

# An FPGA–Based Multiuser Receiver Employing Parallel Interference Cancellation

Steven F. Swanchara

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, Co-chair

Jeffery H. Reed, Co-chair

Brian D. Woerner

July 22, 1998

Blacksburg, Virginia

Keywords: FPGA, CDMA, Multiuser Receiver, Interference Cancellation, Configurable  
Computing

Copyright 1998, Steven F. Swanchara

# An FPGA–Based Multiuser Receiver Employing Parallel Interference Cancellation

Steven F. Swanchara

(ABSTRACT)

Research efforts have shown that capacity in a DS/CDMA cellular system can be increased through the use of digital signal processing techniques that exploit the nature of the multiple access interference (MAI). By jointly demodulating the users in the system, this interference can be characterized and reduced thus decreasing the overall probability of error in the system. Numerous multiuser structures exist, each with varying degrees of complexity and performance. However, the size and complexity of these structures is large relative to a conventional receiver. This effort demonstrates a practical approach to implementing parallel interference cancellation applied to DBPSK DS/CDMA on an FPGA–based configurable computing platform. The system presented acquires, tracks, cancels, and demodulates four users independently and performs various levels of interference cancellation. The performance gain of the receiver in a four–user environment under various levels of noise and cancellation are presented.

# Acknowledgments

I would like to thank Professors Peter M. Athanas and Jeffery H. Reed for being my co-advisors. I would also like to thank Professor Brian D. Woerner for being on my committee and supplying corrections.

I am deeply grateful for all the help I have received during my graduate studies at Virginia Tech from the faculty, staff, and students of the Mobile and Portable Radio Research Group (MPRG) and the VT Configurable Computing team. I would especially like to thank the following people: Peter Athanas, Neiyer Correal, Scott Harper, Lori Hughes, Nitin Mangalvedhe, and Martin Pechanec.

This thesis is dedicated to my wife, Jennifer, and my parents for their constant support during my studies at Virginia Tech.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Organization of Thesis . . . . .	2
1.3	Contribution of this Work . . . . .	3
<b>2</b>	<b>Interference Cancellation</b>	<b>5</b>
2.1	Direct-Sequence Spread Spectrum . . . . .	5
2.2	Multiuser Detection . . . . .	6
2.2.1	Successive Interference Cancellation . . . . .	7
2.2.2	Parallel Interference Cancellation . . . . .	7
2.3	Non-Coherent Parallel Interference Cancellation . . . . .	8
2.3.1	DSP-Based Implementation . . . . .	8
2.3.2	FPGA-Based Implementation . . . . .	11
2.3.3	Computational Requirements . . . . .	14
2.4	Summary . . . . .	14

<b>3</b>	<b>Configurable Computing</b>	<b>16</b>
3.1	Field Programmable Gate Arrays . . . . .	17
3.1.1	FPGA Design Process . . . . .	18
3.1.2	FPGA Synthesis & Implementation . . . . .	19
3.1.3	Run-Time Reconfiguration . . . . .	20
3.2	GigaOps G900 Configurable Computing Platform . . . . .	21
3.3	Motivation for FPGA-Based Implementation . . . . .	24
3.4	Stream-Based Modular Design Concept . . . . .	25
3.4.1	Module Structure . . . . .	26
3.4.2	Module State Machine . . . . .	27
3.5	Summary . . . . .	29
<b>4</b>	<b>Multiuser Receiver Implementation</b>	<b>30</b>
4.1	Overview . . . . .	30
4.2	Prototype System Specifications . . . . .	31
4.3	FPGA-Based Multiuser Receiver Testbed . . . . .	32
4.4	Algorithm Implementation . . . . .	33
4.4.1	Input Module . . . . .	35
4.4.2	Matched Filter Correlator Module . . . . .	39
4.4.3	Acquisition/Tracking Module . . . . .	43
4.4.4	Regenerate Module . . . . .	48
4.4.5	Combine Module . . . . .	49

