professor Peter M. Athanas for being my advisor, and for his guidance and support throughout my graduate studies. I would also like to thank Professors Mark T. Jones and Jeffrey H. Reed for being on my committee and supplying corrections and comments to this thesis.

I am deeply grateful for all the help I have received during the course of this thesis at Virginia Tech. A special thanks is owed to John Davies and Nitin Mangalvedhe for their work and support of this thesis. I would like to thank Scott Harper, Michael Hosemann, Luke Scharf, and Srikathyayani Srikanteswara for their support over the past two years.

This thesis is dedicated to my parents and my wife for their endless support and encouragement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the last decade, the wireless communications industry has grown rapidly. Driven by market demand, service providers are continuously looking for better systems. The main focus of continued research has been to increase the quality of services and system capacity. The Code Division Multiple Access (CDMA) cellular system had been proposed for use as a new standard for cellular telephone systems. This system not only provides higher capacity than current technologies; it also gives the additional benefits of digital systems, including improved privacy and error correction.

A great deal of research has been conducted to develop receiver structures useful for CDMA systems. Traditional receivers such as the correlation and RAKE receivers are vulnerable to the near-far problem, i.e., the problem encountered when one received signal power is stronger than another. This problem is common in mobile environments.

For single-user receivers, adaptive filtering techniques can be employed to alleviate multiple access interference and the near-far problem. In this thesis, an adaptive filter receiver is implemented on the FPGA-based configurable computing platform called GigaOps G900. In recent years, field programmable gate arrays (FPGAs) have gained popularity for rapid prototyping of digital systems. By using FPGAs, designers can implement special-purpose signal processing architectures using specialized data paths, optimized sequencing, and pipelining while still providing some flexibility. This results in better overall system performance, resource utilization, and reduced power consumption.

1.1 Motivation

Because of their finite impulse response (FIR) filter structures, adaptive filter-based receivers are computationally demanding, requiring a great deal of complex-number multiplications and additions. Another restriction imposed by the receivers is that they cannot tolerate much output latency due to the dependency of the operations on the previous output. Thus, the involved calculations must be performed in real-time with relatively low latency.

Traditional DSP processors, which adopt a traditional von Neuman architecture, are not well suited to deliver such computational power. Even though DSP processors have wide internal data paths, which positively impact the accuracy and usability of the implemented algorithm, the processors must perform each operation sequentially. Thus maximum throughput decreases as the numbers of operations increases. Even though multiple DSP processors can be used, the partitioning of algorithms is difficult and requires substantial overhead, and for some algorithms this approach may not be possible. Another disadvantage is that the number of input/output interfaces available is limited for software radio applications.

FPGA-based computing platforms, on the other hand, allow designers to implement customized architectures. By using pipelining techniques, the receiver algorithm can be divided into smaller operations, which can be run in parallel. Thus, the throughput is increased and the output latency is decreased.

This thesis addresses the practical implementation of an adaptive filter single-user receiver. The selected technique is called the differentially coherent adaptive receiver, which has been proven to perform well in mobile environments [1]. The implemented receiver employs a stream-based architecture [2]. In this architecture, user data is processed serially through a string of processing pipelines, which run in parallel. This increases the throughput and also minimizes the overall routing and communication requirements between pipelines. The architecture also provides some flexibility. By allowing users to reconfigure the parameters of the processing pipelines, the functionality of the receiver can be changed.

1.2 Contributions of this Work

This thesis presents a prototype adaptive filter-based single-user receiver implemented on a configurable computing machine (CCM). The designed receiver provides some flexibility through changing the length of the FIR filter and the step-size for the coefficient update algorithm. The maximum clock speed that this system can run is 20 MHz. Although this prototype does not implement some features found in contemporary wireless receivers, it still provides adequate proof of the adaptive filter receiver concept and insight into its design, as well as FPGA-based system implementation.

The primary contribution of this thesis lies in the development of a special-purpose signal processing architecture. The architecture employs a series of processing pipelines with bi-directional data flow among them. Received data is processed in these pipelines at high speed in parallel and then passed to the next pipeline. This in turn increases the system throughput while reducing the latency. Some processing pipelines also employ pipelining techniques inside. This greatly increases the system throughput with a small increase in latency.

Due to limited computing resources, some computational units in pipelines are used for several operations. By running the system at a higher clock speed than the incoming data rate, such units can be reused to execute one operation per clock cycle. The architecture also incorporates some flexibility. By having programming information, the processing pipeline's parameters can change, which in turn changes the receiver parameters.

1.3 Organization of Thesis

The thesis is organized as follows. Chapter 2 provides an introduction to CDMA systems and an insight into different kinds of adaptive filter receivers. Chapter 3 introduces the stream-based architecture, which is employed in the implemented radio. This chapter also provides some background on field programmable gate array devices as well as the configurable platform used in this thesis. In Chapter 4, the system architecture is introduced. This chapter also discusses partitioning issues, which impact the flexibility.

Chapter 5 gives the hardware results in different mobile environments. The thesis ends with Chapter 6, which presents the conclusion and suggestions for future work.

Chapter 2

DS/SS CDMA System

This chapter starts with an overview of DS/SS CDMA systems. Then it introduces the fundamental theory of adaptive filters as well as a class of single-user adaptive receivers. The chapter concludes with the adaptive receiver structure selected for this implementation.

2.1 Introduction

First developed for military applications in World War II due to its anti-jamming capabilities, spread-spectrum systems have gained attention over the last few decades. However, there had not been much unclassified research in this area until 1993, when QUALCOMM Inc. established the IS-95 code division multiple access (CDMA) standard. This was the first widespread commercial spread-spectrum system. With the promise of better performance than the existing systems, a great deal of research has been conducted to improve the performance and capacity of these systems. CDMA promises capacity improvements over time division multiple access (TDMA) and frequency division multiple access (FDMA). Its capacity is, however, limited by multiple access interference (MAI) and the near-far problem, rather than by noise, which limits the other two systems. Multiple access interference is inherent to CDMA systems. It occurs when there is more than one signal transmitted simultaneously at the same carrier frequency. The near-far problem, on the other hand, is not inherent to CDMA. This problem occurs when one signal power is much stronger than another due to the non-ideal power control mechanism of the system. These two problems impose capacity limitations on

conventional receiver structures. The conventional receivers are matched filters, whose coefficients are set to the desired user's spreading codes. Even while these receivers perform well in additive white Gaussian noise (AWGN) environments, their performance is greatly reduced with the presence of MAI and disparities in the received signal powers. This motivates researchers to seek new receiver structures.

Receivers in the CDMA system can be classified into two kinds: multiuser receivers and single-user receivers. Multiuser receivers are used at the base stations to demodulate all mobile units in the cell. These receivers must know the spreading codes of all of the users, which can also be used in the interference cancellation algorithms. On the other hand, a single-user receiver is used in a mobile unit, which only possesses knowledge of its own spreading code.

In this thesis a single-user receiver structure is implemented on a configurable computing platform, i.e., a fractionally-spaced adaptive filter receiver. This filter performs well in multipath channels. It performs interference rejection and coherent multipath combination as an equalizer and despreading as a matched filter. Details on this receiver can be found in [3] and [4].

2.2 System Model

The complex bandpass representation of the k^{th} user's transmitted signal is given by,

$$x(t) = \sqrt{P_k} \sum_{m=-\infty}^{\infty} d_k(m) s_k(t - mT - \tau_k) e^{i[\omega_k (t-\tau_k)+\theta_k]}, \qquad (2.1)$$

where P_k is the k^{th} user's signal power; $d_k(m)$ is the k^{th} user's data symbol, $d_k(m) \in \{-1,+1\}$; $s_k(t)$ is the k^{th} user's spreading waveform; T is the symbol period; and $k = 1,2,...,K$. τ_k is the k^{th} user's delay, which is uniformly distributed over the period $[0,T]$. ω_k is the k^{th} user's carrier frequency, which is equal to $\omega_k = \omega_c + \Delta\omega_k$, where ω_c is the nominal carrier frequency, and $\Delta\omega_k$ is the carrier frequency offset. The spreading waveform of User k is given by

$$s_k(t) = \sum_{n=0}^{N-1} s_{kn} \Pi_{T_c}(t - nT_c), \qquad 0 \le t \le T, \tag{2.2}$$

where $s_k(t) = 0$ for $t \notin [0,T]$, s_{kn} is the n^{th} chip of the k^{th} user's spreading code, and $\prod_{Tc}(t)$ is the rectangular pulse of duration T_c.

In this thesis, the code-on-pulse modulation or modulation-on-symbol is employed; i.e., the period of the spreading code is equal to the duration of each symbol. At the receiver, the transmitted signal is downconverted and low-pass filtered. The output of this low-pass filter (LPF) is sampled to produce baseband digital data. For simplicity, assume for now that the receiver is perfectly synchronized with the desired user's signal, User 1. The baseband signal, after downconversion, can be expressed as,

$$r(t) = \sum_{k=1}^{K} r_k(t) + n(t), \tag{2.3}$$

where $r_k(t)$ is the k^{th} user's downconverted signal,

$$r_k(t) = \sqrt{P_k} \sum_{m=-\infty}^{\infty} d_k(m) s_k(t - mT - \tau_k) e^{j[\Delta\omega_k (t-\tau_k) - \omega_c \tau_k + \theta_k]}, \tag{2.4}$$

and $n(t)$ is a complex white Gaussian noise,

$$n(t) = n_r(t) + j \cdot n_i(t). \tag{2.5}$$

With the above assumption, $P_1 = 1$, $\tau_1 = 0$, $\Delta\omega_1 = 0$, and $\theta_1 = 0$. The sampled received signal can be written as,

$$r(m) = \sum_{k=1}^{K} a_k d_k(m) s_k(m - m_k) e^{j\left(2\pi \frac{f_k}{f_s} m + \theta_k\right)} + n(m), \tag{2.6}$$

where a_k is the amplitude of the k^{th} user's received signal; m_k is the delay of the k^{th} user; f_k is the k^{th} user's carrier frequency offset; θ_k is the phase offset.

2.3 Mean Square Error Minimization

Adaptive filters are useful for estimating a signal which has been sent through an unknown channel with changing characteristics. The concept is to minimize the mean squared error, the difference between the desired response and the filter estimate. This minimum mean square error (MMSE) criterion is the basis of Wiener filter theory and in a stationary AWGN environment leads to the optimum Wiener solution for a linear filter. Let **r**(m) be a vector formed by the received samples, from Equation 2.6, corresponding to the m^{th} symbol,

$$\mathbf{r}(m) = \left[r_0(m)\, r_1(m) \cdots r_{(N-1)}(m)\right]^{\mathbf{T}}. \tag{2.7}$$

Let **w** be a filter weight vector,

$$\mathbf{w} = \left[w_0\ w_1 \cdots w_{N-1}\right]^{\mathbf{T}}. \tag{2.8}$$

The filter output is the product of **w** and **r**(m),

$$y_1(m) = \mathbf{w}^{\mathbf{H}}\mathbf{r}(m) = \mathbf{r}(m)^{\mathbf{H}}\mathbf{w}. \tag{2.9}$$

The error is the difference between the desired response, $d_1(m)$, and the filter output, $y_1(m)$,

$$e(m) = d_1(m) - y_1(m). \tag{2.10}$$

The expected value of the squared error is given as,

$$\mathbf{E}\left[e^2(m)\right] = \mathbf{E}\left[d_1^2(m)\right] - \mathbf{E}\left[d_1(m)\mathbf{r}^{\mathbf{H}}(m)\right]\mathbf{w} - \mathbf{w}^{\mathbf{H}}\mathbf{E}\left[d_1^*(m)\mathbf{r}(m)\right] + \mathbf{w}^{\mathbf{H}}\mathbf{E}\left[\mathbf{r}(m)\mathbf{r}^{\mathbf{H}}(m)\right]. \tag{2.11}$$

Let **p** denote a cross-correlation vector between the desired response and the received signal vector **r**(m),

$$\mathbf{p} = \mathbf{E}\left[d_1^*(m)\mathbf{r}(m)\right], \tag{2.12}$$

and let **R** denote an auto-correlation of the received signal as,

$$\mathbf{R} = \mathbf{E}\left[\mathbf{r}(m)\mathbf{r}^{\mathbf{H}}(m)\right]. \tag{2.13}$$

After substituting these equations into equation 2.11 we obtain,

$$\mathbf{E}\left[e^2(m)\right] = \mathbf{E}\left[d_1^2(m)\right] - \mathbf{p}^{\mathbf{H}}\mathbf{w} - \mathbf{w}^{\mathbf{H}}\mathbf{p} + \mathbf{w}^{\mathbf{H}}\mathbf{R}\mathbf{w}. \tag{2.14}$$

To minimize the MSE, the derivative of the error function with respect to the filter weights should equal zero, i.e.,

$$\nabla = \begin{bmatrix} \frac{\partial \mathbf{E}\left[e^2(m)\right]}{\partial w_0} \\ \frac{\partial \mathbf{E}\left[e^2(m)\right]}{\partial w_2} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial \mathbf{E}\left[e^2(m)\right]}{\partial w_{N-1}} \end{bmatrix} = -\mathbf{p} + \mathbf{R}\mathbf{w} = 0.$$

The optimal weight vector w_{opt} is,

$$w_{opt} = \mathbf{R}^{-1}\mathbf{p}. \tag{2.15}$$

The above equation is known as the *Wiener-Hopf* equation. When the input is stationary, the error function is quadratic and can be viewed as a single concave hyperparaboloid with a unique minimum point. The filter weights adapt towards this minimum point.

2.4 Cyclostationary

As illustrated above, in a stationary AWGN environment, the error surface is a single concave hyperparaboloid, and even a time-invariant filter would be able to achieve the optimum Wiener solution, provided that the input signal characteristics are known. In a cyclostationary system, the error surface is not unique, and the use of time-invariant filters cannot achieve the optimum Wiener solution. However, for a cyclostationary signal, there is a significant value of spectral correlation of the signal that can be exploited by an adaptive filter [3]. With a proper design, DS/SS signals exhibit cyclostationarity. Chen [5] has shown that if the period of the spreading code and that of the data are integrally related to the chip period, then the DS/SS signal has a single fundamental periodicity equal to the code period and there is a significant degree of spectral correlation. Time-dependent adaptive filters (TDAF) can exploit this correlation to obtain a better estimation of the signals than can their time-independent counterparts [3]. It is for this reason that the system implemented in this thesis uses code-on-pulse modulation.

2.5 Fractionally-Spaced Adaptive Receivers

One form of TDAF is Fourier series representation (FSR) TDAF [6] shown in Figure 2.1, also known as the frequency-shift (FRESH) filter [7]. The FSR-TDAF filters different frequency-shifted versions of the signals according to the periodicities of the signal to be exploited using a bank of filters. Here, $\alpha_0, \alpha_1, \ldots, \alpha_{L-1} \in \alpha$ is the set of all harmonics for which the cyclic autocorrelation function is non-zero. More detail can be found in [3].

Aue and Reed [8] have shown that when this filter is used for symbol detection, this structure can be reduced to a single linear transversal filter, which we call the complex-weight fractionally-spaced linear adaptive receiver (CW-FS-LAR). Figure 2.2 illustrates the structure of this filter.

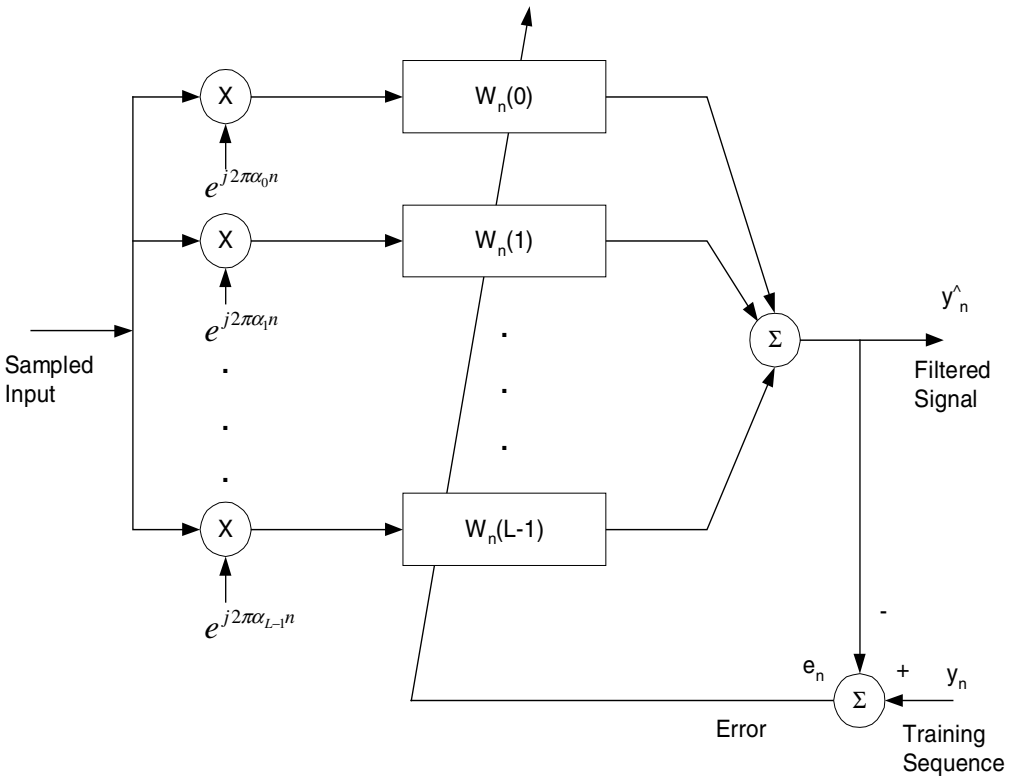


Figure 2.1: Fourier series representation (FSR) TDAF.

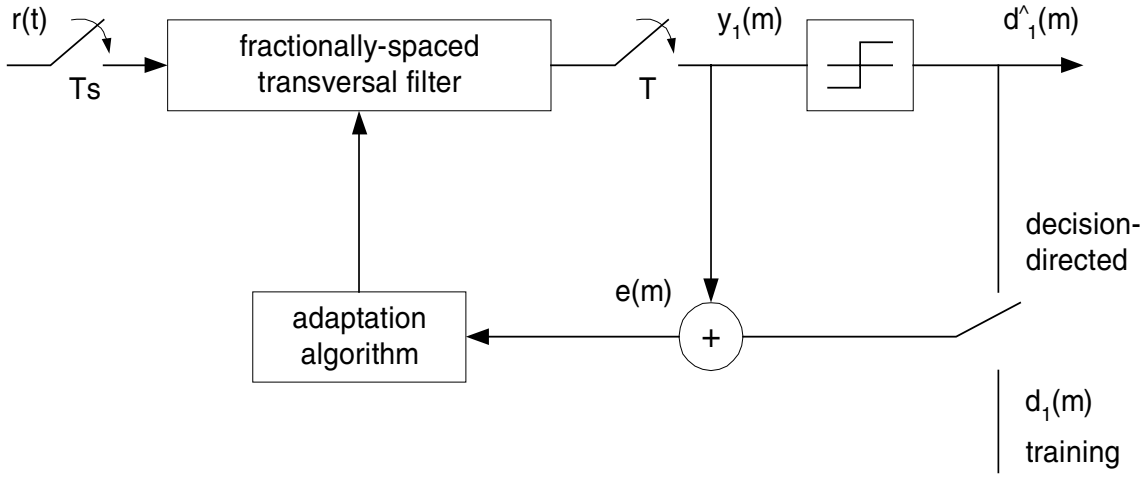


Figure 2.2: Complex-weight fractionally-spaced linear adaptive receiver (CW-FS-LAR).

At baseband, the received signal $r(t)$ is sampled every Ts seconds, where Ts relates to Tc, chip period, as $Ts = Tc/p$. In other words, the received signal is sampled p times per chip. The sampled signal is then fed to the FIR filter. This filter has to be at least Np taps, where N is the spreading gain. Thus, the filter has to at least be able to hold one full symbol. The filter output is sampled every symbol period, T, to obtain the estimate of the transmitted data, $y_1(m)$. The adaptation algorithm is chosen to minimize the mean square error (MSE) between the reference symbols and the estimated symbols.

Fractionally-spaced adaptive receivers for single-user detection have a similar structure to that of fractionally-spaced equalizers. Although equalizers can only reject intersymbol interference (ISI), this adaptive filter can also despread the desired user's signal, eliminate MAI, and combine multipaths. Fractionally-spaced adaptive receivers can exploit the spectral correlation of the signals in a DS/SS system. They can act as time-dependent adaptive filters, combining frequency-shifted and weighted versions of the frequency correlated components of the signal to obtain a better estimate of the desired signal.

There are several adaptation algorithms to choose from, the selection depending upon the applications. Trade-off includes performance, rate of convergence, and complexity.

2.6 Adaptive Receivers for Signals with Carrier Frequency Offsets

As stated earlier, the received signals have to be downconverted to baseband, and then resampled before entering the adaptive receiver. This implies that the carrier frequency is known. The original baseband signal is recovered with additive ambient and receiver noise and interference. The baseband signal is resampled and processed by digital filters.

Some signal detection techniques require not only knowledge of the carrier frequency, but also knowledge of the carrier phase. This kind of detection is known as phase-coherent detection. The receivers of this type have a carrier tracking mechanism and better performance than their noncoherent counterparts. Unfortunately, this type of receiver is not as practical because of its high cost.

Another type of detection is called noncoherent detection. This type of receiver does not require knowledge of the carrier phase. In systems that employ phase shift keying (PSK) signaling, there is yet another class of detection techniques known as differential coherent detection. This technique is useful when it is not possible to resolve phase ambiguities. In such a system, the information is encoded in the phase difference between the successive transmitted symbols. The transmitted information is recovered by detecting the phase difference between the successive received signals.

In mobile communications, where there is relative motion between transmitters and receivers, the Doppler effect causes a carrier frequency offset at the receivers. Thus, even if we have a perfect oscillator in both the transmitter and the receiver, there is still a frequency offset between the carrier frequency and the reference frequency at the receivers. This effect makes coherent detection techniques difficult because the constellation of the received signal keeps rotating. Thus, the receiver in mobile environments must be able to eliminate or compensate for this carrier offset. More details on these types of receivers can be found in [1].