4.4.6	Revised Module . . . . .	49
4.4.7	Buffer Module . . . . .	50
4.4.8	Differential Demodulator Module . . . . .	51
4.4.9	Output Module . . . . .	52
4.5	Algorithm Mapping and Synthesis . . . . .	54
4.5.1	<i>INPUT1</i> XMOD . . . . .	55
4.5.2	<i>MFC1</i> XMOD . . . . .	57
4.5.3	<i>MURX1</i> XMOD . . . . .	60
4.5.4	<i>INPUT2</i> XMOD . . . . .	61
4.5.5	<i>MFC2</i> XMOD . . . . .	63
4.5.6	<i>MURX2</i> XMOD . . . . .	63
4.6	Implementation Summary . . . . .	66
4.6.1	Computational Performance . . . . .	67
4.6.2	External Memory Access . . . . .	67
4.7	Host-Based Receiver Control Application . . . . .	68
4.8	Summary . . . . .	68
<b>5</b>	<b>Results</b>	<b>70</b>
5.1	System Calibration . . . . .	70
5.2	FPGA-Based Multiuser Transmitter . . . . .	71
5.3	Noise Generator . . . . .	72
5.4	Hardware Results . . . . .	75

5.4.1	Case 1 : 0 Hz Carrier Offset . . . . .	76
5.4.2	Case 2 : 500 Hz Carrier Offset . . . . .	77
5.4.3	Comparison to Expected Results . . . . .	81
5.5	Summary . . . . .	82
<b>6</b>	<b>Summary and Future Work</b>	<b>85</b>
6.1	Summary of Results . . . . .	85
6.2	Future Work . . . . .	87
6.2.1	Short-Term System Upgrades . . . . .	87
6.2.2	Long-Term System Upgrades . . . . .	89
6.3	Conclusions . . . . .	90

# List of Figures

2.1	Structure of algorithm used in DSP-based implementation. . . . .	12
2.2	Structure of algorithm used in FPGA-based implementation. . . . .	15
3.1	Internal structure of Xilinx 4000 series CLB [13]. . . . .	18
3.2	Diagram of the GigaOps G900 configurable computing platform. . . . .	23
3.3	Adopted stream packet format for the multiuser receiver. . . . .	26
3.4	Internal structure of module that operates on streams. . . . .	27
3.5	Diagram of simplified module state machine. . . . .	28
4.1	Diagram of the FPGA-based receiver testbed. . . . .	32
4.2	Diagram of the modularized receiver structure. . . . .	34
4.3	Organization of the Input module stream output. . . . .	37
4.4	Diagram of stream programming header. . . . .	38
4.5	Diagram of the Matched Filter Correlator module processing pipeline. . . . .	41
4.6	Boundary conditions that occur due to the tracking process. . . . .	47
4.7	Diagram of the Demodulator module processing pipeline. . . . .	52
4.8	Diagram of the Output module. . . . .	53

4.9	Memory-mapped registers within the Output module. . . . .	54
4.10	Hardware block diagram of the G900-based receiver. . . . .	56
4.11	Diagram of the <i>INPUT1</i> XMOD. . . . .	58
4.12	Diagram of the <i>MFC1</i> XMOD. . . . .	58
4.13	Diagram of the <i>MURX1</i> XMOD. . . . .	61
4.14	Diagram of the <i>INPUT2</i> XMOD. . . . .	62
4.15	Diagram of the <i>MFC2</i> XMOD. . . . .	64
4.16	Diagram of the <i>MURX2</i> XMOD. . . . .	65
5.1	PDF of Digital Noise Generator at Level 1. . . . .	74
5.2	PDF of Digital Noise Generator at Level 2. . . . .	75
5.3	PDF of Digital Noise Generator at Level 3. . . . .	76
5.4	Probability of bit error versus $E_b/N_o$ for FPGA-based multiuser receiver with 0 Hz offset. . . . .	78
5.5	Probability of bit error versus $E_b/N_o$ for FPGA-based multiuser receiver with 500 Hz offset. . . . .	79
5.6	Example of amplitude error introduced by frequency offset. . . . .	80
5.7	Comparison of FPGA-based receiver with Matlab simulation. . . . .	83
5.8	Results from DSP-based non-coherent receiver [26]. . . . .	84