2.7 The Differentially Coherent Adaptive Receiver

As stated earlier, a carrier frequency offset results in a rotating constellation of the detected symbols. The rate of rotation depends on the magnitude of the offset, and the direction is determined by the sign of the offset. With small frequency offset, the phase shift of the adjacent symbols is likewise small.

A differentially coherent detection technique can solve this rotation problem, provided that the frequency offset is small. The information is recovered by measuring the difference between successive detected symbols. Thus, as long as the frequency offset is small compared with the phase separation between successive transmitted signals, the differential detection will work. However, since the current decoded data is based on previous data, the accuracy of this detection technique is less than that of coherent detection techniques.

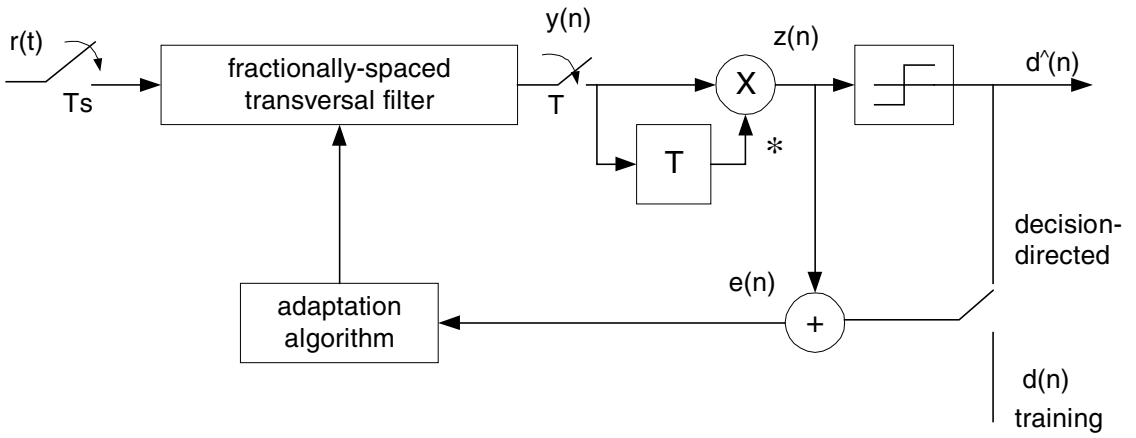


Figure 2.3: Adaptive receiver incorporating differential detection.

Figure 2.3 shows the structure of a differentially coherent adaptive receiver. Yoshida et al. [9] was the first to exploit this structure for fading channel applications. In this structure, the filter minimizes the difference between the differentially detected symbol and the reference signal, which may come from two sources—a known sequence (training mode) or the output of the decision device (decision-directed mode). With a constant carrier frequency offset, there is a fixed phase shift between successive symbols, which can be incorporated into the filter solution to eliminate any phase shift in the differentially detected symbols.

For this thesis, the selected adaptation algorithm is the Least Means Square Error (LMS) algorithm. This algorithm has an acceptable convergence rate [10] with small MSE after it converges [11]. This algorithm was chosen over the Normalized Least Means Square Error (NLMS). Although the NLMS algorithm converges faster than the LMS, it requires a division operation and its steady–state MSE is higher [11]. Bershad [12] also illustrated that for small step-sizes, the NLMS does not converge faster and both algorithms behave similarly. Interested readers can find more details of the LMS algorithms in [13]-[17] and [11, 12] for the NLMS algorithms. The LMS algorithm is summarized mathematically as follows:

$$y(n) = \mathbf{w}^{\mathbf{H}}(n)\mathbf{r}(n), \tag{2.16}$$

$$z(n) = y(n)y^*(n-1), \tag{2.17}$$

$$e(n) = d(n) - z(n), \tag{2.18}$$

where $\mathbf{w}(n)$ is the filter-weight vector at the n^{th} symbols, $\mathbf{r}(n)$ is the received samples of the n^{th} symbol, and $e(n)$ is the error between the desired signal and the filter output. The adaptation equation is,

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e^*(n)\mathbf{r}(n)y^*(n-1), \tag{2.19}$$

where μ is the step-size.

The LMS algorithm starts by calculating the filter output (Equation 2.16) when the received samples of a symbol align with the filter. Next the estimated received symbol is obtained (Equation 2.17) and subtracted from the reference signal (Equation 2.18). The difference is used to calculate the new filter coefficients (Equation 2.19). Due to the dependency of the later calculations on the earlier ones, the receiver requires real-time computations with low output latency. This makes the receiver very computationally intensive. Figure 2.4 depicts the algorithmic data flow of the LMS algorithm.

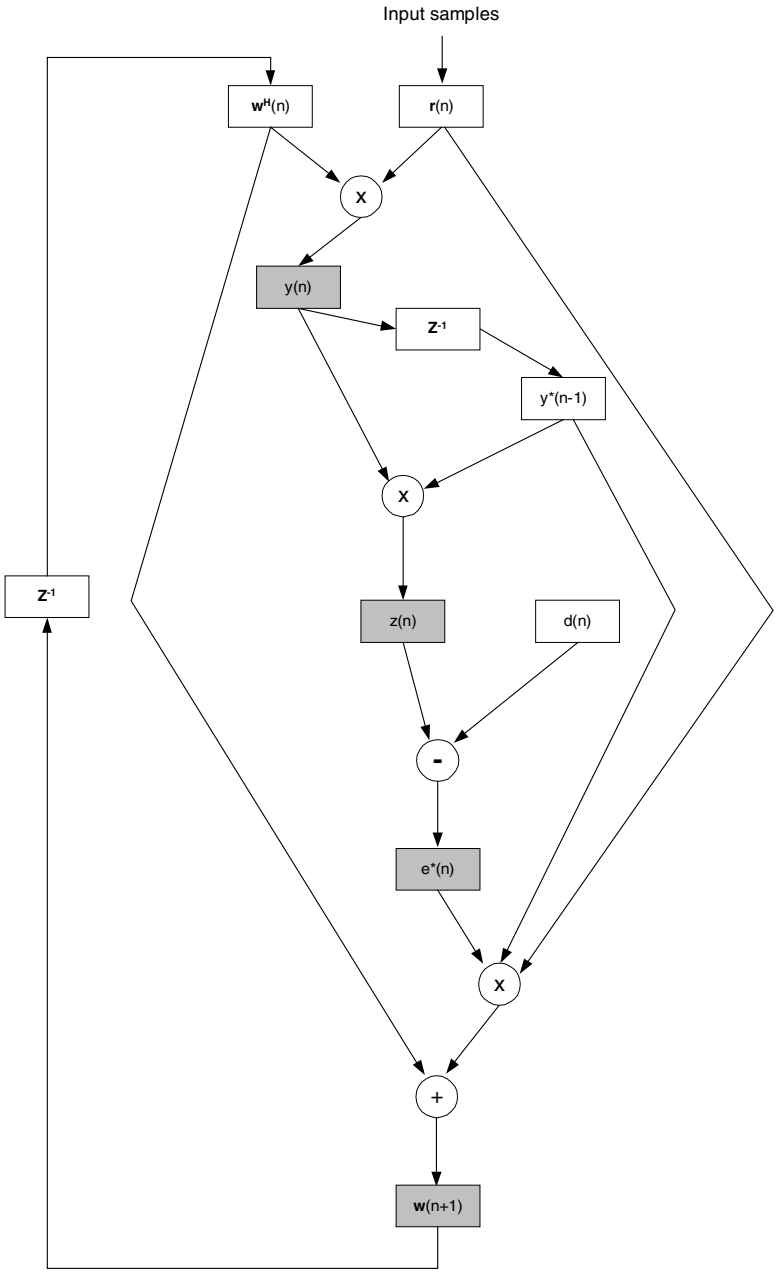


Figure 2.4: Algorithmic data flow of the LMS algorithm.

Chapter 3

Configurable Computing

This chapter begins with an overview of configurable computing machines (CCMs). It then introduces field programmable gate array (FPGA) devices as well as FPGA design approaches. Next, the chapter presents the CCM used in this thesis, the GigaOps G900. Finally, the chapter concludes with the stream-based architecture, which is implemented in this receiver.

3.1 Introduction to CCMs

The computational requirements of current digital signal processing tasks are significant. These tasks are conventionally accomplished through either special-purpose processors (DSPs) or hard-wired processors (ASICs) that implement single algorithms. However, there are certain applications that also require flexibility in addition to computational power, e.g., software radio applications. These applications implement different computationally complex algorithms (or parts thereof) at different times. Fixed-hardware processors are obviously not suitable for these tasks, while special-purpose processors may not be well matched in terms of performance. These applications require new types of computing machines that not only provide better computational power but also reconfigurability.

Dynamic reconfiguration to increase performance is not a new idea. In the 1970s, extensive research was conducted to enable alteration of a processor's instruction set. The purpose was to allow the designers to tailor the instruction set to specific applications

to improve performance. The limitation of this concept was that machines are reconfigurable only at the instruction level, while the data path was still static.

The second era of reconfigurable computing machines began with the availability of SRAM-based field programmable gate arrays (FPGAs). The new machines allowed designers to completely change processing architectures (data path and control logic) to suit the implemented applications by reconfiguring basic components. The basic building blocks of most FPGAs are flexible logic blocks with programmable interconnections among these blocks. The exact structures of these blocks vary across device families. However, they generally consist of several look-up tables (LUTs) and storage devices. The LUTs are used to implement logic functions instead of gates, as would be the case for ASICs. The inputs to the logic function are treated as addresses, and the function's outputs are the values stored in those addresses. Thus, by storing different values, different logic is implemented.

The first configurable computing machine was the Programmable Active Memories (PAM) [22]. It was constructed in 1988. The purpose was to assist general-purpose processors in performing computationally intensive algorithms or parts thereof. PAM consists of a 5x5 matrix of Xilinx XC3020 FPGAs, four 32-bit wide RAM banks, and a host processor interface circuit.

The next generation of configurable computing machines was SPLASH. The machine provides 16 Xilinx XC4010 FPGAs fully connected via a 16x16 crossbar switch. It can connect to a Sun SPARC station through a separate interface board. SPLASH can be extended to include up to 16 boards to give more configurable components for big designs. The advantages of SPLASH over its predecessors are results of its bigger FPGAs, better interconnections, and shorter configuration time.

Another interesting machine is PRISM-II, constructed at Brown University by Athanas, Agarwal, and Silverman, et al. [23]. The machine consists of three Xilinx XC4010 FPGAs for configurable computing devices and an AM29050 processor to provide an interface with the main processor. Unlike PAMs for which the designers have to manually determine and implement the hardware parts, the PRISM-II automatically synthesizes the hardware from C source code for the computationally demanding parts of

the algorithm in order to supplement the computational abilities of the general-purpose processor.

In this thesis the differentially coherent adaptive filter using the LMS algorithm is implemented on the FPGA-based configurable computing platform called GigaOps G900. An overview of this machine is given in Section 3.4. The entire receiver structure is divided into several sub-modules connected in series. The partitioning is done such that each block performs only one specific task at high speed and then sends the processed output to the next module.

To eliminate the need for a buffer, the rate of incoming information for a module has to be equal to or less than the rate of outgoing information. This makes the system only as fast as the slowest module. However, by having one module for each specific task, the throughput of the FPGA-based systems is still higher than conventional DSP-based systems for comparable clock speeds. Even if the algorithm can be divided among multiple DSP processors, the overhead and complexity of the partitioning can be overwhelming.

3.2 Field Programmable Gate Arrays

The receiver is partitioned and implemented with a number of programmable logic devices known as field programmable gate arrays (FPGAs). As the name suggests, these devices are capable of being reconfigured to implement any desired digital circuit. This is made possible by having a large number of small configurable logic blocks and a very flexible connection mechanism between these blocks.

Because the FPGA used is in the Xilinx family, in this thesis we will follow the terminology that Xilinx uses. Xilinx calls this small configurable device a "Configurable Logic Block (CLB)." For the XC4028EX-series Xilinx FPGAs, each CLB contains two 4-input lookup tables (LUTs) and a 3-input LUT. These three elements together can form up to an 8-input combinational logic. Along with them are two D flip-flops for data storage. Since every digital circuit can be implemented using combinational and sequential logic, CLBs are very useful basic elements. The simplified block diagram of a CLB is shown in Figure 3.2.

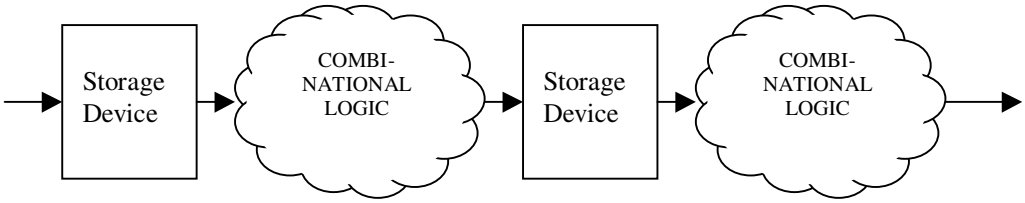


Figure 3.1: Basic building block of digital circuits.

To interface between the internal logic and the external package pins, the input/output blocks (IOBs) are used. They can be synchronous or asynchronous, and they can be input, output, tri-state output, or bi-directional pads. The structure of an IOB is shown in Figure 3.3.

Besides the direction, users are also allowed to program some features of the IOBs. For XC4000-series Xilinx FPGAs, the IOBs' thresholds can be set to TTL-compatible (1.2 V) or CMOS-compatible (5.0 V), to be able to interface with both TTL and CMOS logic. For the IOB programmed as a synchronous input block, the data signal arriving at the pad can be delayed by a delay device before being clocked into the IOB. This device is used to allow some clock delay from the clock pin to the clock input at the IOB. IOBs serve a critical function in high-speed implementation with more than one FPGA. To reduce the delay of the signals between two FPGAs, and thus increase the speed of the system, the IOBs should be set to synchronous mode. In the adaptive filter implementation, every IOB is synchronous.

Another interesting element is the high-fan-out, low-skew global buffer. For the XC4028EX FPGA, there are eight—two at each corner. These buffers can be used to buffer clock signals for synchronous logics with a guaranteed low skew rate. They also serve as general-purpose buffers for internal signals that have a large number of fan-outs.

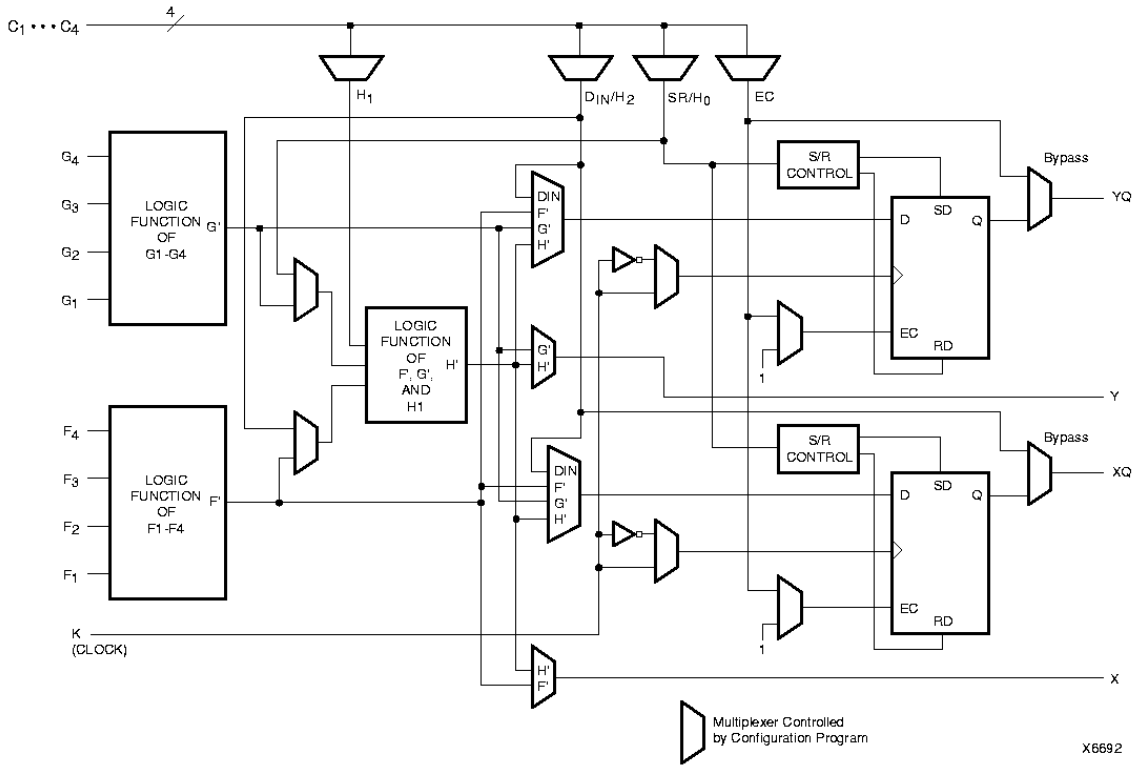


Figure 3.2: Simplified block diagram of XC4000-series configurable logic block.

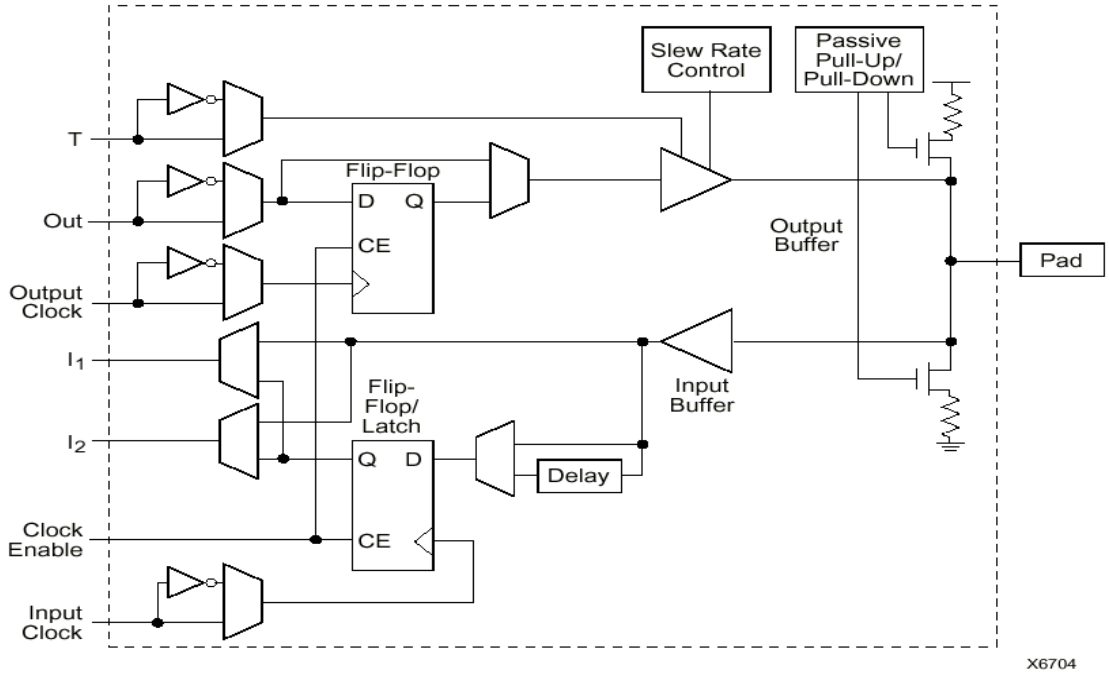


Figure 3.3: XC4000-series input/output block.

3.3 FPGA Design Process

In this thesis, the design is done in text-based Hardware Description Languages (HDL). The language of choice is VHDL. VHDL, with commercial tools, allows users to model designs, simulate the functionality of the designs, and then synthesize the designs.

There are two approaches in coding designs with VHDL. The first approach is behavioral-level modeling. In this approach, the design is coded in a high-level description, more like the C language. This provides a quick way to simulate the functionality of the design. Behavior-level modeling is not suitable for efficient synthesis because it does not provide a way to specify the actual components in the design.

The second approach is structural-level modeling. This approach allows users to precisely specify components used in the design and the connections between them. This approach is useful in synthesis but can produce a long simulation time. Generally, designers model systems using the behavioral style first. Then, after functionality is verified, the system is divided into smaller modules and written in the structural style.

Another advantage of structural-level modeling is the reusability of the commonly used devices, such as counters, multiplexers, or adders. Utilities such as Xilinx LogiBLOX are capable of generating technology-specific components, which guarantee high speed and minimum silicon area when placed on the specific FPGAs. By using these components, the speed of the system is greatly increased and its resource requirements are reduced.

3.4 GigaOps G900 Configurable Computing Platform

The configurable computing platform used is a PCI-based GigaOps G900. The platform can hold up to 32 field programmable gate arrays, with a versatile interconnection between them. Figure 3.4 shows a schematic diagram of the board.