# List of Tables

4.1	Utilization of processing bandwidth. . . . .	39
4.2	Rules for translating desired correlator weight to filter configuration bit. . . .	43
4.3	Synthesis results for the <i>INPUT1</i> XMOD. . . . .	57
4.4	Synthesis results for the <i>MFC1</i> XMOD. . . . .	59
4.5	Synthesis results for the <i>MURX1</i> XMOD. . . . .	61
4.6	Synthesis results for the <i>INPUT2</i> XMOD. . . . .	63
4.7	Synthesis results for the <i>MFC2</i> XMOD. . . . .	64
4.8	Synthesis results for the <i>MURX2</i> XMOD. . . . .	66
4.9	Summary of estimated gate count of final FPGA implementations. . . . .	66
4.10	Breakdown of arithmetic operations per module. . . . .	67
5.1	Statistics from output of the digital noise generator. . . . .	73
5.2	Normalized BER performance gain for Case 1. . . . .	77
5.3	Normalized BER performance gain for Case 2. . . . .	78

# Chapter 1

## Introduction

The past ten years have brought remarkable advances to the wireless communications industry. Among these advances is the ongoing development of wideband digital standards. More users are accommodated at higher data rates and expanded service. Yet these advances are still limited by system capacity, an obstacle that continues to challenge wireless systems engineers. As more users are placed onto the system, more interference is generated, degrading call quality throughout the system. By guaranteeing a minimum call quality, each provider sets a limit on the number of concurrent users of their system. If capacity could be increased, more users would be able to use the system, thus generating greater revenue for the service provider.

### 1.1 Motivation

Research efforts have shown that capacity in a DS/CDMA cellular system can be increased through the use of digital signal processing techniques that exploit the nature of multiple access interference (MAI). This is the concept behind multiuser detection. By jointly demodulating the users in the system, the interference can be identified and reduced, thus

decreasing the overall probability of error in the system. Numerous multiuser structures exist, each with varying degrees of complexity and performance. The size and complexity of the resulting structures is greatly increased relative to a conventional receiver. This raises implementation concerns such as algorithm selection and development and choice of the required computing platform.

This effort deals with the practical implementation of a multiuser receiver employing a technique called multistage parallel interference cancellation (PIC). All users are demodulated in parallel, which imposes large computational and complex routing requirements between the components of the receiver. The organization of the receiver is based on a stream-based modular concept that improves the flexibility of the high performance yet static structure of deep computational pipelines. Users are processed serially at a high rate through the system to utilize the same computational resources. This reduces the overall routing and interprocess communication required by the algorithm. Every block of data entering the receiver is processed for each user in the system by rapidly configuring each processing unit along the line to handle the current user. This provides a simple and convenient way to implement interference cancellation in routing-limited situations. If separate, lower-speed receiver subsystems were utilized for each user, the spatial requirements and interconnect would become prohibitively expensive for practical systems with a large number of users.

## 1.2 Organization of Thesis

In this thesis, the methods used to implement and test the FPGA-based multiuser receiver will be addressed as follows. Chapter 2 provides background information on interference cancellation and examples of some existing implementation efforts. Chapter 3 examines configurable computing, focusing primarily on the fundamentals of pipeline processing and the use of Field Programmable Gate Arrays (FPGAs). The FPGA design environment and the modular stream-based design methodology used throughout this effort are also illustrated.

Chapter 4 provides a detailed description of the implementation of the FPGA-based multiuser receiver, including the hardware setup, partitioning of the design into modules, and the mapping of the receiver structure to the FPGA computing platform. In Chapter 5, the procedure and results from baseband digital tests of the testbed are outlined. Analysis and comparison with simulation results and a DSP-based implementation are also performed. A summary of the research effort follows in Chapter 6 along with suggestions for future work.

### 1.3 Contribution of this Work

Although many of the features found in contemporary wireless systems are not fully implemented in this prototype, it does provide adequate proof of concept and insight into practical multiuser receiver design. The primary contributions of this thesis are as follows.

- The implementation of a differentially coherent DS/CDMA PIC scheme using a dataflow-oriented approach amendable to pipeline processing is explained. This technique reduces the routing dependencies between the various stages of the receiver, making them more independent.
- The development of a serial processing scheme reduces the routing requirements of the PIC scheme. Users are processed serially at a higher rate to allow them to share computing resources. This requires a faster processing clock but drastically reduces the size and routing requirements of the overall receiver structure.
- The implementation of a configurable processing pipeline architecture that allows processing elements to be reconfigured via programming information merged into the data path. The operation of the processing elements within the pipeline can be modified although the data paths themselves are fixed.
- The implementation and baseband verification of a differentially coherent BPSK DS/CDMA multiuser receiver employing PIC on an FPGA-based reconfigurable computing plat-

form. The system independently acquires, tracks, regenerates, cancels, and then demodulates up to four users, each at 31.25 kbits/sec and with a processing gain of sixteen.