The G900 board can generally be divided into two parts—the platform bed and the configurable modules. The platform bed provides communication between the host application and the configurable devices. It is also responsible for distributing the clocks and the connection between the computing modules. On the platform, there are four

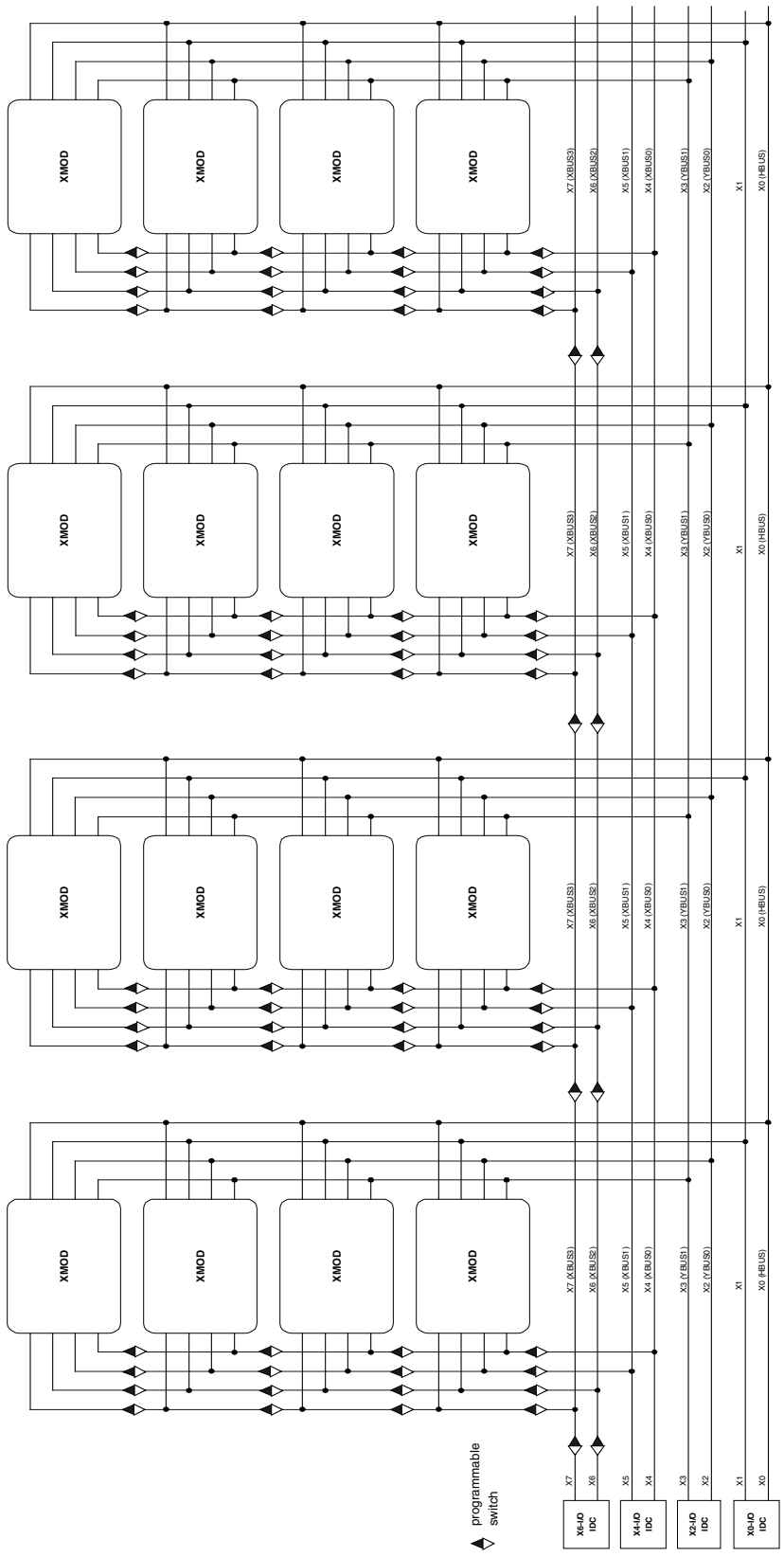


Figure 3.4: G900 board block diagram.

sockets on which configurable modules can be placed. These modules are known as XMODs. They are the building blocks of the receiver.

A G900 platform bed allows up to four XMODs to be stacked on each socket. This makes it capable of supporting up to sixteen XMODs. XMODs can contain various types of computing and memory devices. The XMODs used in this thesis each contain two Xilinx XC4028EX-3HQ4028 FPGAs, 8 MB of DRAM, and 256 kB of SRAM. A block diagram of an XMOD is given in Figure 3.5.

As noted earlier, the communication between XMODs is provided by the platform bed. There are six primary 16-bit busses and two 8-bit busses [18]. All of these are available to all XMODs. The two 8-bit secondary busses, called X0 (HBUS0) and X1 (HBUS1), are primarily used to implement the communication between the G900 platform and the host application and are not designed to be used for general interconnection between XMODs. However, they can be used when the associated timing is managed. The remaining six 16-bit busses, designated as X2, X3, X4, X5, X6, and X7, can be used for general interconnection in the design.

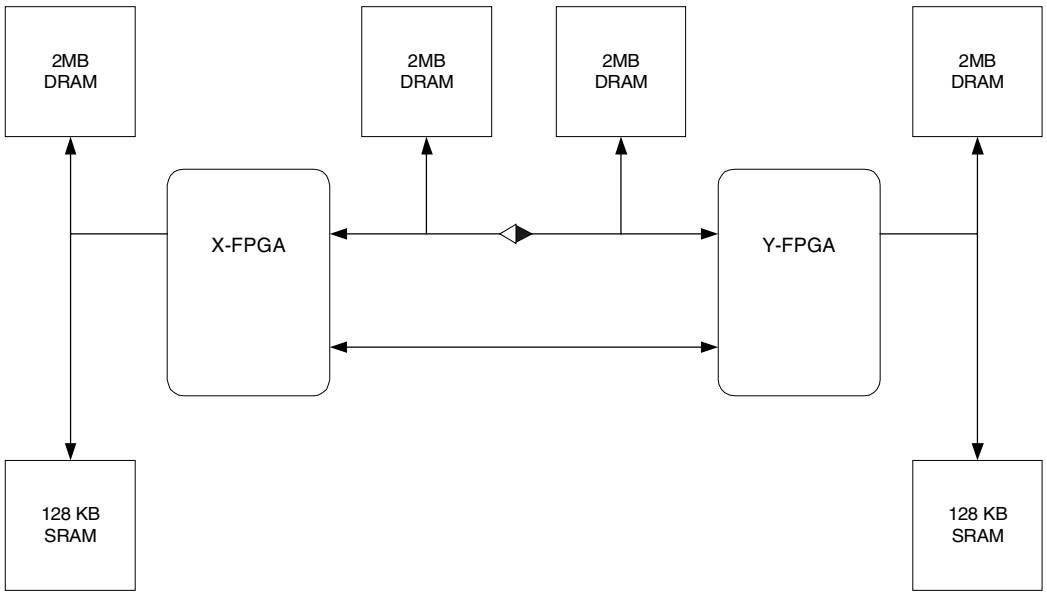


Figure 3.5: XMOD block diagram.

To add more flexibility, four of the primary busses are programmable. By turning the switches on and off, the bus can be separated into several shorter busses. This provides the reusability of the same bus among different XMODs. Moreover, an FPGA on each XMOD can communicate with the other FPGA on the same XMOD via auxiliary busses, which can be up to 40 bits wide.

Even though these six primary and two secondary busses are available to all XMODs, a particular FPGA on the XMOD cannot access all of the busses. The first FPGA (X-FPGA) can access only busses X4, X5, X6, and X7, while the second FPGA (Y-FPGA) can access busses X0, X1, X2, and X3.

3.5 Stream-based Architecture

In this thesis, the stream-based concept is employed. The stream-based architecture [2], which provides a degree of flexibility, is useful in environments that allow parallel operations with only one data flow path. In this environment, the operations that can be performed in parallel are separated and implemented on the different modules. Each module then performs only the designated task at high speed and passes the processed output to the next module in a manner similar to an assembly line.

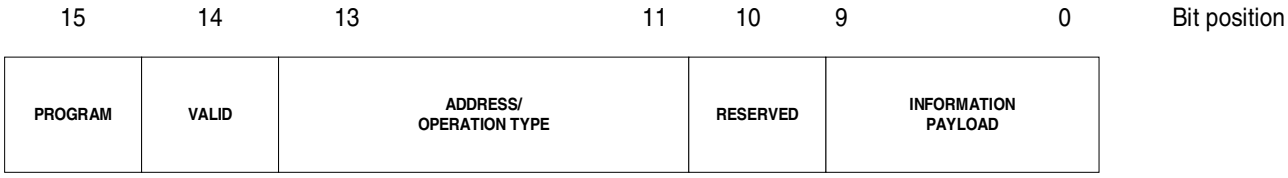
In this architecture, a stream of information flows through the system, starting with the first module and finishing at the last module. While it propagates through the modules, it is processed depending on its type. Generally, there are two types of information in the stream—data information and configuration information. The data information contains data that the modules process, while the configuration information carries the necessary information for changing the parameters of the system. In each data word, which is 16 bits wide, a bit called Program is set aside to indicate the type of the information contained in it—either data information or configuration information. Another bit called Valid is used to indicate the validity of the transferred information, allowing the modules to transfer information with the adjacent modules at an arbitrary rate.

In Chapter 2, Equations 2.16-2.19 describe each distinct operation of the adaptive receiver algorithm. Because of the dependency of each of these operations on the output

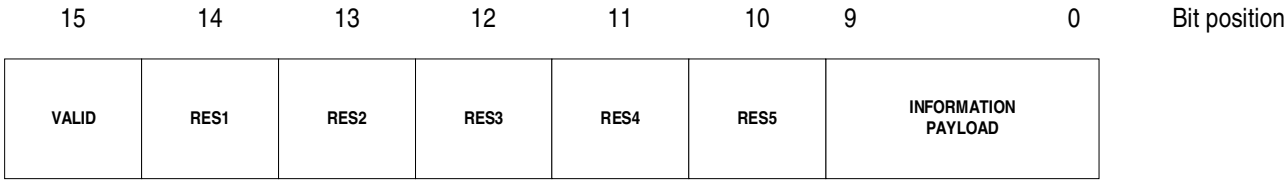
of the prior operations, and similarities among the operations, they are implemented in the same module. For such a module that implements more than one operation, three more bits in the stream word are used to indicate which operation should be performed on the incoming data. This leaves up to eight possible operations to be implemented on the same module. The same field is also used as an address field for the configuration information stream to indicate the module to which the information belongs. Figure 3.6 (a) shows the layout of a word flowing forward in the system.

For the radio implementation, the incoming data rate is slower than the maximum possible running speed of the FPGA devices. By having the data clock, which triggers the data to transfer between two modules, run at a higher speed than is actually needed, there is the continual presence of invalid information. These periods of invalid information can be used to carry information backward from latter stages to prior stages. In this implementation, the data clock runs eight times faster than the incoming data rate. Thus, for every eight clock cycles, only two clocks are used to transfer data, one for in-phase (I) data and the other for quadrature (Q) data, while the others are unused. The in-phase (I) and quadrature (Q) data are explained in Chapter 4. These unused slots are used to carry information backward. This approach is similar to time-division multiplexing. Figure 3.6 (b) shows the layout of the word flowing backward, while Figure 3.7 illustrates the direction of information flow in an 8-clock period. An advantage of this approach is that fewer busses are needed to carry information.

From Figure 3.7, the first two time slots are used to carry data forward. The next two slots are reserved and can be used for applications requiring more than 10-bit data precision or having more than two data to transmit. The next slot is unused. For this slot, both modules produce a tri-state value. This is to prevent both modules from driving the bus at the same time, which may permanently damage the FPGAs. The next two slots are used for backward information. The last slot is unused and the output from both modules has a high impedance. We refer to a group of these eight slots as one packet.



(a)



(b)

Figure 3.6: Format of information word. a) forward information. b) backward information.

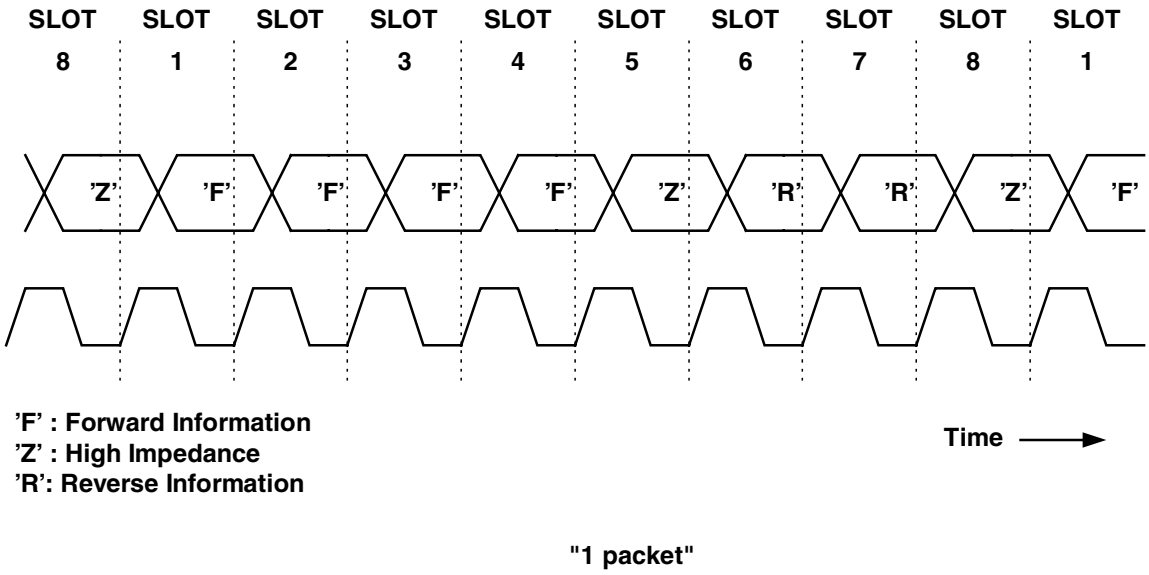


Figure 3.7: Time division multiplex of information flow between two adjacent modules.

With this stream-based approach, flexibility is achieved—i.e., changeable step-size, and filter length—and the module synchronization requirements are decreased, while the system is also allowed to have feedback in the data path. Also, the modules can be developed and tested individually, reducing the design, implementation, and debugging time.

3.5.1 Module Structure

Every module has the same basic structure in this stream-based modular architecture. Figure 3.8 illustrates this structure. It is comprised of a state machine, a Process Pipeline, a Program Pipeline, a Bypass Pipeline, and a Backward Bypass Pipeline.

The state machine is responsible for controlling the flow of information to the appropriate pipelines, depending upon the type of information. It also selects the appropriate pipeline output to pass on to the next module. In some modules the state machine also selects the type of operation required based on the incoming data. The inputs to the state machine are Valid, Program, and Address/Operation Type.

The Process Pipeline implements the operation assigned to the module. The Program Pipeline stores the configuration data, which in turn changes the functionality of the Process Pipeline. The Bypass Pipeline provides the path for the configuration information that does not belong to the module to flow through.

The Backward Bypass Pipeline provides the path for the backward information to pass from the latter module to the previous module without changing the content of the information.

Even though each module's state machine is different from the others, they all have four common states, shown in Figure 3.9. As noted before there are eight clock cycles between consecutive data, where the first four cycles are for forward data and other two are for reverse data with a high-impedance slot between the forward and reverse data. The information types are the same for all the time slots in the same packet. Thus, it is sufficient for the state machine to test only the word in the first time slot of a packet.

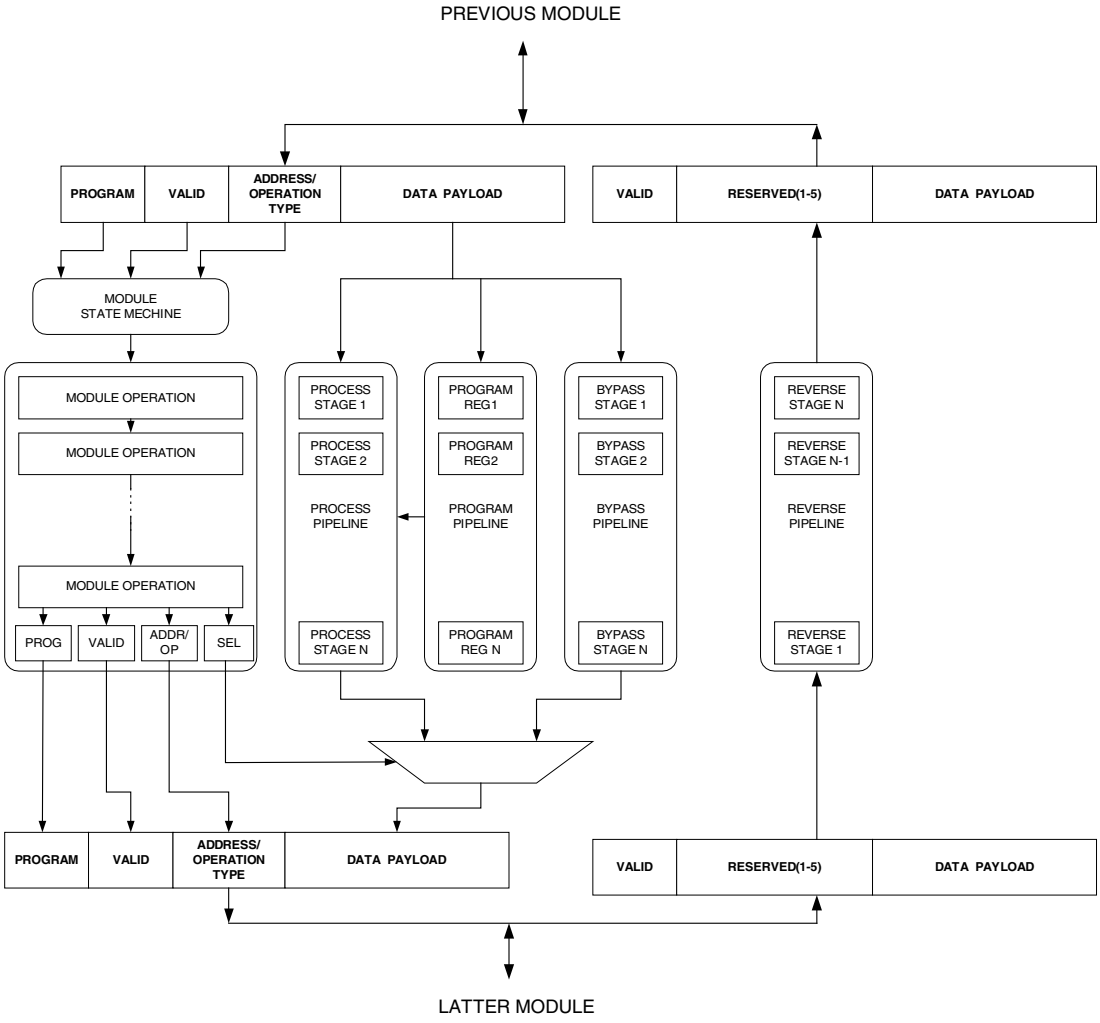


Figure 3.8: Block diagram of modules in the implementation.

The state machine changes to the appropriate state, discussed below, at the first slot, and then transfers back to the IDLE state at the eighth slot to prepare for the new packet. When Valid is negated, the state is unchanged and stays at the IDLE state. At this state no pipeline is enabled and the output of the module is invalid. When Valid is asserted, however, the state machine changes to one of three possible states: PROGRAM, BYPASS, or DATA. If the Program Bit is negated, the state is changed to the DATA state and the incoming data is manipulated with the implemented operation of that module.

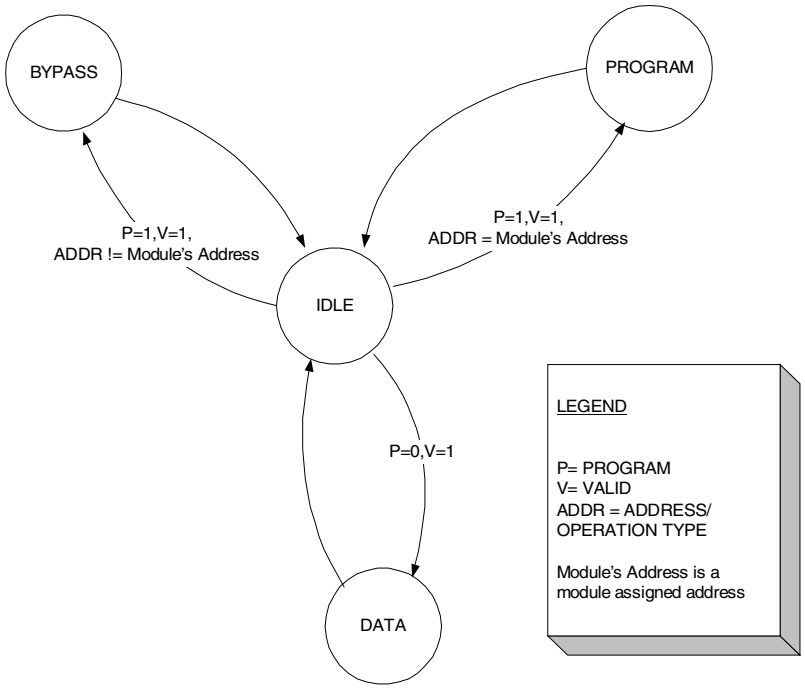


Figure 3.9: State diagram.

If the Program Bit is asserted and the configuration information does not belong to the module, then the state machine changes to the BYPASS state; otherwise it changes to the PROGRAM state. This is made possible by looking at the ADDRESS/Operation Type field of the incoming word. At the PROGRAM state, the Configuration Pipeline retrieves the information and stores it, which in turn changes the functionality of the module.

3.5.2 Module Addressing

The configurable parameters are different for each module. Thus, to be able to set these parameters correctly, each module is assigned with a distinguishable address. When the system is first started, every module has the same address, 0. Then, to assign the address, the first module, which acts as a control module, generates the configuration packets with the ADDRESS/Operation Type field set to 0 and the PAYLOAD field set to the desired

address. Upon receiving this packet, the next module compares the ADDRESS field with its own current address, 0. And because they are the same, the module takes the value in the PAYLOAD as a new address. The address mechanism is enabled only for the first time that the module receives configuration information.

To assign an address to the next module, the same mechanism is employed. The control module generates a configuration packet with the address set to 0 and PAYLOAD set to the desired address. When this packet is passed on to the second module, because of the mismatch between the module's address and the value in the ADDRESS field, the packet bypasses the current module and continues on to the next module. At the next module, which has the current address set to 0, the value in the PAYLOAD is used as a new module's address.

Chapter 4

FPGA-based Single-User Receiver Implementation

This chapter starts with a description of the operational aspects of the receiver. Next, an overview of the receiver testbed is presented. Finally, the implementation aspects of the receiver, including the mapping of the receiver structure to the stream-based concept and the utilization of the G900 board, are explained in detail.

4.1 Receiver at a glance

The prototype receiver is designed to support multiple data rates, multiple spreading gains, and a variable number of samples per chip. The available data rates and the corresponding spreading gains are shown in Table 4.1. Because of the limited number of XMODs available, the implemented prototype system supports only a data rate of 31.25 Kbits/sec., with two samples per chip and a spreading code of length 16. This results in a chip rate of 500 KHz and sampling rate of 1 MHz.

User's data rate (KHz)	Sample per chip (p)	Spreading Gain (N)
31.25	2	16, 24, 32
	4	8, 12, 16
62.5	2	16
	4	8

Table 4.1: The available data rate and corresponding spreading gain of the system.

The PN sequences used are Gold codes of length 15 extended to 16. The Gold codes offer better cross-correlation properties with little sacrifice of auto-correlation properties compared with the maximum-length sequence.

As shown in Chapter 2, the coefficient update is performed to reduce the difference between the reference signal and the estimated data. There are two sources of reference data. The first is a known sequence. This sequence is a special string of bits which is known to both the transmitter and the receiver. At the transmitter, this sequence is interleaved with the user data. Taking this special sequence as a reference source, the receiver updates the weights accordingly. This mode of operation is called training mode. There is no user data sent in this mode.

The other mode is called decision-directed mode. In this mode, the transmitter sends out user data. At the receiver, this data is estimated and used as a reference source. This mode is enabled only when the receiver is known to have a low bit-error-rate (BER).

The prototype system starts with it first loads the receiver with the desired user's spreading code and disables the coefficient update mechanism. During this time the receiver acts as a matched filter receiver, i.e., filter coefficients are equal to the spreading code of the desired user. After the user of interest is acquired, the coefficient update process is turned on. Because the BER of the matched filter output is low enough to use as a reference signal, the system only works in the decision-directed mode. This coefficient update mechanism is turned on as long as the user of interest is still acquired.

4.2 Receiver Testbed

The receiver testbed consists of an RF front end, a Harris 50214 digital downconverter, the PCI-based GigaOps G900 computing platform, and a host PC. Figure 4.1 shows the diagram of the system.

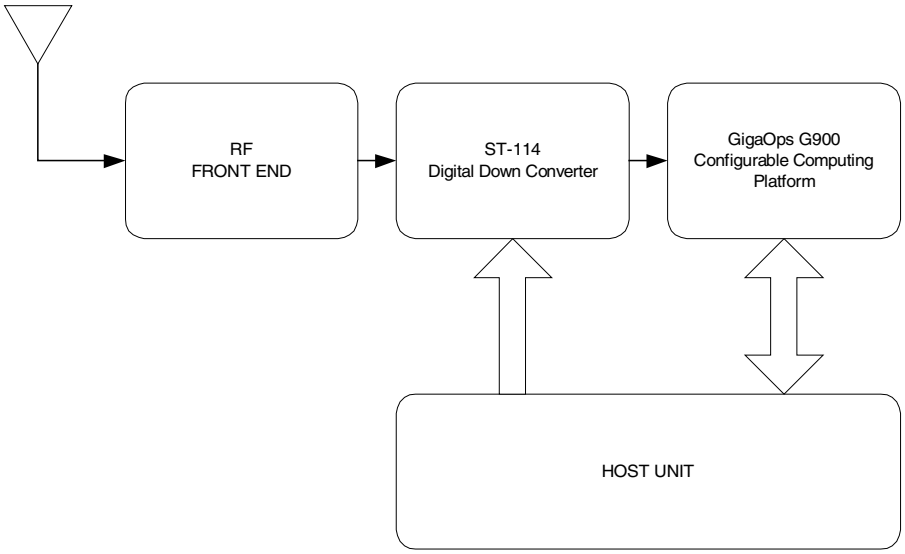


Figure 4.1: The system block diagram.

4.2.1 Receiver Front End

The RF front end includes the RF filter and amplifier, mixer and local oscillator, and IF filter and amplifier. The incoming signal from the antenna, at 2050 MHz, is mixed down to an IF frequency below 100 MHz, which is then passed to the digital downconverter board.

4.2.2 Harris 50214 Digital Downconverter (DDC)

The Harris DDC receives the IF signal from the front end and converts it to a digital baseband signal. The Harris 50214 Programmable Digital Downconverter chip is available through the Sigtek ST-114 Evaluation Board [19]. The IF signal, from the RF front-end part, is sub-sampled at 8 MHz by an on-board 10-bit analog to digital converter (ADC). The signal is then mixed down to baseband and decimated to 1 MHz using the Cascaded Integrator-Comb (CIC) decimation filter and an FIR filter. This sequence of operation results in two signals, in-phase (*I*-channel) and quadrature (*Q*-channel) data, both of which are 16 bits wide. However, only the top ten bits are passed to the receiver.

4.2.3 GigaOps G900 Configurable Computing Platform

The G900 configurable computing platform provides some flexibility. Users can select the type and number of computing modules as well as program the connections between them. The platform also provides the host-platform communication protocol which tightly couples the host program with the FPGA-based computing modules. The 10-bit wide *I* and *Q* data are passed to the platform via ribbon cable through busses X6 and X7.

4.3 Receiver Implementation

Figure 4.2 illustrates the floor plan of the prototype receiver. The design employs the stream-based concept described in Chapter 3. This results in four modules connected in series. The modules include (a) the Input module, (b) the Adaptive Filter & Decision module, (c) the Acquisition & Tracking module, and (d) the Output module.

The Input module functions as a system controller. The primary functions of the module are: 1) buffering the incoming in-phase and quadrature data and 2) forming a stream and sending different stream types at the appropriate time. The module is also capable of communicating with the host program to receive a new configuration, which in turn changes the receiver's parameters.

As discussed in Chapter 2, the equations comprising the adaptive algorithm have to be performed in a certain order. By sending out different data types at the appropriate time, the Input module can choose a particular operation of the Adaptive Filter & Decision module. This timing information is generated by the Acquisition & Tracking module and sent backward to the Input module via the available backward time slots, as described in Chapter 3.

The Adaptive Filter & Decision module implements the differentially coherent adaptive filter receiver described in Section 2.7. It consists of a variable-length finite impulse response (FIR) filter, a coefficient update unit, and a hard-decision device. The dependency of one operation on previous operations and their computational requirement similarity (multiplication and addition) are the reasons these three operations are implemented in the same module.

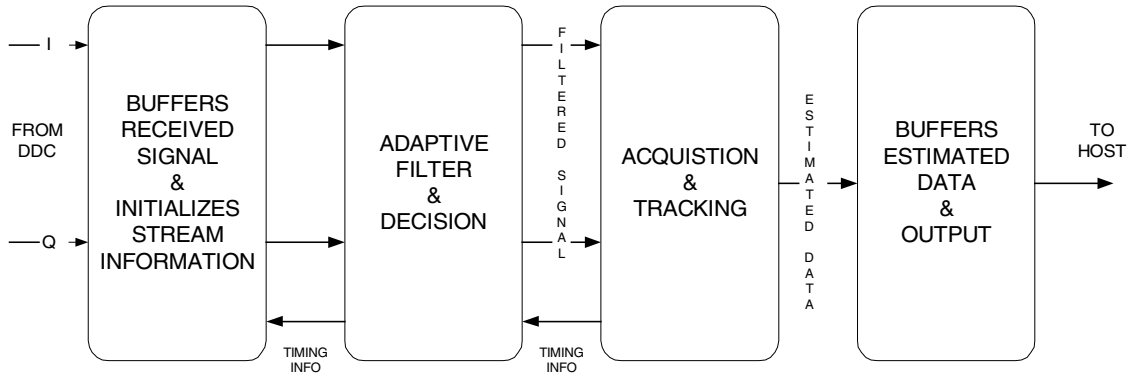


Figure 4.2: Diagram of stream-based single-user receiver.

The Acquisition & Tracking module performs two functions simultaneously. The Acquisition portion is responsible for determining if the user of interest is sending any data. By searching for a repetitive correlate peak location, whose approximate magnitude is above a certain threshold, the existence of any user can be verified. After the desired user is acquired, the tracking mechanism begins. Because of clock jittering effects, the correlate peak location gradually moves with time. The Tracking portion monitors this change, updates this information, and sends it back to the Input module.

The Output module is the last module of the system. It is responsible for collecting the estimated data and transferring this data to the host program upon request.

4.3.1 Input Module

The Input module, as the name suggests, is the first module of the system. It is responsible for two functions. The first is to buffer the incoming I and Q data and incorporate this data into the data stream for transport to the next module, the Adaptive Filter & Decision module. The second function is to communicate with the host program so that new system configurations can be sent into the system.

As shown in Chapter 2, the operations of the adaptive receiver (Equations 2.16-2.19) have to be executed in a certain order. As discussed earlier, due to the dependency of later operations on the output of the prior operations, these operations are implemented

in a single module: the Adaptive Filter & Decision module. This reduces the amount of data transferred between modules. From Equations 2.16-2.19, the filter output at the peak location must first be calculated. Next this output is multiplied with the complex conjugate of the output at the previous peak. This product is then subtracted from the reference signal. Finally, the difference is used to update the filter coefficients. Thus, the Input module must send out a different type of data stream, by using a different Address/Operation Type, to the Adaptive Filter & Decision module at the appropriate time. The Acquisition & Tracking module provides this timing information by sending timing measurement back to the Input module. Table 4.2 shows the valid value of the Operation Type field and its meaning.

Operation	Operation Type field
FIR (Equation 2.16)	"0XX"
Initialize Weight	"101"
Update Error (Equation 2.17, 2.18)	"110"
Update Weight (Equation 2.19)	"111"

Table 4.2: Value in Operation Type field and its meaning.

Even though for demodulation purposes, it is sufficient that the adaptive filter calculates the output only at the peak location—a position that is reached when all samples of a symbol are in the filter taps—for acquisition and tracking, the filter must produce an output every time it receives a new sample. Thus, after the desired user is acquired, the Input module must first generate Np FIR type streams, where N is the spreading gain and p is the number of samples per chip. Following these streams is an Update Error packet. This packet causes the Adaptive Filter & Decision module to calculate the error between the estimated output and the reference signal. An Update Weight packet is then sent out to let the Adaptive Filter & Decision module update its coefficients. This cycle of operations exists until the user of interest is no longer acquired. Figure 4.3 shows the forward stream at the Input module's output. Note that

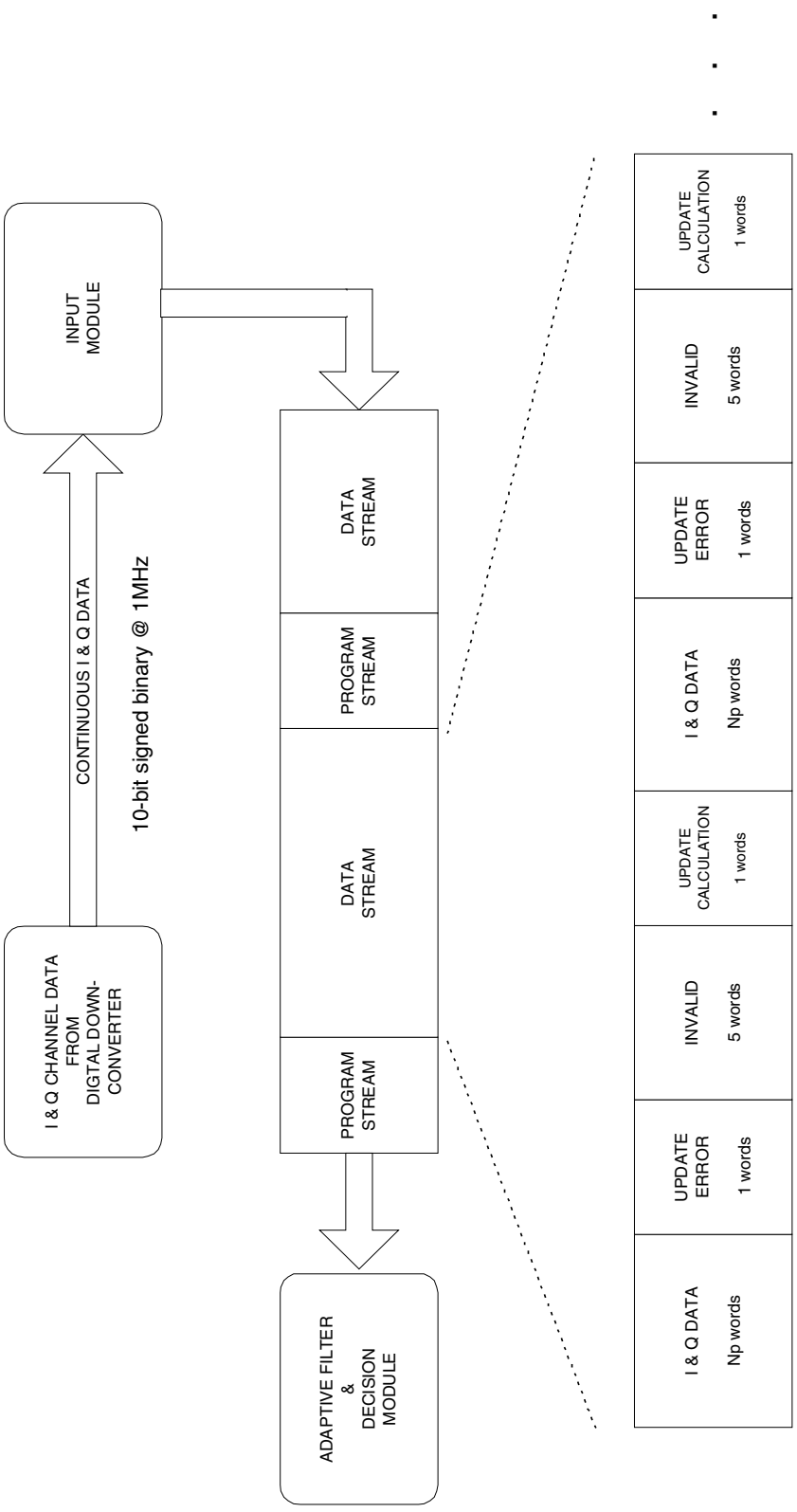


Figure 4.3: Organization of the stream output of the Input module.

for simplicity's sake, only the forward messages are shown in the figure, while the reverse slots are still available for the feedback information.

There is one more operation type implemented on the Adaptive Filter & Decision module: the Initialize Weight operation. This operation is not required for an adaptive receiver; however, it is necessary for the receiver to operate as a matched filter. As explained in Section 4.1, the receiver first operates as a matched filter. Initially the Input module sends a data information packet with the Operation field indicating the Initialize Weight operation. The Input module then sends the data information packets with the Operation field indicating the FIR filter operation. By initializing the adaptive filter with the desired user's spreading code using an Initialize Weight packet and sending out only FIR-type data packets, the receiver effectively works as a matched filter. This mode continues until the user of interest is acquired. Only then is the weight update process enabled; the receiver subsequently operates as an adaptive filter.

4.3.2 Adaptive Filter & Decision Module

The Adaptive Filter & Decision module implements the differentially coherent adaptive filter described in Chapter 2. The adaptive algorithm is the Least Mean Square (LMS) algorithm. As noted in the previous section, the module performs different operations depending on the type of incoming data. The first operation is a complex-FIR filter described by Equation 2.16. The output of this operation is marked valid and sent via forward slots to the next module, the Acquisition & Tracking module.

At the peak location, the module calculates the error of the filter output, using Equations 2.17 and 2.18, and then updates its complex weights using Equation 2.19. These are done when the module receives Update Error and Update Weight packets. There is no valid output from these operations. After the coefficients are updated, the operation switches back to the FIR filter operation again.

There are two reasons why these operations are implemented on the same module. The first is the similarity between the FIR and the Update Weight operation. The FIR filter operation is,

$$y(n) = \mathbf{w}^{\mathbf{H}}(n)\mathbf{r}(n),$$

where $\mathbf{w}(n)$ is the filter-weight vector and $\mathbf{r}(n)$ is the received-sample vector at time nT, where n is an integer number and T is the symbol period. The operation requires complex number multiplications and additions.

On the other hand, the Update Weight operation is,

$$\mathbf{w}(n+\mathbf{1}) = \mathbf{w}(n) + \mu e^{*}(n)\mathbf{r}(n)y^{*}(n-1),$$

where μ is the step-size, $e(n)$ is an error at the n^{th} symbol, $\mathbf{r}(n)$ is the received signal vector, and $y(n\text{-}1)$ is the output of the filter at the peak location of the $(n\text{-}1)^{th}$ symbol. This operation also requires complex-number multiplications and additions.

The second reason why these operations are implemented on the same module is the dependency of one operation on the others. The Update Weight operation requires knowledge of the filter output as well as the error signal. On the other hand, the FIR operation needs the newly updated coefficients, which are calculated with the Update Weight operation. Therefore, to minimize the data transfer between the modules and the resource requirements, these three operations are grouped together.

As explained later in this chapter, the Adaptive Filter & Decision module is implemented on three XMODs because this module uses multiplication extensively. The multiplication process demands many computational resources (CLBs) and a single XMOD on the target platform does not provide a sufficient number of these resources. Figure 4.4 shows the mapping between the receiver structure described in Chapter 2 and the implementation on three XMODs.

Each complex number multiplication requires four multipliers and two adders. A 10-bit by 14-bit multiplier generated by the Xilinx LogiBLOX utility requires 143 CLBs. Thus, it could take as many as 32x4x143, or 18304, CLBs just for multiplication for a 32-tap FIR filter. With the XC4028 Xilinx FPGAs used in this thesis, that could add up to more than 17 FPGAs or eight XMODs. These multipliers can, however, be used by the different operations to reduce the amount of required resources. This results in a more efficient but also more complicated design. The filter part is implemented on two XMODs, where each FPGA implements a smaller 8-tap filter. There are four 10-bit by

14-bit multipliers on each FPGA. By having the processing pipeline running at eight times the incoming data rate, these four multipliers can be used to calculate a complex multiplication for all eight taps, one tap per clock cycle.

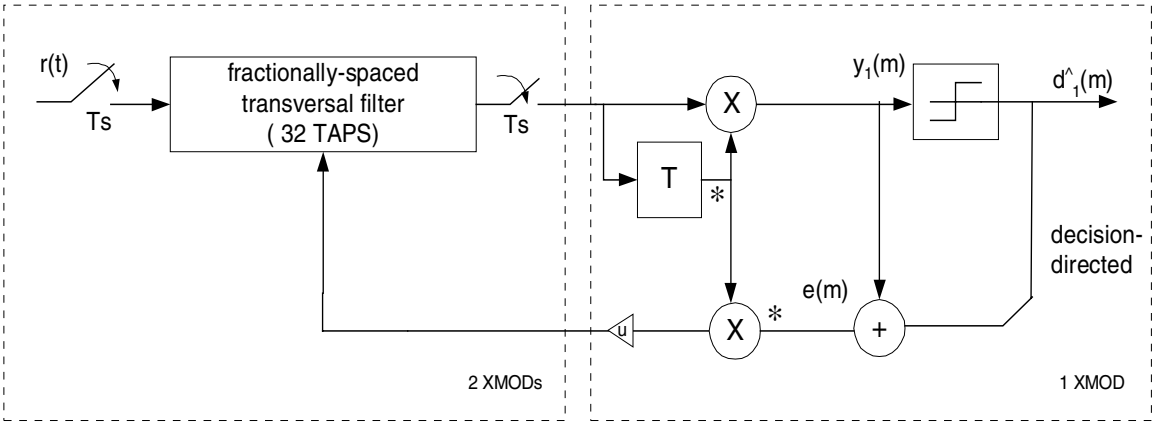


Figure 4.4: Implementation of the Adaptive Filter & Decision module.

By adopting this scheme, the same multipliers can also be used for the weight update part. From Equation 2.19, the weight update operation is similar to the filter operation. The weights are changed by an amount equal to a complex number $\mu e^*(n)y^*(n\text{-}1)$, which we call *scaled error value* in this thesis, multiplied by the corresponding received samples. By using the third XMOD to calculate this scaled error value, the same multipliers used for the filter part on the first two XMODs can also be used for the weight update part. By allowing one input of the multipliers to come from either the weights or the scaled error value, while the other inputs come from the receive samples, the same multipliers can be used for both operations. Figure 4.5 illustrates the simplified structure of the circuit implemented on the first two XMODs. An advantage of separating the filter and weight update part from the scaled error value calculation part is that to change the weight update algorithm, the designers would need to change only the structure of the third XMOD. By having a different scaled error value calculation, the design effectively implements a different adaptive algorithm.

4.3.3 Acquisition & Tracking Module

The Acquisition & Tracking module implements two mechanisms: the acquisition mechanism and tracking mechanism. The acquisition mechanism is responsible for determining the presence of the desired signal. This module takes as input the correlated output between the received samples and the desired user's spreading code. This output, from the filter operation, should show the persistent peaks corresponding to the user's symbol. By tuning the automatic gain control (AGC) in the digital downconverter (DDC) board, these peaks have magnitude above a certain value. The acquisition process searches for such peaks, and when the number of the repetitive peaks exceeds a set value, the user is deemed acquired.

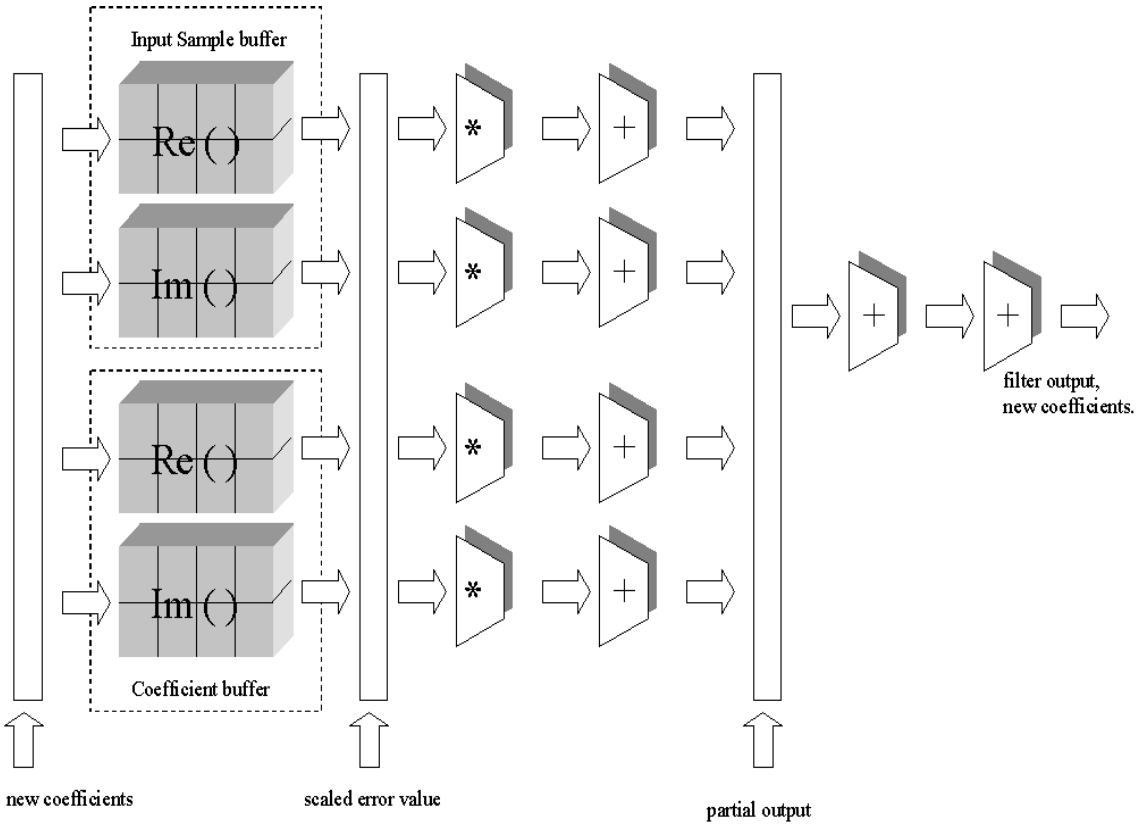


Figure 4.5: Block diagram of the Adaptive Filter & Decision module (filter part).