The implementation presented here is not optimal due to the use of a commercial FPGA computing platform that imposes restrictions on signal routing and on access to external memory. If a custom board utilizing FPGAs or the Stallion chip were developed, much higher device utilization and lower power consumption could be achieved.

# Chapter 2

## Interference Cancellation

In this chapter, the modulation scheme and multiple access technique used in the system are outlined. The concept of multiuser detection is also introduced. The DSP-based approach, upon which this effort is based, is also outlined along with the necessary modifications to the parallel interference cancellation scheme that make it more amenable to an FPGA implementation.

### 2.1 Direct-Sequence Spread Spectrum

The communications system described in this work utilizes the modulation technique known as direct-sequence spread spectrum (DS/SS). In a DS/SS system, each user's narrowband data signal is spread over a wider frequency band by mixing it with a higher rate pseudo-random (PN) sequence or code. Different users in the system can occupy the same frequency since they are separated somewhat due to the low cross correlation of their PN codes. This is known as code division multiple access (CDMA). The codes, their alignment, and their received power levels dictate the amount of interference the users impose on one another. Interference caused by others using the system is called multiple access interference (MAI).

As users are added to the system, the MAI level increases, thus degrading the quality of all links in the system and imposing a soft limit on the capacity of the system.

One primary disadvantage to DS/CDMA is the near–far problem. If one user is close to the receiver, that user’s signal is received at a higher power level than others transmitting with equal power but located further out. This degrades the reception of the more distant users and requires power control to ensure that all users are received at approximately the same power level.

## 2.2 Multiuser Detection

Conventional DS/CDMA receivers have knowledge of and demodulate a single desired user. The MAI created by the lack of orthogonality between the desired user and others present in the system is treated as additional noise. It has been shown that improved performance in a multiuser environment can be achieved by taking into account the structure of the MAI. An optimal technique for multiuser detection in AWGN channels was presented in [1], but its high complexity hinders practical implementation. Other sub–optimal lower–complexity techniques, whose performance still exceeds that of the conventional approach, have been proposed [2]. Of particular interest are the class of detectors based upon interference cancellation. Interference cancellers reconstruct the MAI based upon estimates of users present in the system and subtract it from the received signal. Most notable within this class are the methods of successive and parallel interference cancellation.

Multiuser detection is most likely to be used in the base stations of DS/CDMA systems since they are on the receiving end of the reverse link. Mobile users transmit on the reverse link asynchronously with respect to one another, making the degree of orthogonality between users unpredictable and potentially harmfully low. Also, there is an inherent need for base stations to demodulate and maintain knowledge of all users within their domain. In addition, the semi–fixed location of base stations makes receiver size and complexity less of an issue.

### 2.2.1 Successive Interference Cancellation

In successive interference cancellation [3], the users are first ranked in order of received power level. The strongest users are then estimated, regenerated, and subtracted from the received signal. The resulting corrected signal is then used to demodulate the weaker users in the system. This approach helps only the weaker users in the system; however, the weaker users will likely be the most adversely affected by MAI. Successive cancellation is preferred over parallel cancellation when received signal powers are widely disparate.

An FPGA-based solution for successive interference cancellation is presented in [4]. This effort utilizes a pipeline structure for regeneration of the strongest users present in an EIA/TIA/IS-95 system. Although the entire standard was not able to be fully implemented on the FPGA device, the performance gain achieved from using successive interference cancellation was verified.

### 2.2.2 Parallel Interference Cancellation

In parallel interference cancellation, initially proposed in [5, 6], each user's signal in the system is estimated and regenerated. Then, in parallel, all users' signals are demodulated using a revised signal formed by subtracting off the estimate of the other users in the system from the actual received signal. This is done in consecutive stages with each refining the estimate and improving performance. Parallel cancellation outperforms successive cancellation when received signal powers are roughly equal such as within a cellular system.

A custom ASIC-based solution known as the CESSIUM was developed by CEA-LETI in France [7]. It incorporates estimation and regeneration of a single user's signal into a conventional DS/CDMA transceiver chip. This single chip solution can then be combined into a larger system to perform subtractive interference cancellation for all users over multiple stages. In the first stage, each user in the system would have a chip that produces an estimate of his or her received signal. In the second stage, each user has a dedicated chip that

recovers that user's data stream after cancellation of the other users.

At the input to each chip is an optional summation circuit that accumulates the regenerated signals of five other users. Thus the system requires routing of each user's regenerated signals from the first stage to each chip in the second stage. This is a prime example of the routing bottleneck inherent to parallel interference cancellation. For practical systems of fifteen to thirty users, this fan-out becomes prohibitively expensive.

## 2.3 Non-Coherent Parallel Interference Cancellation

This work builds upon the design and implementation of the DSP-based multiuser receiver illustrated in [8]. Specifically, it is based upon the non-coherent version of this receiver presented in [9]. Although the receiver complexity is reduced by eliminating the need for carrier synchronization, the amount of processing required is doubled since estimation and cancellation must now be performed on both the  $I$  and  $Q$  channels in parallel.

### 2.3.1 DSP-Based Implementation

The system model used to describe the DSP-based implementation of a two-stage non-coherent receiver employing parallel interference cancellation is outlined. A more generic model from which this one was derived is presented in [9]. A block diagram of the algorithm used in the DSP-based approach is illustrated in Figure 2.1. The complex envelope of the signal transmitted by each user in the system is given by

$$s_{I_k} = \sqrt{P_k} a_k(t) b_k(t) \cos \theta_k(t) \quad (2.1)$$

$$s_{Q_k} = \sqrt{P_k} a_k(t) b_k(t) \sin \theta_k(t) \quad (2.2)$$

where  $P_k$  is the power of user  $k$ ,  $a_k(t)$  is the spreading signal,  $b_k(t)$  is the data signal, and

$\theta_k(t)$  is the received phase. The time-varying nature of  $\theta_k(t)$  is dependent on the frequency and phase offsets between the user's transmitter and the receiver's local oscillator.

In the first stage of the receiver, the signal at the antenna is the composite of all the users presently in the system plus noise as expressed in

$$r_I(t)^{(1)} = n_I(t) + \sum_{k=1}^K s_{I_k}(t - \tau_k) \quad (2.3)$$

$$r_Q(t)^{(1)} = n_Q(t) + \sum_{k=1}^K s_{Q_k}(t - \tau_k). \quad (2.4)$$

A bank of matched filters, one for the  $I$  and one for the  $Q$  channels, correlate the received signal  $r(t)$  with the signature sequence of each user in the system. The acquisition and tracking mechanisms allow the system to determine and maintain the correct delay  $\tau_k$  for each user to provide a sufficient statistic. These yield the decision statistics  $Y_{I_k}$  and  $Y_{Q_k}$ ,

$$Y_{I_k}^{(1)} = \frac{1}{T} \int_{t-T}^t r_I(t) a_k(t - \tau_k) dt \quad (2.5)$$

$$Y_{Q_k}^{(1)} = \frac{1}{T} \int_{t-T}^t r_Q(t) a_k(t - \tau_k) dt, \quad (2.6)$$

which represent the amplitude estimates used when regenerating each user.  $T$  denotes the symbol duration, which is a function of the processing gain,  $N$ , and the number of samples per symbol,  $N_s$ . Using the amplitude estimates, each user's signal is then regenerated to form an estimated received signal  $\hat{s}(t)$ ,

$$\hat{s}(t)_{I_k}^{(1)} = Y_{I_k}^{(1)} a_k(t - \tau_k) \quad (2.7)$$

$$\hat{s}(t)_{Q_k}^{(1)} = Y_{Q_k}^{(1)} a_k(t - \tau_k). \quad (2.8)$$

The users' regenerated signals are added together to form an estimate of the received signal

$\hat{r}(t)$ ,

$$\hat{r}(t)_I^{(1)} = \sum_{k=1}^K C_k^{(1)} \hat{s}(t)_{I_k}^{(1)} \quad (2.9)$$

$$\hat{r}(t)_Q^{(1)} = \sum_{k=1}^K C_k^{(1)} \hat{s}(t)_{Q_k}^{(1)}. \quad (2.10)$$