Upon acquisition the operation switches to tracking. Because of the mismatch between the clock at the digital to analog converter (DAC) of the transmitter and the

other clock at the analog to digital converter (ADC) inside the DDC board, the user symbol is gradually shifted with time. The tracking mechanism monitors this shift. By using an early-late tracking method, the module can find if the peak is shifted, and if so, whether it is shifted to the left or to the right. This peak information is crucial for the system operation because the adaptive filter must update its weights at the right time.

After the peak location has been determined, the module sends this information back to the Input module using the available backward slots. The Input module then incorporates this information into its functionality so that it sends packets with appropriate operations at the appropriate time, which in turn allows the Adaptive Filter & Decision module to perform the weight update at the right time. After that the Acquisition & Tracking module updates the Input module every symbol period with a new peak location if the peak has shifted. The tracking mechanism is enabled as long as the desired user is still acquired.

4.3.4 Output Module

The Output module collects the demodulated data of the user of interest and stores it in a first-in, first-out (FIFO) register until they are retrieved by the host program. Figure 4.6 shows the structure of this module. It consists of a serial-to-parallel converter, one 256x16 bit FIFO, a Status register and an FIFO Data register. A FIFO of this size allows a maximum of 4096 bits (0.1311 sec. at a data rate of 31.25 KHz) worth of data to be stored. This allows the host program plenty of time to retrieve the data before the FIFO overflows.

A Status register maintains the status of the user: empty, full, or overflow. This register is mapped to a variable in the host program. When the host program read this variable, the value of the Status register is loaded into the variable, allowing the host program to determine the status of user's data and act accordingly.

The FIFO Data register stores the 16 oldest decoded symbols. It is also mapped to a variable in the host program. When the data is read, the value of this register is transferred to the host program, and the next 16 symbols are loaded into the register. Figure 4.7 illustrates these two FIFO registers.

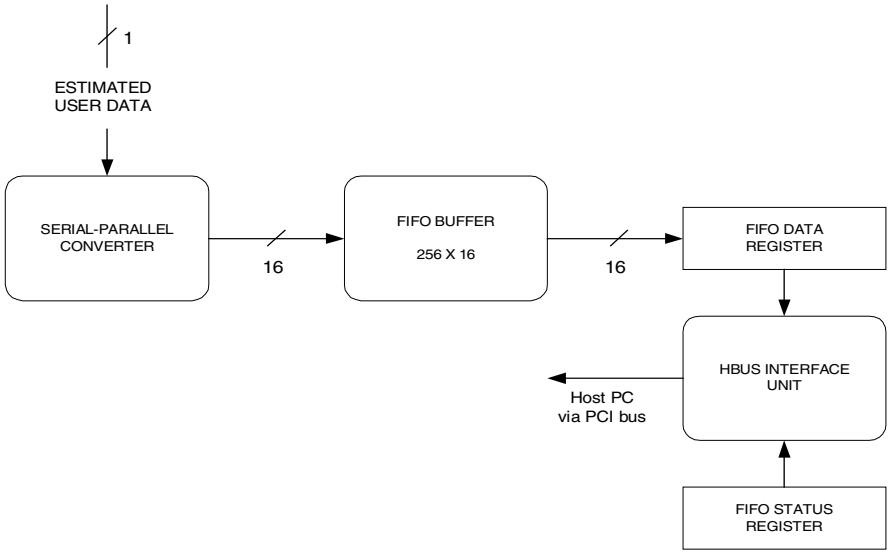


Figure 4.6: Output module structure.

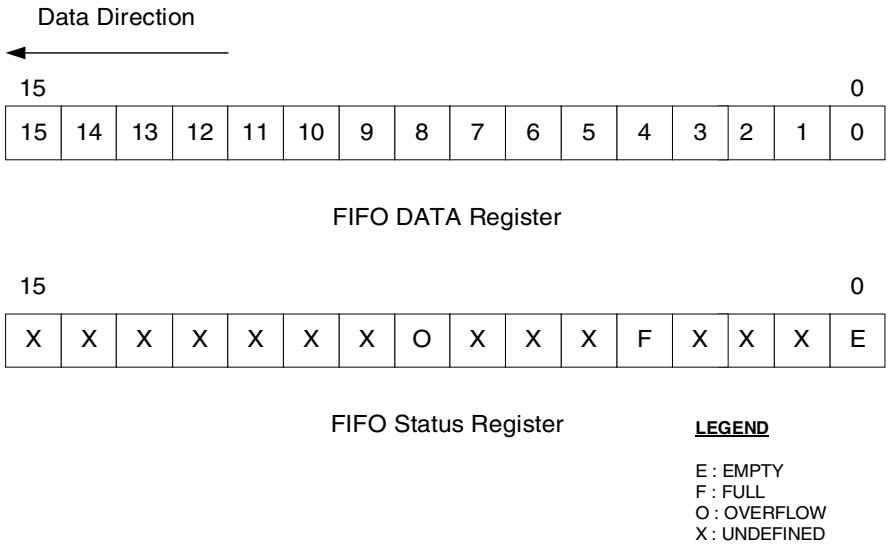


Figure 4.7: FIFO registers. Top: FIFO data register. Bottom: FIFO status register.

4.4 Mapping of the receiver structure onto the platform

This section presents the mapping of the receiver structure on to the GigaOps G900 configurable computing platform. The previous section describes the operations that must be implemented on each module. This section, as a complement, details how these functions are mapped onto the target configurable board.

There are six XMODs used for this system. Four of them are stacked on the first socket, and the other two are on the second. The diagram of the receiver on the GigaOps G900 board is presented in Figure 4.8.

The XMODs on the first stack include a) the Input XMOD, b) the Adaptive Filter Decision Part XMOD, c) the Acquisition & Tracking XMOD, and d) the Output XMOD. The second stack, holding two XMODs, implements the filter part of the Adaptive Filter & Decision module. They are the Adaptive Filter Main XMOD and the Adaptive Filter Auxiliary XMOD.

These XMODs are named according to the module they implement, except for the following three XMODs. The Adaptive Filter Decision Part XMOD implements the second portion of the Adaptive Filter & Decision module shown in Figure 4.4. As noted earlier, the prototype receiver implements a 32-tap adaptive filter. The first 16 taps of the receiver are implemented on the Adaptive Filter Main XMOD, while the other 16 taps are implemented on the Adaptive Filter Auxiliary XMOD. Note that in Figure 4.8, the dashed XMODs indicate the possible implementation if such XMODs are available.

4.4.1 Input XMOD

The Input XMOD is placed on the bottom of the first stack. The XMOD implements the Input module described above. 16-bit busses X6 and X7 are used to carry the in-phase and quadrature signals from the DDC board through the X4 external port to the XMOD. Only the top 10 bits are utilized; the bottom 6 bits are discarded due to the limited resources of the latter module.

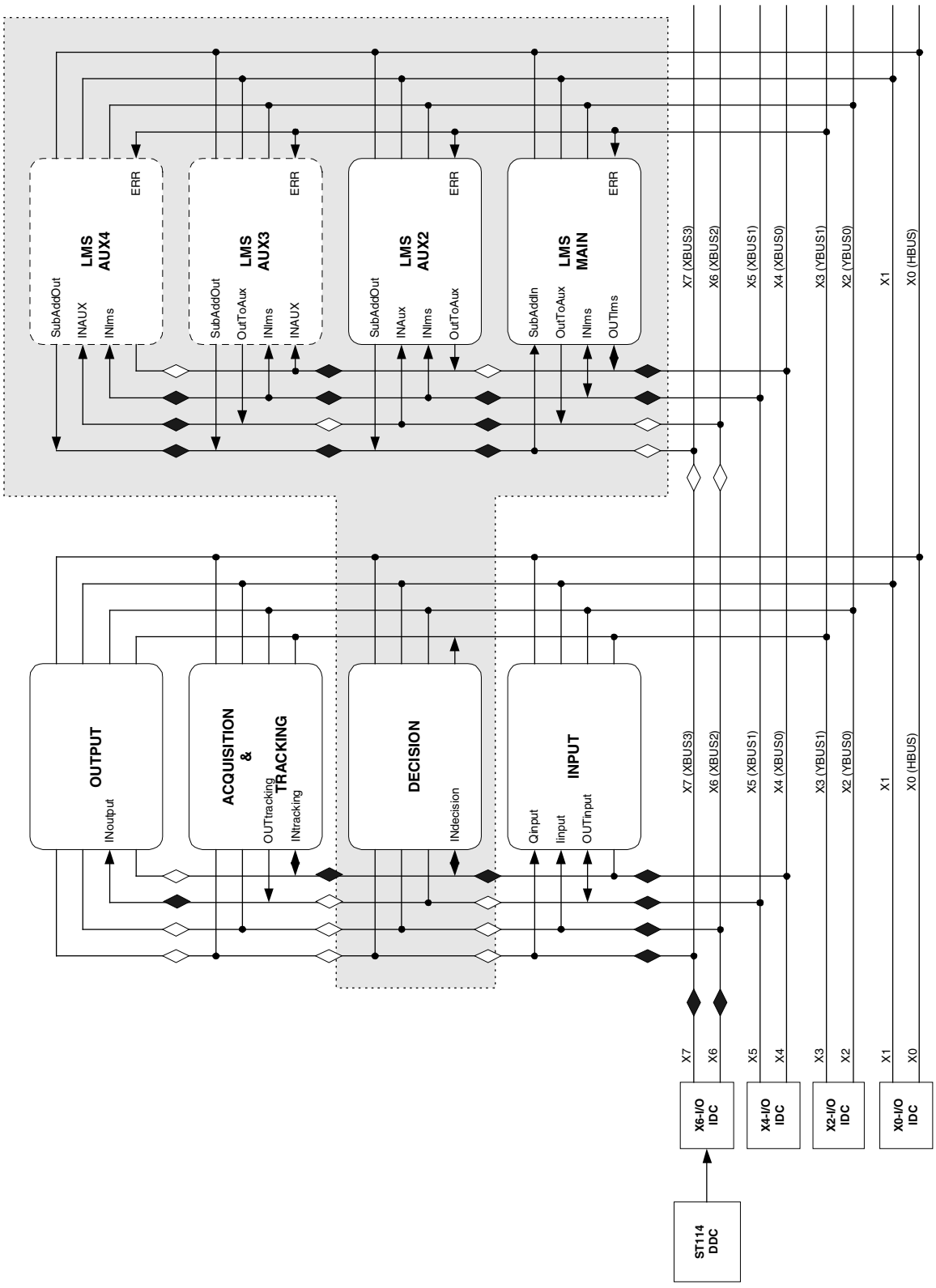


Figure 4.8: Implemented receiver on the GigaOps G900 configurable platform.

As described in Chapter 3, there are two 128 KB-SRAMs and four 2 MB-DRAMs on each XMOD. With the input data rate of 1 MHz from the DDC board and the Input module's output data rate around 1.25 MHz, it is sufficient to use the SRAMs to buffer the data. By using the SRAMs, the module is allowed to stop sending output for 65.54 milliseconds without losing any data. Furthermore, using SRAMs also simplifies the memory interface circuit. Both SRAMs are used: one for buffering the in-phase data and the other for the quadrature data. As shown in Chapter 3, these two SRAMs are separated and each one is available to only a particular FPGA. Thus, the *Q* data, which also enters the XMOD at the X-FPGA, is routed to the Y-FPGA to be stored there, while the *I* data is buffered on the X-FPGA. Because the SRAM used is a single-port SRAM, to be able to both write and read the SRAM there are two clocks associated with the SRAM interface circuit. One clock is running at twice the rate of the other. The higher-speed clock (CLK2) activates the operation, while the lower-speed one (CLK1) controls the type of operation, i.e., whether to read or write. The CLK1 clock runs at two times faster than the incoming data rate. With the edge detection circuit, the availability of the new data is known. This new data is written into the SRAM when the CLK1 is high as well as CLK2. The data, on the other hand, is read back when the CLK1 is low and the CLK2 is high. The read and write operations as well as the associated clocks are shown in Figure 4.9.

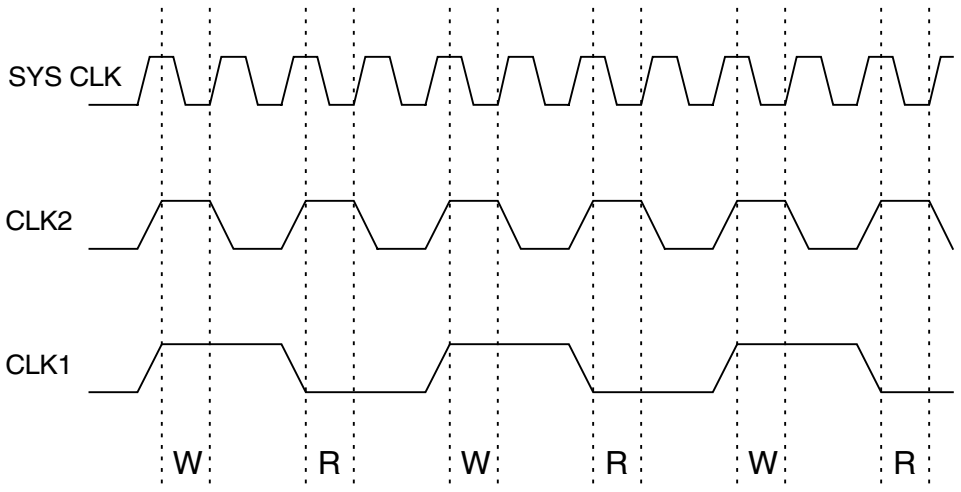


Figure 4.9: Read and write operation and associated clocks.

The block diagram of the Input XMOD is shown in Figure 4.10. The *I* data is buffered on the X-FPGA, while the *Q* data is buffered on the Y-FPGA. Registers are placed where data enters or leaves the chip. Since the *Q* data must be routed to the Y-FPGA, there are two extra registers placed on this signal path: one before it leaves the X-FPGA and the other when it enters the Y-FPGA. A 2-clock delay element, therefore, is placed on the *I* data path to produce the same amount of delay.

Besides storing the *Q* data, the Y-FPGA also communicates with the host program via HBUS protocol. The Y-FPGA receives new configuration information from the host program, then sends this information to the Input processing unit on the X-FPGA, which in turn forms the corresponding configuration information stream and sends it to the next module.

The utilization of both FPGAs is listed in the Table 4.3. Clearly, the CLB utilization is less than half of the chip's CLB count due to the simplicity of the Input module.

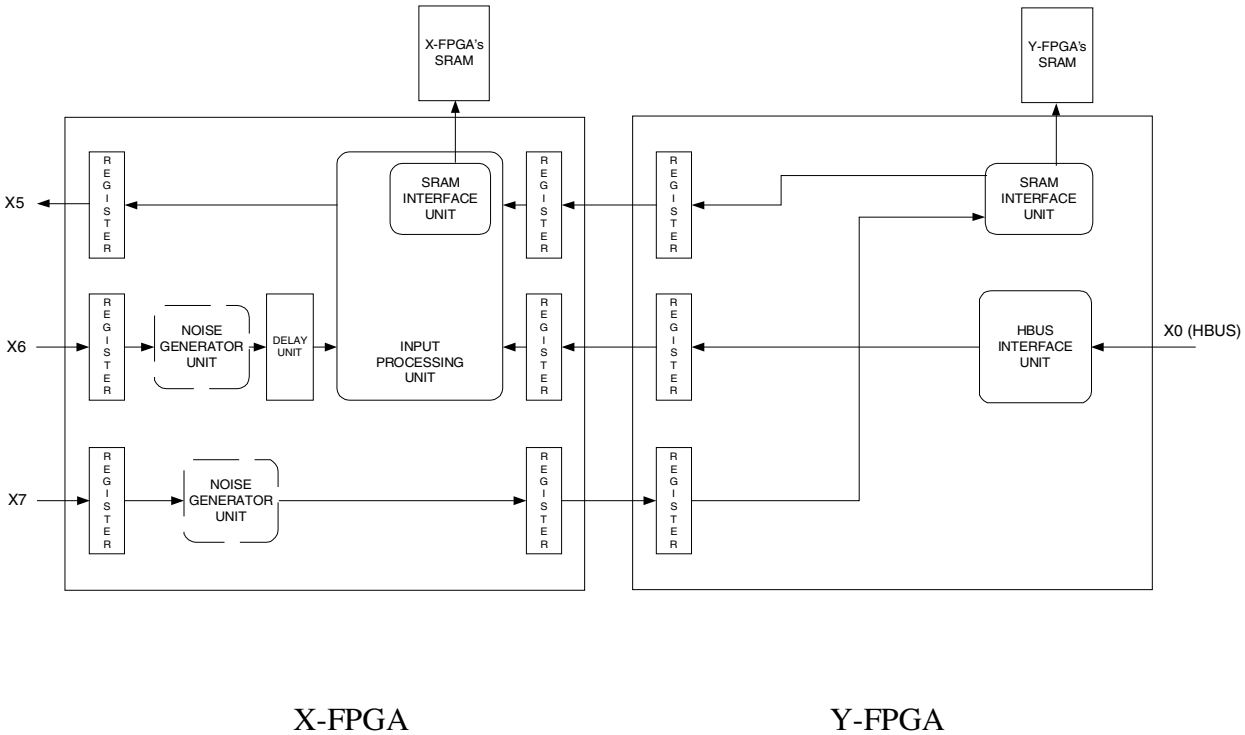


Figure 4.10: Diagram of Input XMOD.

	X-FPGA	Y-FPGA
Number of CLBs	262 of 1024 (25%)	163 of 1024 (15%)
CLB flip flops	180 of 2048 (9%)	237 of 2048 (125)
Number of IOBs	103 of 160 (64%)	98 of 160 (61%)
IOB flip flops	89 of 160 (56%)	70 of 160 (44%)
Equivalent gate count	18162	3298
Maximum clock speed	29.707 MHz	29.362 MHz

Table 4.3: Utilization of X- and Y-FPGA for the Input XMOD.

4.4.2 Adaptive Filter & Decision XMODs

The differentially coherent adaptive filter is implemented on three XMODs, called the Adaptive Filter Main XMOD, the Adaptive Filter Auxiliary XMOD, and the Adaptive Filter Decision Part XMOD. The Adaptive Filter Main and the Adaptive Filter Auxiliary XMOD together implement the FIR filter, while the Adaptive Filter Decision Part XMOD implements the decision device and scaled error value calculating unit.

The Adaptive Filter Main and the Adaptive Filter Auxiliary XMODs are mainly responsible for: 1) calculating the filter output and 2) updating the filter coefficients based on the scaled error value given by the Adaptive filter Decision Part XMOD. The entire filter structure, consisting of 32 taps, is divided into four smaller 8-tap filters and implemented on four FPGAs. Thus, each XMOD implements a 16-tap FIR filter. At each XMOD, the X-FPGA functions as the beginning part and the Y-FPGA implements the end of the 16-tap FIR filter.

The new I and Q data, multiplexed on the forward slots, arrive at the Adaptive Filter Main XMOD to the X-FPGA. This data is passed to the 8-tap FIR filter implemented inside of the X-FPGA and the eighth sample currently stored in the filter is passed on to the next 8-tap FIR filter implemented on the Y-FPGA. On the Y-FPGA, the similar 8-tap FIR filter is implemented. The partial filter output is fed back to the X-FPGA to sum with the rest of the filter output to obtain the complete output before being sent to the next module, the Acquisition & Tracking module.

Along with the partial filter output, the Y-FPGA also sends the input sample stored at the eighth tap of the filter back to the X-FPGA, so that it can be passed on to the Adaptive Filter Auxiliary XMOD. This Adaptive Filter Auxiliary XMOD receives the *I* and *Q* data at X-FPGA from the Adaptive Filter Main XMOD. At X-FPGA, the data is processed to produce the partial filter output, while the oldest received sample stored in the filter taps is passed to the Y-FPGA. The partial filter output at Y-FPGA is passed back to be summed at the X-FPGA, which in turn feeds it back to the Adaptive Filter Main XMOD to sum with the other part of the filter output to produce to the complete filter output. Figures 4.11 and 4.12 show the diagram of these two XMODs.

The utilization of the FPGAs on these two XMODs is almost at 100%. These resources are mostly used for the multiplication units.

	X-FPGA	Y-FPGA
Number of CLBs	1024 of 1024 (100%)	1002 of 1024 (97%)
CLB flip flops	1532 of 2048 (75%)	1396 of 2048 (68%)
Number of IOBs	98 of 160 (61%)	65 of 160 (40%)
IOB flip flops	119 of 320 (37%)	50 of 320 (16%)
Equivalent gate count	43314	40971
Maximum clock speed	37.874 MHz	30.498 MHz

Table 4.4: Utilization of the Adaptive Filter Main XMOD.

	X-FPGA	Y-FPGA
Number of CLBs	1024 of 1024 (100%)	1007 of 1024 (98%)
CLB flip flops	1452 of 2048 (71%)	1396 of 2048 (68%)
Number of IOBs	95 of 160 (59%)	64 of 160 (40%)
IOB flip flops	68 of 320 (21%)	48 of 320 (15%)
Equivalent gate count	41841	40985
Maximum clock speed	29.337 MHz	30.362 MHz

Table 4.5: Utilization of the Adaptive Filter Auxiliary XMOD.

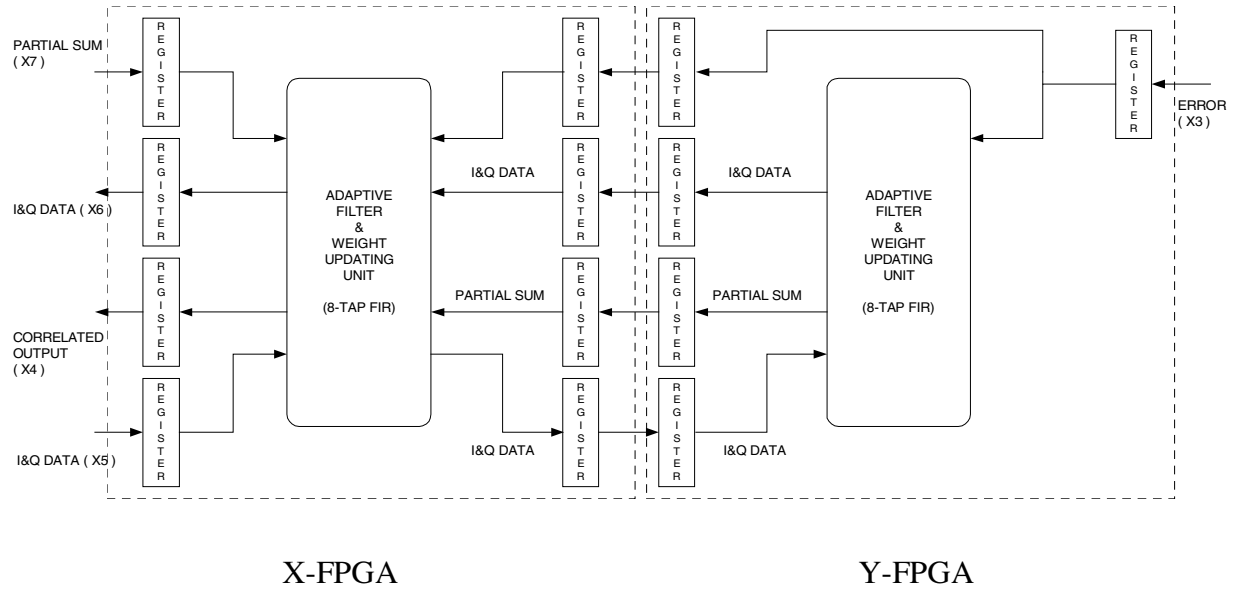


Figure 4.11: Adaptive Filter Main XMOD.

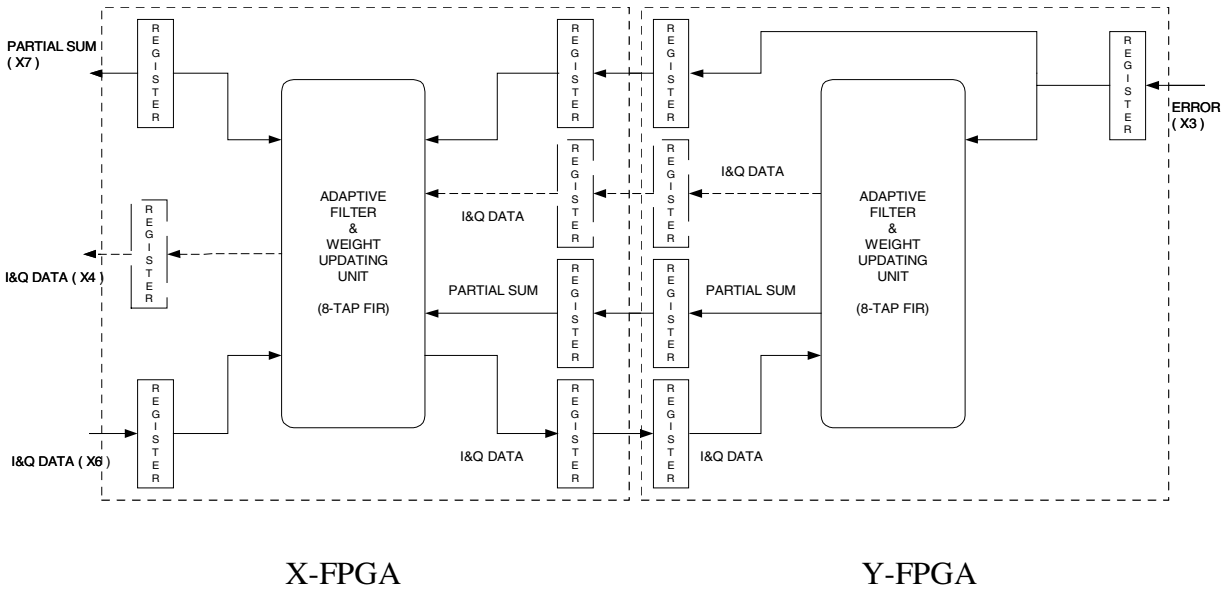


Figure 4.12: Adaptive Filter Auxiliary XMOD.

The Adaptive Filter Decision Part XMOD implements the second part of the adaptive filter. This part is responsible for finding the estimated symbol and calculating a part of the weight update equation, which is referred to as the scaled error value in this

thesis. The Adaptive Filter Decision part XMOD receives the correlated output from the Adaptive Filter Main XMOD via bus X4. Only the correlated output at the previous and current peaks are kept inside the module. Upon receiving an Update Error packet, which is originally generated by the Input module and bypassed through the filter module, the module calculates the scaled error value according to the equations below:

$$z(n) = y(n)y^*(n-1),$$

$$e(n) = d\hat{}(n) - z(n),$$

$$ue^*(n)y^*(n-1).$$

The resulting value is then passed to the Adaptive Filter Main XMOD and the Adaptive Filter Auxiliary XMOD via bus X3. The diagram of the Adaptive Filter Decision Part XMOD is shown in Figure 4.13. The utilization of the XMOD is listed in the Table 4.6.

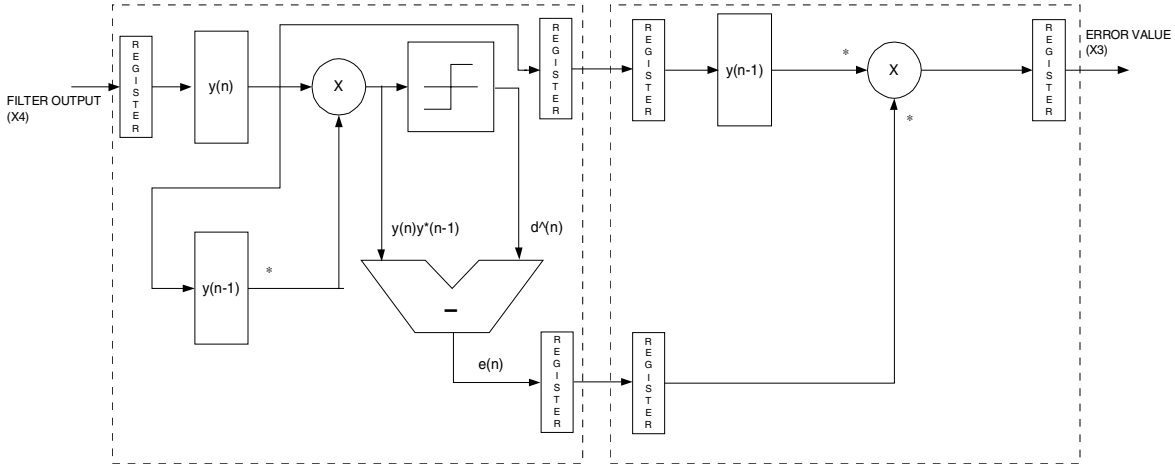


Figure 4.13: Block diagram of Adaptive Filter Decision Part XMOD.

	X-FPGA	Y-FPGA
Number of CLBs	1024 of 1024 (100%)	1002 of 1024 (98%)
CLB flip flops	1452 of 2048 (71%)	1396 of 2048 (68%)
Number of IOBs	95 of 160 (59%)	64 of 160 (40%)
IOB flip flops	68 of 320 (21%)	48 of 320 (15%)
Equivalent gate count	41841	40985
Maximum clock speed	29.337 MHz	30.362 MHz

Table 4.6: Utilization of Adaptive Filter Decision Part XMOD.

Note that the reason for implementing the Adaptive Filter & Decision module on three XMODs is because of the limited size of the FPGAs. This module is, however, conceived of as one module. Therefore, from the conceptual point of view, the stream-based concept is not violated.

4.4.3 Acquisition & Tracking XMOD

The Acquisition & Tracking XMOD implements the Acquisition & Tracking module. The correlated filter output enters the XMOD through bus X4. Due to the simplicity of the acquisition and tracking mechanism currently used, only the X-FPGA is utilized. The output of the module, the peak location information, is fed back to the Input module in the reverse information slots via bus X4. Figure 4.14 illustrates the module diagram and the utilization of the X-FPGA is listed in Table 4.7.

4.4.4 Output XMOD

The Output module is implemented on this XMOD. The X-FPGA receives the user's estimated data from bus X5 and bypasses this data to the Y-FPGA, where the FIFO output buffer is implemented. At the Y-FPGA, the valid data is put in the FIFO. By reading the FIFO Data register, the 16 oldest symbols in the FIFO are stripped off and loaded into the register. The Y-FPGA also holds the Host interface unit, which provides

a data transfer mechanism from the FIFO Data register and FIFO Status register to the host program.

This Host interface unit, unlike the other units, is designed using the X language [18]. The X language is a hardware description language developed and used by Giga Operations. The reason for using the X language, rather than VHDL, is because the GigaOps G900 board uses the HBUS protocol for communications between the Y-FPGA and the host PC. The simplest way to implement this protocol is to design the interface unit in the X language. The block diagram of Output XMOD is shown in Figure 4.15. Table 4.8 shows the utilization of the Y-FGPA.

	X-FPGA
Number of CLBs	243 of 1024 (23%)
CLB flip flops	183 of 2048 (9%)
Number of IOBs	43 of 160 (26%)
IOB flip flops	25 of 320 (8%)
Equivalent gate count	4868
Maximum clock speed	18.933 MHz

Table 4.7: Utilization of the Acquisition & Tracking XMOD.

	Y-FPGA
Number of CLBs	433 of 1024 (42%)
CLB flip flops	126 of 2048 (6%)
Number of IOBs	41 of 160 (25%)
IOB flip flops	44 of 320 (14%)
Equivalent gate count	36457
Maximum clock speed	19.550 MHz

Table 4.8: Utilization of Y-FPGA on the Output XMOD.

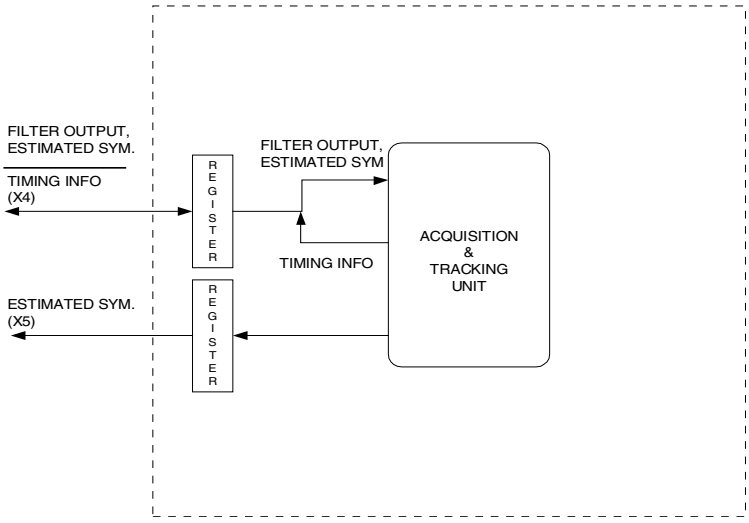


Figure 4.14: Acquisition & Tracking XMOD.

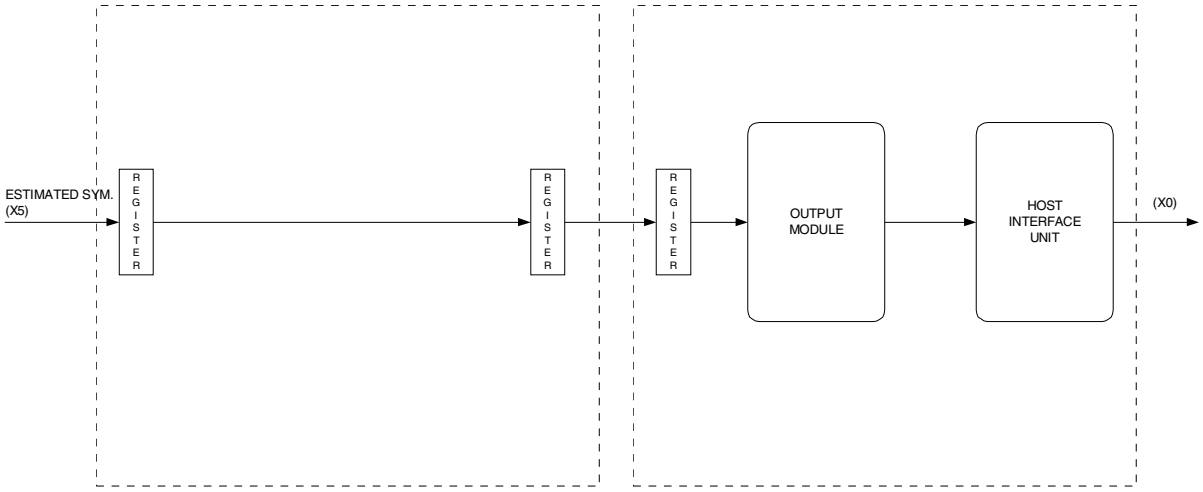


Figure 4.15: Output XMOD block diagram.

Chapter 5

Test Results

This chapter presents the strategy used to verify the functionality and performance of the receiver implemented, as well as the test results. It starts with the description of the test environment, which is followed by a description of the method used to simulate an AWGN channel. This gives us insight into how the receiver performs in the presence of white noise. The chapter is concluded with a comparison of the results from different receiver parameters.

5.1 Digital Testbed

The receiver is tested at the baseband to evaluate its performance with different capacities and noise levels. A digital transmitter is employed to generate the desired user's signal at the baseband, as well as the other users' signals to produce multiple access interference. The signals are fed to the digital downconverter board to create in-phase and quadrature data. This allows us to simulate the effect of carrier frequency offset on the performance of the receiver.

At the Input module, before buffering, digitally-generated white noise is added to the I and Q data. With this technique, a noisy environment is simulated. As explained later, receivers in this digitally generated noise environment may produce overly optimistic system performance as compared with those in a true noise environment due to the limited data precision used in the system.

5.1.1 Baseband Multiuser Transmitter

To statistically evaluate the performance of the receiver, a Xilinx FPGA-based evaluation board was used to simulate the CDMA multiuser environment. This digital multiuser transmitter contains four single-user transmitters. Each user's data is differentially encoded and spread with the designated spreading codes. This results in output at the rate of 0.5 MHz when users' data rate is 31.25 KHz and the spreading gain is 16. The resulting output is re-clocked at 8 MHz, effectively producing 16 samples per chip. This over-sampled signal is then fed to the digital downconverter board. The external DIP switches on the evaluation board allow us to control the number of users in the system.

5.1.2 Noise Generator

A digital noise generator is used to test the functionality of the receiver in the presence of additive white Gaussian noise (AWGN). Based on the Central Limit Theorem, the noise generator adds 48 10-bit random numbers to produce a Gaussian noise. The incoming *I* and *Q* data are summed with this noise at the Input module before processing.

The prototype system runs at 12.5 MHz, while the rate of arrival of *I* and *Q* data is 1 MHz. Thus there are 12 clocks available for generating each noise value before the next data is available. The noise generator consists of four M-sequence generators and an accumulator. The M-sequence generators are linear feedback shift registers (LFSR) of lengths 28, 29, 30, and 31. By treating the last 10 bits of the shift register as a signed binary, a random number is generated.

At each system clock, four uniform random numbers are generated and added into the accumulator. At the accumulator, this sum is added to the sum from the previous clock. After 12 system clock cycles, the accumulator output is equal to the sum of 12x4 uniform random numbers. This output has a Gaussian distribution characteristic. The incoming *I* and *Q* are then independently added to these random noises and the accumulator is reset to prepare for the next 12 clock cycles.

To create a different noise power, this noise value is scaled down before being added to the incoming data. There are four possible scale factors: 0.03125, 0.0625, 0.0938, and 0.125. Figure 5.1 illustrates the noise generator structure.

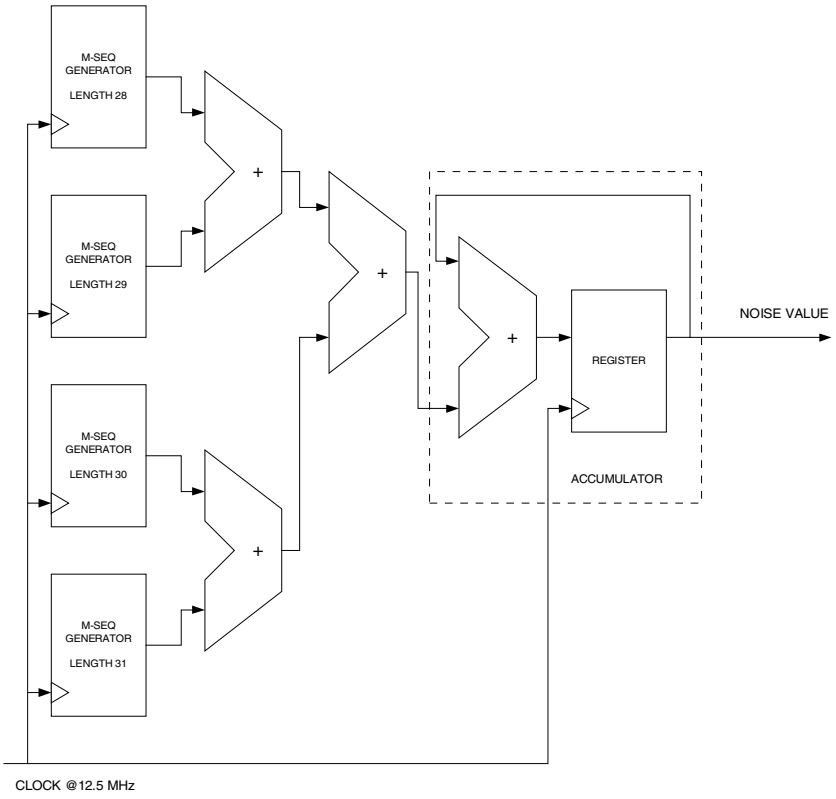


Figure 5.1: Digital noise generator.

By using this technique, however, the resulting BER is overly optimistic, when compared with results from a truly AWGN channel. This is because the sum of the incoming data and noises is truncated to ten bits due to the limited processing resource in the latter module. Thus, the possibility of high value noise is lost. Figures 5.2 – 5.5 show the probability density function (PDF) of noise generated at the Input module with scale factors of 0.03125, 0.0625, 0.0938, and 0.125, respectively.

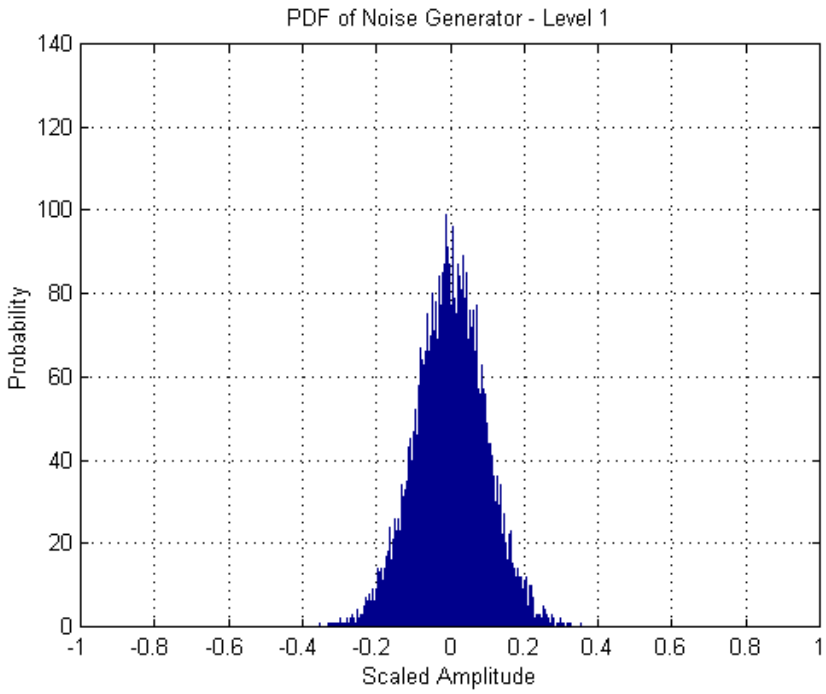


Figure 5.2: PDF of noise level 1 (Magnitude is scaled with a range from –1 to 1).

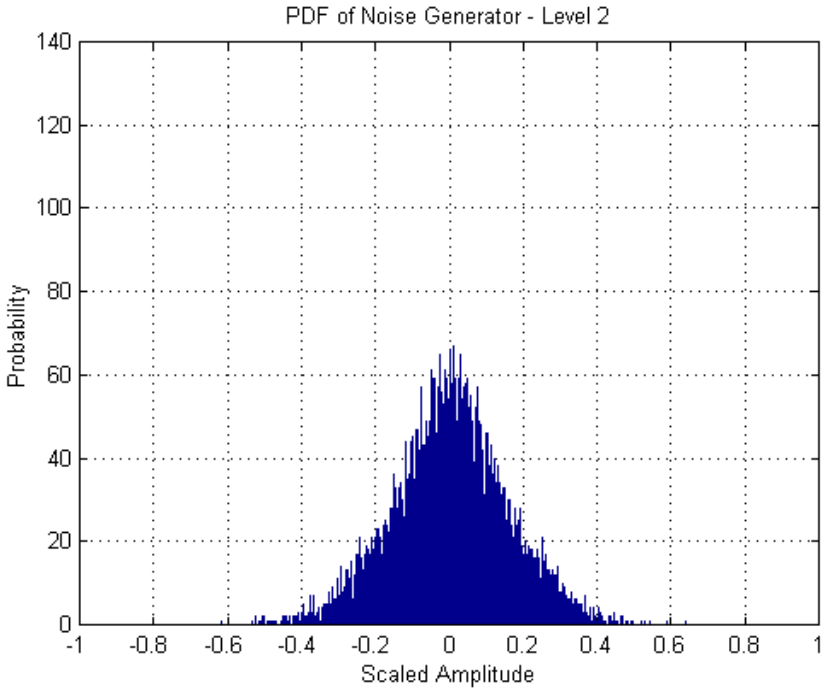


Figure 5.3: PDF of noise level 2 (Magnitude is scaled with a range from –1 to 1).