$C_k$  represents the “backoff” factor described in [8] and resulting in partial interference cancellation. Since estimates are based upon noisy versions of the users’ signals, the probability that the estimates are incorrect increases. Relying completely on the estimate in cases of low signal-to-noise ratio or when a large number of users are present can degrade performance. Incorrect estimates introduce noise into the system and actually cause errors. Multiplying the estimate by a factor  $C_k$ , where  $0 \leq C_k \leq 1$ , backs off from the original estimate. In [10], a method for obtaining the optimal factor for each user in a given environment had been derived and can be summarized as

$$\tilde{r}(t)_I^{(1)} = r_I(t)^{(1)} - \hat{r}(t)_I^{(1)} \quad (2.11)$$

$$\tilde{r}(t)_Q^{(1)} = r_Q(t)^{(1)} - \hat{r}(t)_Q^{(1)}. \quad (2.12)$$

In the second stage, both parts of the residual signal are passed through a bank of matched filters that correlate with the user’s signature sequences. From there, decision statistics are formed. The DSP-based approach then formulates the residual signal  $\tilde{r}(t)$  and passes it to the second stage, and it is expressed as

$$Y_{I_k}^{(2)} = \frac{1}{T} \int_{t-T}^t \tilde{r}(t)_{I_k}^{(1)} a_k(t - \tau_k) dt \quad (2.13)$$

$$Y_{Q_k}^{(2)} = \frac{1}{T} \int_{t-T}^t \tilde{r}(t)_{Q_k}^{(1)} a_k(t - \tau_k) dt. \quad (2.14)$$

The final decision statistics are computed as the sum of the Stage 2 statistics and the Stage

1 statistics weighted by the respective backoff factor:

$$Y_{I_k}^{(3)} = C_k^{(1)} Y_{I_k}^{(1)} + Y_{I_k}^{(2)} \quad (2.15)$$

$$Y_{Q_k}^{(3)} = C_k^{(1)} Y_{Q_k}^{(1)} + Y_{Q_k}^{(2)}. \quad (2.16)$$

The bit statistic is then obtained from taking the dot product of the current and previous complex decision statistics. Hard limiting the result recovers the user's data sequence:

$$Z_{k,n} = Y_{I_k,n}^{(3)} Y_{I_k,n-1}^{(3)} + Y_{Q_k,n}^{(3)} (n) Y_{Q_k,n-1}^{(3)}. \quad (2.17)$$

### 2.3.2 FPGA–Based Implementation

Reconfigurable devices, such as FPGAs, trade speed for complexity. Therefore these devices require different types of optimizations. The DSP approach has some advantages. First, it performs the final summation of the decision statistics at the data rate rather than at the sample rate. Second, the backoff factor can be different for each user in the system. This is at the expense of the added storage and overhead associated with managing the decision statistics from the first stage. Due to the larger number of users and higher sampling rates required with practical systems, the DSP approach could become impractical since the design must be partitioned across multiple devices.

Although straightforward to implement on a DSP using C and/or assembly language, to replicate this approach in an FPGA would be impractical. Since the summation and filtering operations in the second stage are linear operations, their order can be reversed, creating the structure shown in Figure 2.2. Here, the summation now must occur at the sample rate along with the formulation of the residual signal. By combining the two operations and lumping them into the first stage, the result is the formulation of the revised version of the received signal. Thus the first stage performs interference cancellation, which produces a “cleaner”