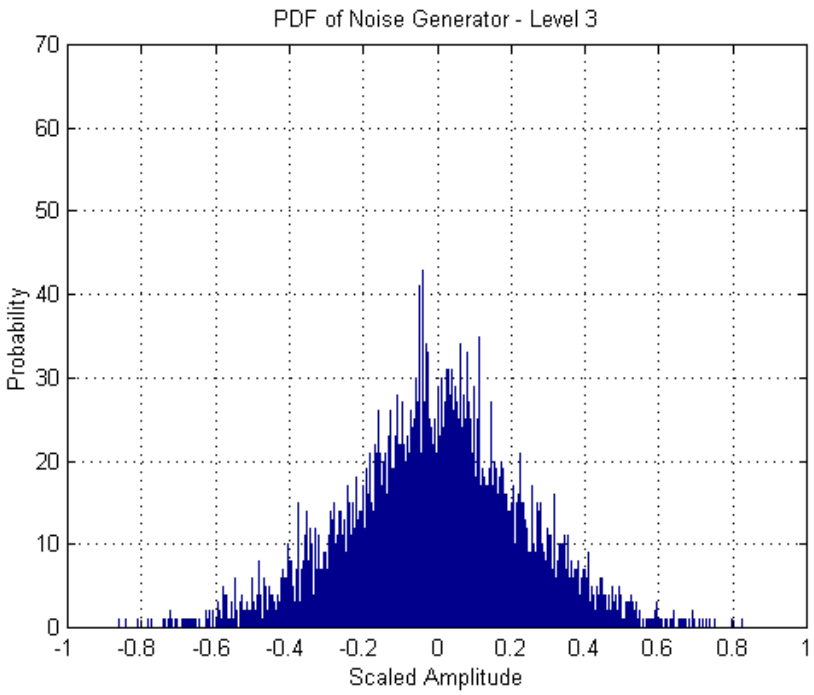


Figure 5.4: PDF of noise level 3 (Magnitude is scaled with a range from –1 to 1).

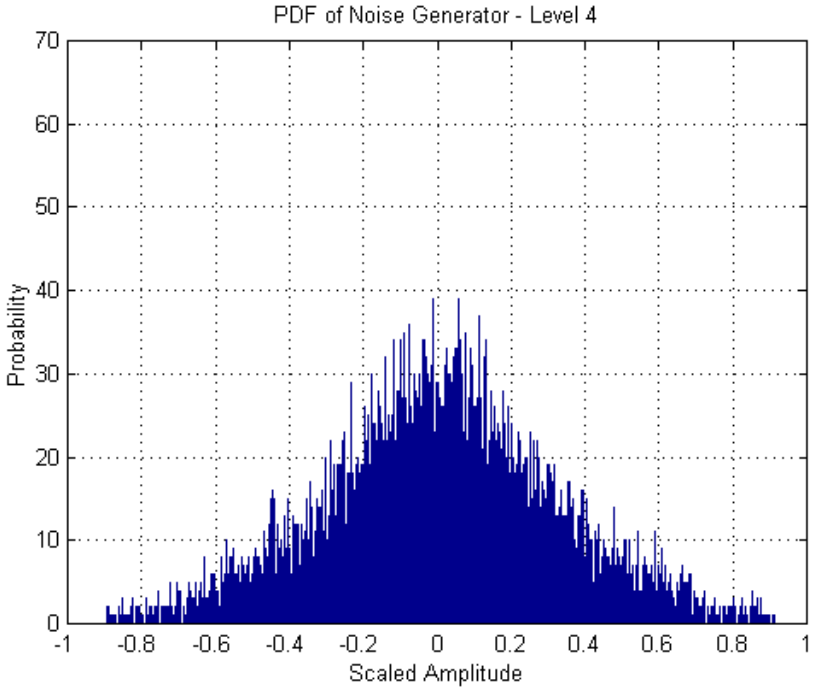


Figure 5.5: PDF of noise level 4 (Magnitude is scaled with a range from –1 to 1).

5.1.3 Receiver Operating Point

The implemented receiver is tested under various noise levels. The operating point of the receiver is determined using the following method. First, the signal power of each user is calculated by using Equation 5.1,

$$P_k = \frac{1}{N}\sum_{n=1}^{N}\left[R_{k,n}(t)R_{k,n}^{*}(t)\right], \tag{5.1}$$

where P_k is the the k^{th} user's signal power per sample, $R_{k,n}$(t) is the n^{th} sample of the k^{th} user's received signal, and N is the number of samples. The average energy per bit is then computed using Equation 5.2,

$$E_b = \frac{\frac{1}{K}\sum_{k=1}^{K} P_k}{R_b}, \tag{5.2}$$

where E_b is the average energy per bit of the user, and R_b is the transmitted data rate, which is equal to 31.25 KHz.

By using the logic analyzer, a snapshot of the complex received samples is recorded into a file in binary format. A MATLAB program is then used to convert these binary values into fractional numbers, allowing us to plug these values into the above equations.

The noise power is then calculated. By turning off the user's signal at the digital transmitter, the noise signal with the different scaled factors is then captured with the logic analyzer. The N_o was calculated for each noise level using,

$$\sigma_n = \frac{1}{N}\sum_{i=1}^{N}\left[n(i)n^{*}(i)\right], \text{ and} \tag{5.3}$$

$$N_o = \frac{2*\sigma_n}{f_s}, \tag{5.4}$$

where *fs* is the sampling frequency at the receiver, 1 MHz. The resulting receiver operation points are listed in Table 5.1.

Noise Level	1	2	3	4
Samples	10000	10000	10000	10000
Mean	0.00368	0.00325	-0.00461	0.00364
Standard Deviation	0.09052	0.15236	0.22727	0.30434
Variance	0.00819	0.02321	0.05165	0.09263
Min.	-0.32226	-0.61523	-0.85937	-0.89453
Max.	0.33203	0.64062	0.82617	0.91015
Eb/No (dB)	14.0	9.66	6.18	3.65

Table 5.1: Receiver operating points at different noise levels.

5.2 Hardware Results

At this point in our study, the receiver has only been tested using baseband signals. By using an XC4010 Xilinx Evaluation board, the digital baseband multiuser signal was created. The system is assumed to be synchronous, where all users' data start at the same time and the power of every signal is equal—as would be expected for the forward channel. The channel is modeled by a 2-ray minimum phase channel model. The multipath component has a power of 3 dB lower than that of the first arriving component and an excess delay of 5 *us*. Figure 5.6 depicts the magnitude response of the 2-ray channel with respect to the magnitude spectrum of the spread DBPSK signal. The deep notches in the magnitude response indicate the frequency-selective nature of the channel.

5.2.1 Case 1: 0 Hz Carrier Frequency Offset

In this case, the carrier frequency offset was set to 0 Hz. Because the signal is already at the baseband, a carrier frequency offset can be achieved by tuning the complex digital mixer on the DDC board to some predetermined offset value.

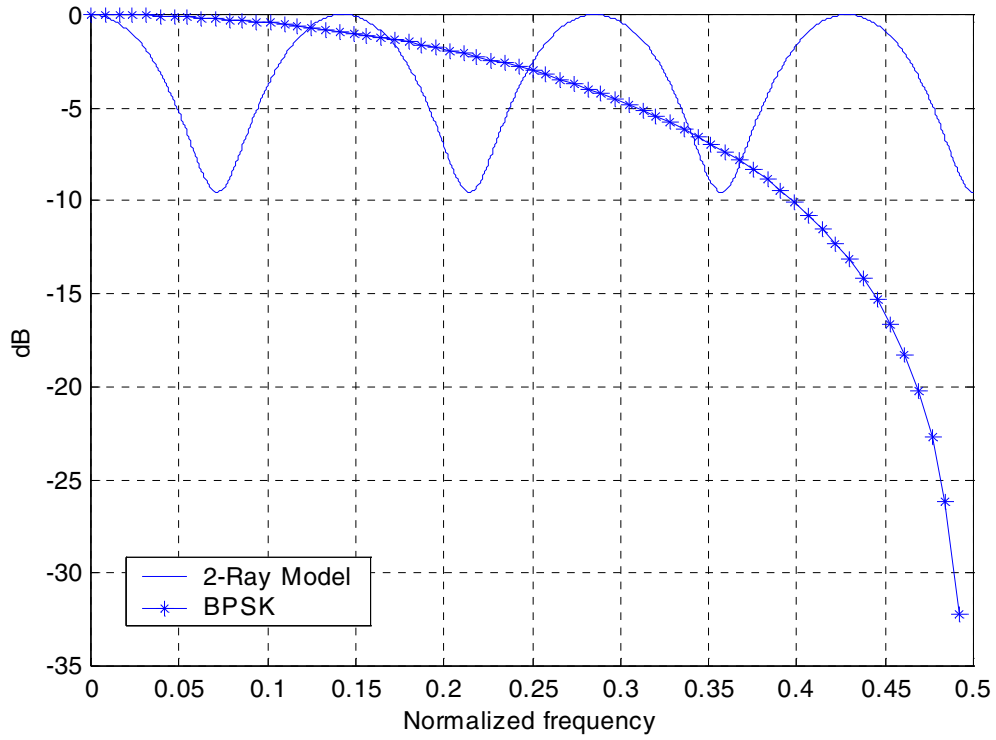


Figure 5.6: Magnitude response of static 2-ray channel model compared with magnitude spectrum of BPSK.

The BER of each user was first measured at different noise levels; then the average value of all users' BERs was calculated. This produced the graph in Figure 5.7. Note that these results are overly optimistic due to the clipping of the noise when summed with *I* or *Q* data at the Input module.

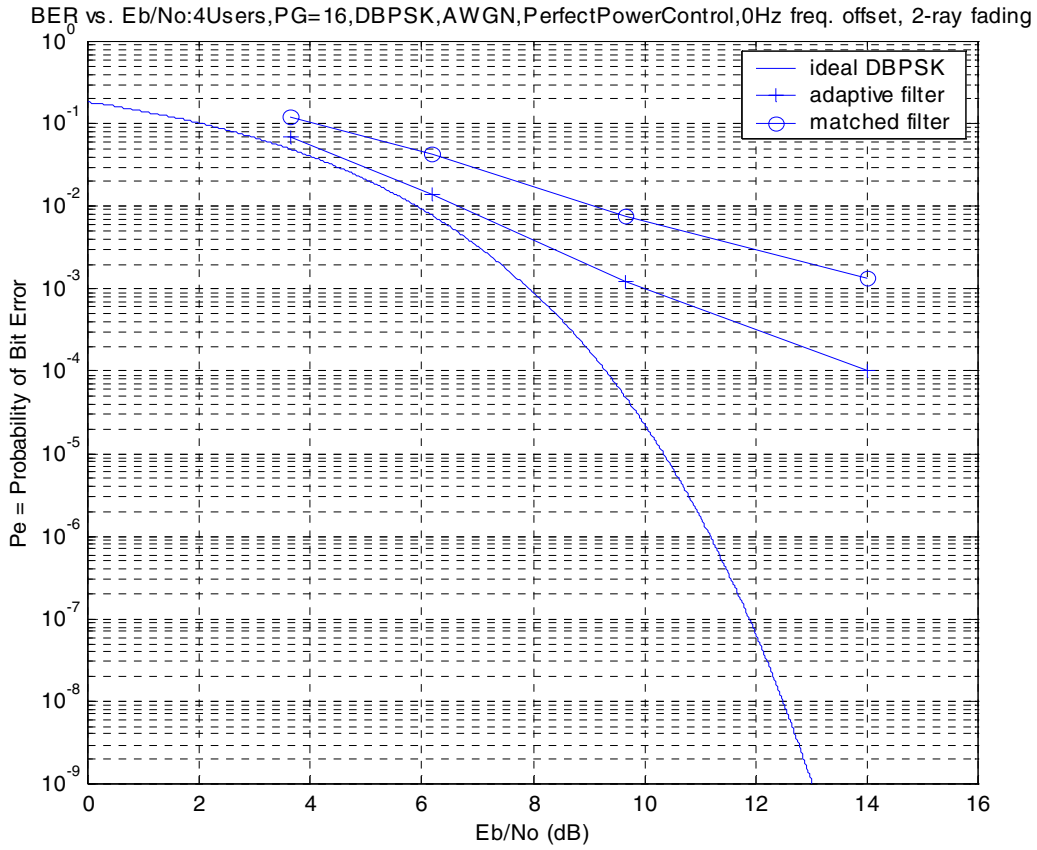


Figure 5.7: BER vs. SNR in 2-ray fading channel with 0 Hz carrier frequency offset.

5.2.2 Case 2: 500 Hz Carrier Frequency Offset

In this case, the carrier frequency offset was set to 500 Hz. This results in a constant constellation rotation of the received signal. Figure 5.8 shows the BER at different signal per noise ratio (SNR).

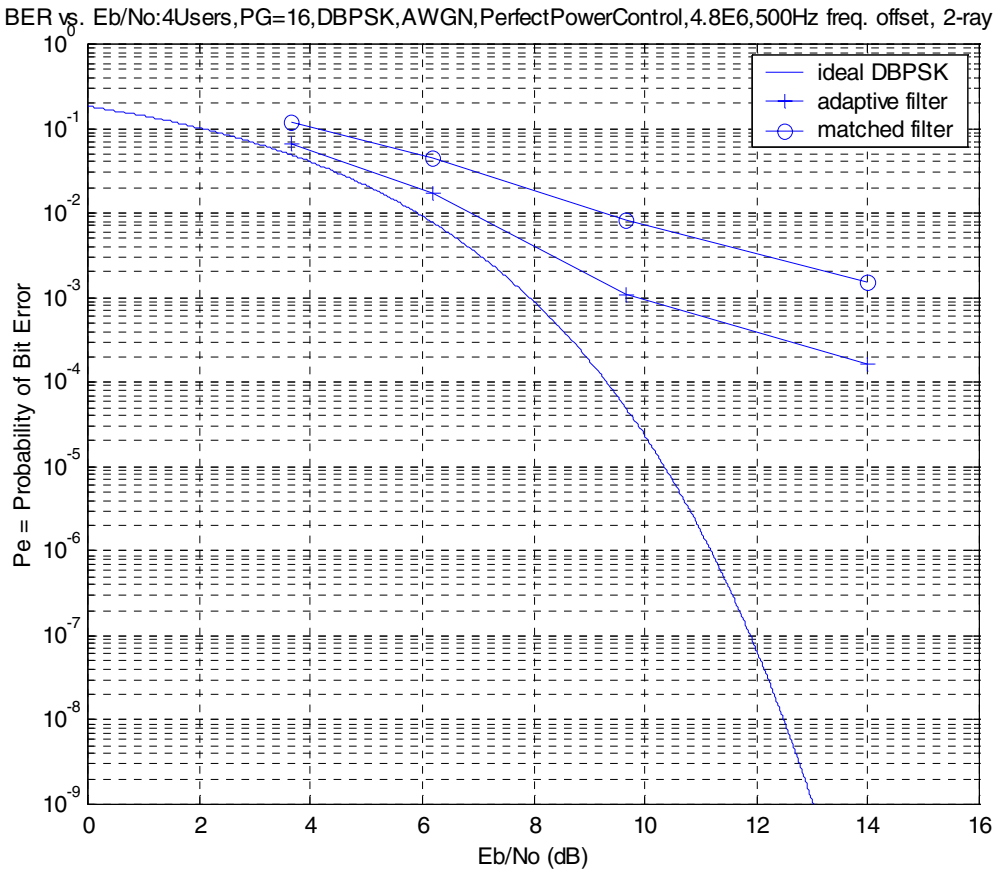


Figure 5.8: BER vs. SNR in 2-ray fading channel with 500 Hz carrier frequency offset.

The BER of a conventional matched filter is also shown for the sake of comparison. From these figures, it can be observed that the BER of the adaptive receiver is improved over that of the matched filter. These improvements are greatly dependent on the input SNR: the figures can be as much as ten times higher when the SNR is higher than 10 dB. One factor that might contribute to this outcome is the fact that the adaptive filter only operates in the decision-directed mode. At a high SNR the receiver output has a low BER, and the weight adaptation is likely to be performed in the right direction. However, at a low SNR the estimated output has a high BER, and by using these outputs for the weight adaptation, the weights are more likely to be updated in the wrong direction.

Also note that for the adaptive filter to fully combine the energies of all multipath components, the filter has to be at least as long as the largest excess delay in the channel. However, due to the limited number of the computing modules, the implemented filter (length 32) is not long enough to hold the longest-delayed component. This results in a low BER improvement over the matched filter. The improvement of the adaptive receiver over the matched filter might be higher if the filter's length could be increased.

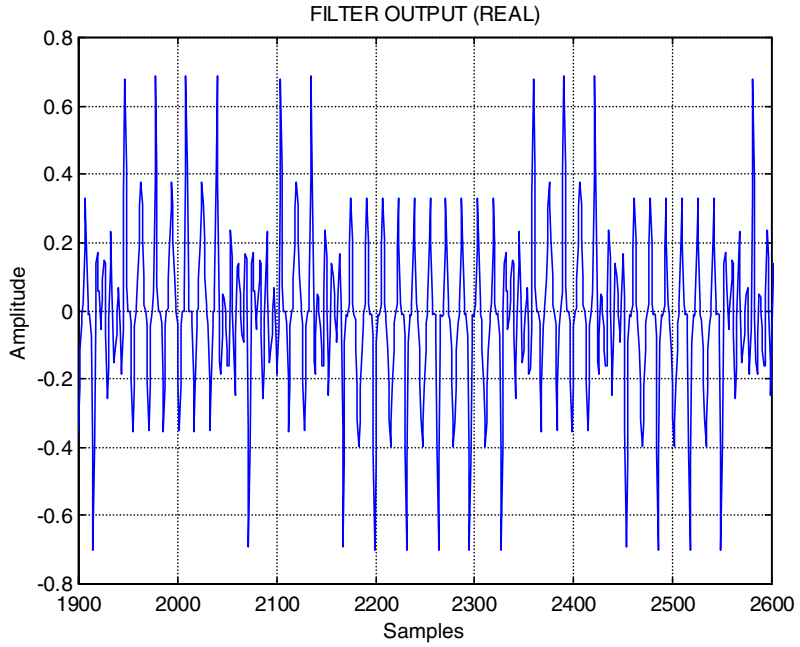
Another observation that can be made from the results is that the carrier frequency offset degrades system performance. This is because the constellation of the received signal is continuously rotating. Adaptive filters try to update their coefficients to follow this rotation, but due to the data precision in the system and the value of the step-size, the coefficients could not be updated fast enough. This results in a suboptimal solution.

5.3 Filter Outputs

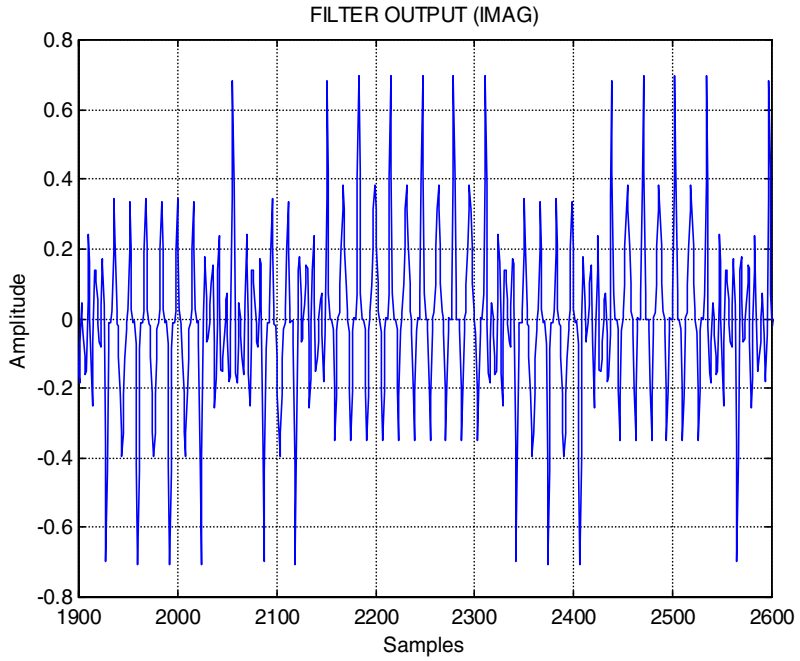
To verify the operation of the implemented adaptive filter, outputs of the filter for different environments were captured using the logic analyzer. As noted earlier, the receiver first starts with the disabled adaptive algorithm. In this period, the receiver acts as a conventional matched filter. After the desired user is acquired, the receiver switches to the adaptive receiver mode. After the adaptive algorithm converges, the absolute value of the peaks of the filter's output should be close to 1.

5.3.1 Case 1: Only user of interest is present in system

Figures 5.9-5.11 illustrate the filter's output for the case when only the desired user is present in the system and the carrier frequency offset is varied. From the figures, the absolute value of the filter output at the peak locations is close to 1 in all cases. Thus, the adaptive filter is functioning correctly.

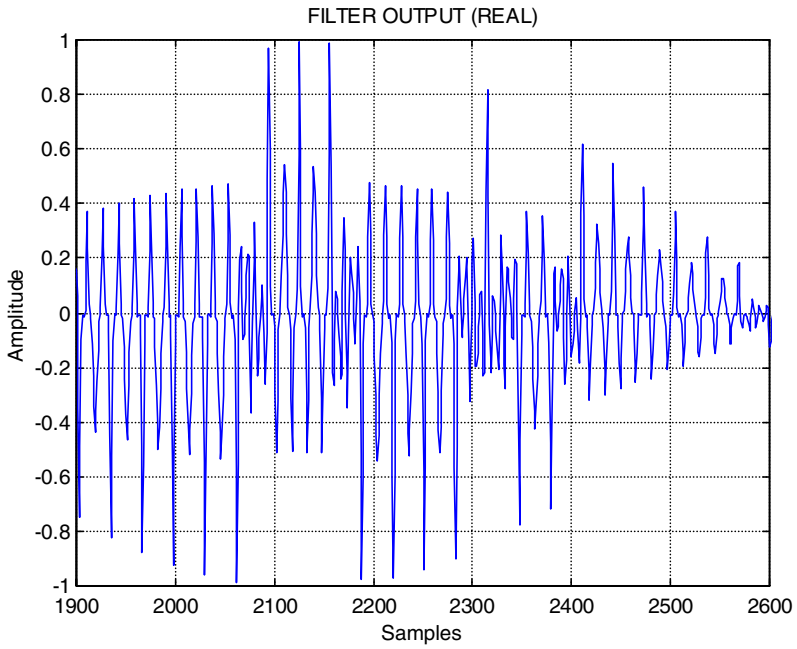


(a)

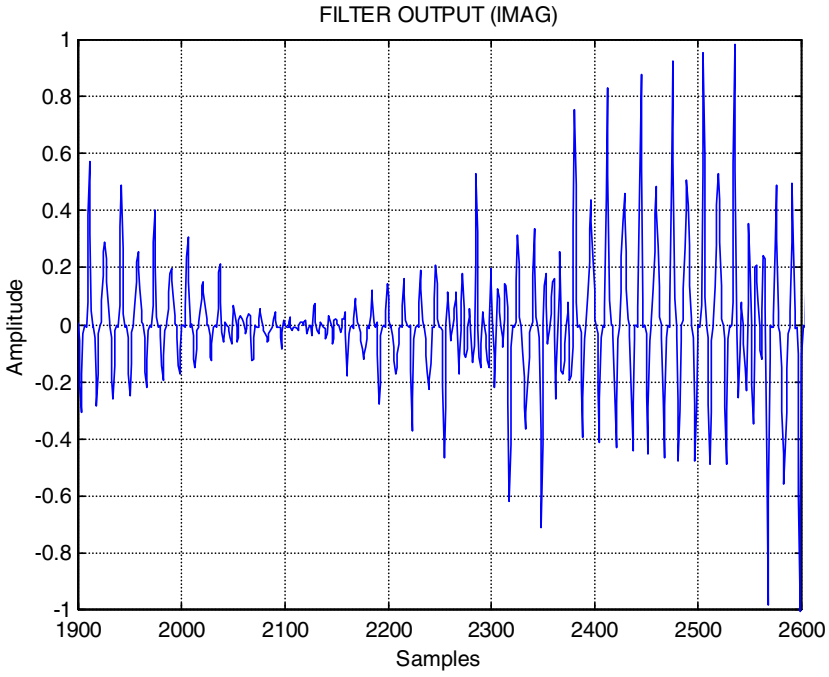


(b)

Figure 5.9: Filter output with 0 Hz carrier frequency offset and 0 degree carrier phase offset.

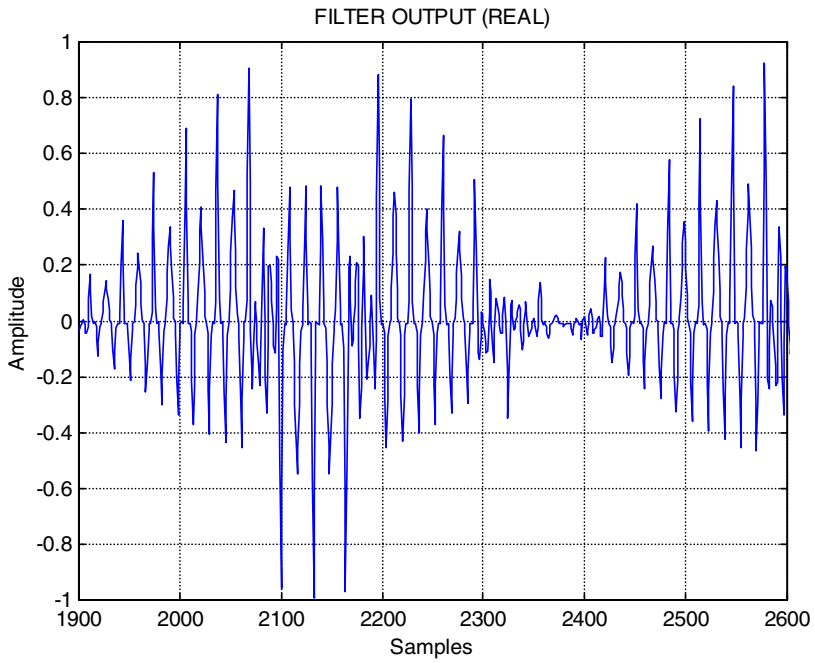


(a)

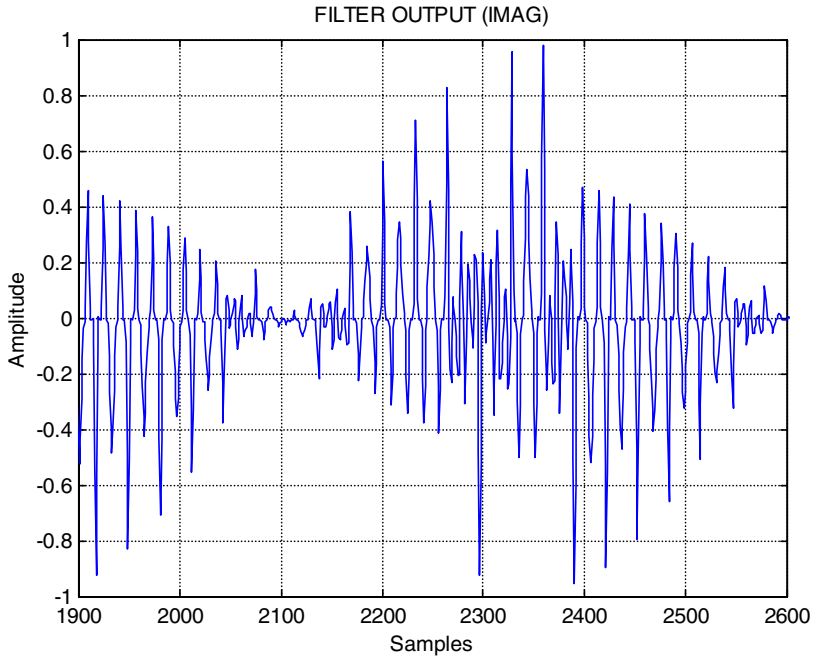


(b)

Figure 5.10: Filter output with 500 Hz carrier frequency offset.



(a)



(b)

Figure 5.11: Filter output with 1000 Hz carrier frequency offset.

5.4 Comparison to MATLAB-based Adaptive Receiver

To further verify the functionality of the implemented receiver, a MATLAB-based adaptive receiver was devised. The parameters—e.g., number of users, spreading codes, and channel model—used in this receiver are the same as those used in the FPGA-based receiver. The BERs of the MATLAB-based and FPGA-based receiver are shown in Figure 5.12 for a 0 Hz carrier frequency offset and 5.13 for a 500 Hz carrier frequency offset. From the figures, the BER of the receivers agree with each other. The small difference between these two BERs comes from the fact that the quantization error in the receivers is not exactly the same. In the FPGA-based receiver, most of the data path is ten bits wide, but there are some parts that are wider than ten bits to reduce as much quantization error as possible. However, the MATLAB-based receiver is assumed to have a 12-bit data path throughout the system. The difference in this quantization error is one factor contributing to the difference in the BER.

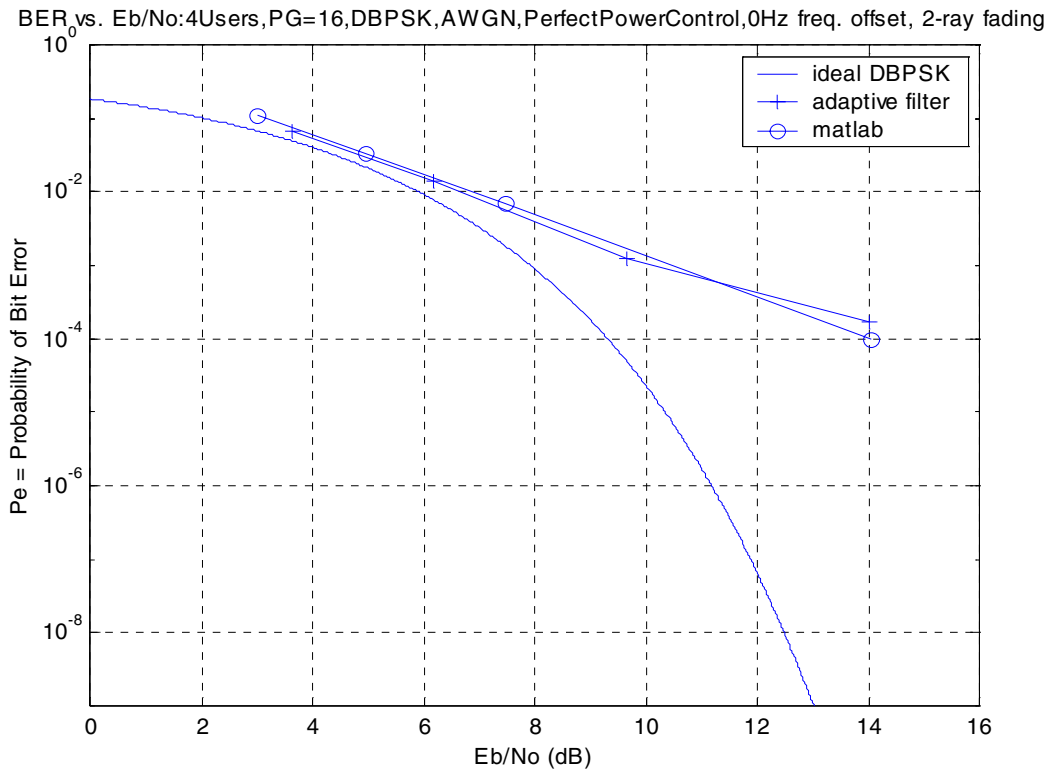


Figure 5.12: BER vs. SNR of the FPGA-based and MATLAB-based adaptive receiver.

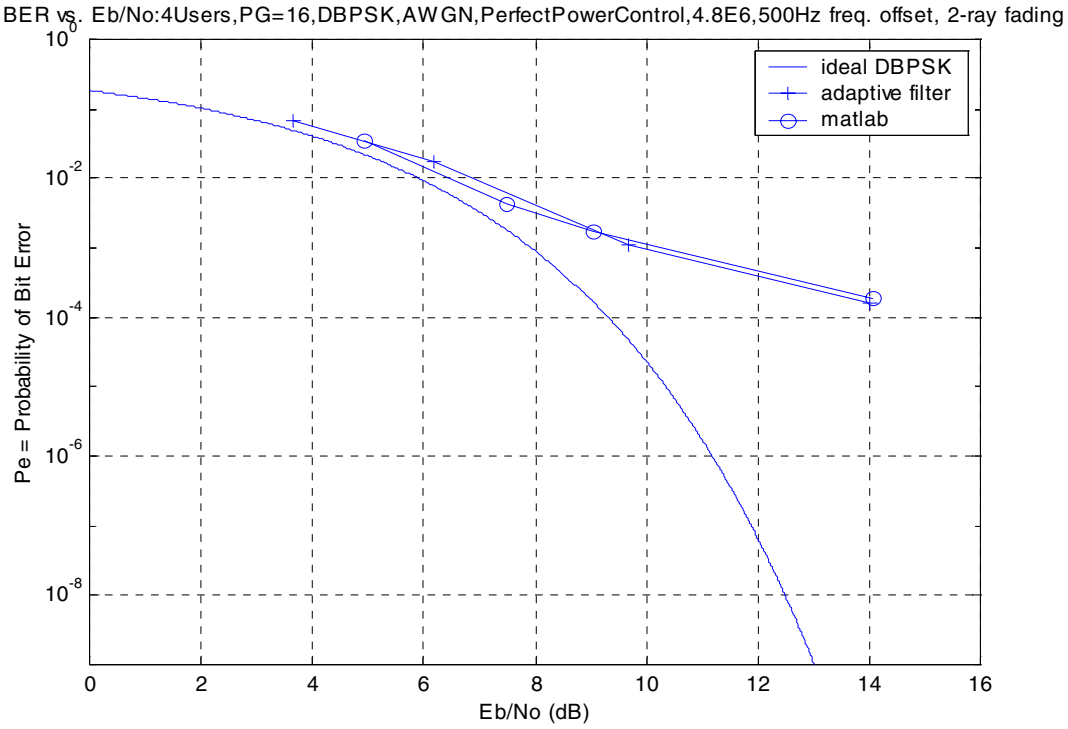


Figure 5.13: BER vs. SNR of the FPGA-based and MATLAB-based adaptive receiver.

Chapter 6

Conclusion and Future Work

This thesis describes a differentially coherent adaptive receiver implemented with an FPGA-based configurable computing platform. The designed architecture provides a viable solution to the intensive computational requirements of the receiver. This architecture, with slight alterations, could be used for other digital signal processing tasks. This chapter starts with a brief summary of the thesis. Section 6.2 presents suggestions for future work, and Section 6.3 gives the conclusions of this thesis.

6.1 Summary

The design and implementation of the differentially coherent adaptive filter receiver is detailed in this thesis. The target system is an FPGA-based configurable computing platform called the GigaOps G900. Chapter 1 gave a brief description of the CDMA system and its major obstacles, i.e., the near-far problem and multiple access interference. The solution to this problem for single-user detection was also described. The motivation for using an FPGA-based configurable computing platform was also elaborated.

Chapter 2 provided insight into CDMA systems and their special characteristics. A class of CDMA single-user detection techniques employing time-dependent adaptive filters was presented. Chapter 2 also discussed a problem commonly found in the mobile environment, i.e., carrier frequency offset and the solution to this problem. Chapter 3 provided an overview of Xilinx's field programmable gate array devices used in this research as well as a detailed description of the target configurable computing platform, the GigaOps G900. The stream-based architecture, on which the receiver is based, was

also described in this chapter. This architecture is based on dividing the implemented algorithm into smaller operations which can be run in parallel, resulting in a higher system throughput. Another feature of this architecture is its single data path between adjacent modules, resulting in a decreased circuit routing requirement.

Chapter 4 first detailed the system partitioning. Due to the receiver's computational requirements, its structure is divided and implemented on several smaller modules. Each module is responsible for only one designated task. This allows the designers to design and implement each module separately. Furthermore, for modules that implement computationally intensive operations, e.g., the filter operations, a pipelining technique is incorporated. This gives a better resource utilization but also requires a more complicated design. Chapter 4 concluded with hardware mapping between partitioned modules to the configurable computing platform.

Chapter 5 presented techniques used to verify and measure the performance of the implemented receiver in the different mobile environments. The receiver was tested at the baseband with various degrees of noise and carrier frequency offsets. Carrier frequency offset degrades the receiver performance since the constellation of the received signal is rotating. An adaptive receiver updates its coefficients in order to track this rotation. With the high carrier frequency offset, however, the coefficients cannot be updated fast enough. This results in a higher BER than in the case of a small offset. Noise also degrades the system performance. This results from the fact that the implemented adaptive filter operates in the decision-directed mode only. With the presence of severe noise, the estimated symbol is more likely to be in error at the point where decisions are made. Thus the filter's weights are updated incorrectly, leading to degradation of receiver performance.

6.2 Suggestions for Future Work

This section focuses on modifications that could increase overall system utilization and performance.

6.2.1 Input Module

In the current receiver, the output of the adaptive filter is calculated for every input data sample. This is necessary when the receiver is in the acquisition phase. However, when the user is acquired, there is no need to perform all of these calculations. It is sufficient for the system to calculate filter outputs only when the samples of a symbol are aligned with the filter. For tracking, it is also necessary to calculate the output one sample before and after this alignment point. When the input sample window is aligned, the output is necessary for estimating the received user data and for the weight update mechanism. When the symbol's samples are one sample off from aligning with the filter, the outputs are necessary for the tracking mechanism. Only computing the filter output at these three samples would allow the system to run at a lower clock speed (reducing power consumption) or to support higher data rates.

This change requires modifications to the Input and Adaptive Filter & Decision modules. The Input module needs to generate an additional operation type that controls the filter output calculations. Until the Adaptive Filter & Decision module receives the appropriate code, the *I* and *Q* data shift into the filters taps without an output calculation. When this operation type is received (at the last sample of a symbol), the Adaptive Filter & Decision module calculates the corresponding output.

6.3 Conclusions

This thesis has demonstrated a practical approach for implementing a computationally complex, differentially coherent adaptive filter receiver for CDMA systems. The receiver is implemented on a standard off-the shelf configurable computing machine. Currently, the receiver has been tested in a baseband environment. Digital white Gaussian noise was generated to verify the functionality of the receiver for various signal-to-noise ratios.

The architecture of the receiver is based on stream-based processing. The incoming data is processed by a series of processing elements which run in parallel at high speed. This produces high system throughput and low latency, which are essential for the receiver. By employing the stream-based concept, the design and implementation

time is reduced. Furthermore, the system portability is increased due to that fact that the architecture has only one data path throughout the system. The architecture also includes some flexibility by allowing programming information to be merged into the data path to modify the operation of the processing elements.

The implemented adaptive filter receiver is shown to have a good BER in the presence of the multiple access interference and fading channel environments. The designed receiver supports a variable-length FIR filter. However, due to the available computing modules, the implemented receiver has a small filter length. With larger FPGAs, which have been recently become available, or a larger number of computing modules, the length of the spreading code and corresponding processing gain can be increased.

Bibliography

[1] N. R. Mangalvedhe, "Development and Analysis of Adaptive Interference Rejection Techniques for Direct Sequence Code Division Multiple Access Systems," Ph.D. Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, July 1999.

[2] S. F. Swanchara, "Design and Implementation of an FPGA-Based Multiuser Receiver," Master's Thesis, Virginia Polytechnic Institute and State University, Blacksburg, July 1998.

[3] M. V. Majmundar, "Adaptive Single-User Receiver for Direct Sequence CDMA Systems," Master's Thesis, Virginia Polytechnic Institute and State University, April 1996.

[4] V. Aue, "Optimum Linear Single-User Detection in Direct-Sequence Spread Spectrum Multiple Access Systems," Master's Thesis, Virginia Polytechnic Institute and State University, Blacksburg, March 1994.

[5] C.-K. Chen, "Spectral Correlation Characterization of Modulated Signals with Application to Signal Detection and Source Location," Ph.D. Dissertation, University of California, Davis, 1988.

[6] J. H. Reed, T. C. Hsia, "The Performance of Time-Dependent Adaptive Filters for Interference Rejection," *IEEE Transactions on Acoustics, Speech and Signal Processing,* vol. 38, no. 8, pp. 1373-1385, August 1990.

[7] W. A. Gardner, "Cyclic Wiener Filtering: Theory and method," *IEEE Transactions on Communications,* vol. 41, no. 1, pp. 151-163, January 1993.

[8] V. Aue, J. H. Reed, "An Interference Robust CDMA Demodulator that Uses Spectral Correlation Properties," *IEEE Vehicular Technology Conference,* pp. 563-567, 1994.

[9] J. Litva and T. K.-Y. Lo, *Digital Beamforming in Wireless Communications*, Artech House Publishers, Boston, 1996.

[10] D. T. M. Slock, "On the Convergence Behavior of the LMS and the Normalized LMS Algorithms," *IEEE Transactions on Signal Processing*, vol. 41, no. 9, pp. 2811-2825, September 1993.

[11] M. Tarrab and A. Feuer, "Convergence and Performance Analysis of the Normalized LMS Algorithm with Uncorrelated Gaussian Data," *IEEE Transactions Information Theory*, vol. IT-34, no. 4, pp. 680-691, July 1988.

[12] N. J. Bershad, "Analysis of the Normalized LMS Algorithm with Gaussian Inputs," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-34, pp. 793-806, August 1986.

[13] W. A. Gardner, "Learning Characteristics of Stochastic-Gradient-Descent Algorithms: A General Study, Analysis, and Critique," *Signal Processing*, vol. 6, pp. 113-133, 1984.

[14] L. L. Horowitz and K. D. Senne, "Performance Advantage of Complex LMS for Controlling Narrow-Band Adaptive Arrays," *IEEE transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-29, pp. 722-735, June 1981.

[15] A. Feuer and E. Weinstein, "Convergence Analysis of LMS Filters with Uncorrelated Gaussian Data," *IEEE transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-33, no. 1, pp. 222-230, February 1985.

[16] J. B. Foley and F. M. Boland, "A Note on the Convergence Analysis of LMS Adaptive Filters with Gaussian Data," *IEEE transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-36, no. 7, pp. 1087-1089, July 1988.

[17] B. Widrow *et al.*, "Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter," *Processing of IEEE*, vol. 64, no. 8, pp.1151-1162, August 1976.

[18] GigaOperations Corporation, Berkeley, CA, *SPECTRUM Reconfigurable Computing Platform Documentation*, 1997.

[19] Sigtek, Inc., Columbia, MD, *ST-114 HSP50214 Harris Downconverter Evaluation Board*, 1997.

[20] J. Henkelman and D. Damerow, "Calculating Maximum Processing Rates of the HSP50124 PDC," Application Note AN9720.1, Harris Semiconductor, Feb. 1998.

[21] Xilinx, *The Programmable Logic Data Book,* 1998.

[22] P. Bertin *et al.*, "Programmable Active Memories: Reconfigurable Systems Come of Age," *IEEE transactions on Very Large Scaled Integration (VLSI) Systems*, vol. 4, no. 1, pp. 56-69, 1996.

[23] M. Wazlowski and L. Agarwal, "PRISM-II Compiler and Architecture," *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 9-16, 1993.

Vita

Prinya Atiniramit was born in Bangkok, Thailand on July 18, 1975. He received his Bachelor of Science (BS) degree first class honor from King Mongkut's Institute of Technology Ladkrabang. After working one year for Submicron Technology, Thailand, he decided to join Virginia Tech as a Master's student in 1997. He later joined the Configurable Computing Machine Lab (CCM) in December of 1997. His area of concentration is configurable computing machine and computer architectures. He will join Matrox Tech. in October 1999.