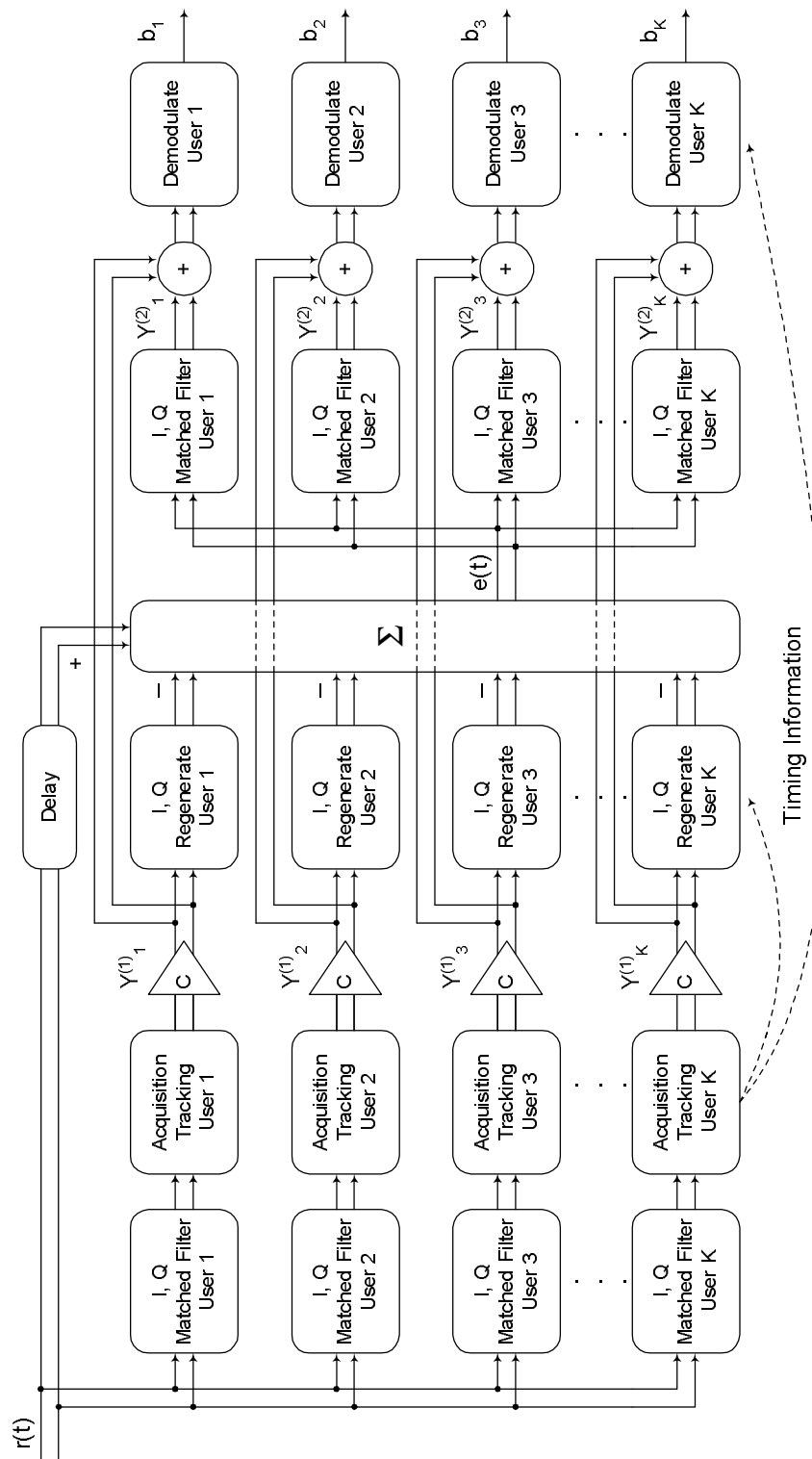


Figure 2.1: Structure of algorithm used in DSP-based implementation.

received signal, and the second stage is a conventional receiver. The entire algorithm is now more data-flow in nature, the routing requirements of the design are greatly reduced, and the two stages are almost completely independent.

Since the two stages are independent, interference cancellation can be easily integrated into existing conventional receiver hardware. The cancellation stage is placed before the conventional demodulation stage. The revised signal is fed to the latter stage instead of the actual received signal. This is an attractive option allowing quicker time to market. However, it does not allow some of the other benefits of interference cancellation to be exploited, such as cleaner acquisition off the residual signal.

## System Model

The latter portion of the system model used for the FPGA-based implementation is different. Here, the FPGA-based approach formulates a revised signal,  $\acute{r}(t)$ , and passes it to the second stage. The structure of this version of the algorithm is

$$\acute{r}(t)_{I_k}^{(1)} = C_k r_I(t)^{(1)} + \hat{r}(t)_{I_k}^{(1)} \quad (2.18)$$

$$\acute{r}(t)_{Q_k}^{(1)} = C_k r_Q(t)^{(1)} + \hat{r}(t)_{Q_k}^{(1)}. \quad (2.19)$$

In the second stage, both parts of the revised signal are passed through a bank of matched filters that correlate with the user's signature sequences. From there, decision statistics are formed:

$$Y_{I_k}^{(2)} = \frac{1}{T} \int_{t-T}^t \acute{r}(t)_{I_k}^{(1)} a_k(t - \tau_k) dt \quad (2.20)$$

$$Y_{Q_k}^{(2)} = \frac{1}{T} \int_{t-T}^t \acute{r}(t)_{Q_k}^{(1)} a_k(t - \tau_k) dt. \quad (2.21)$$

The Stage 2 decision statistics are then used to directly recover the users' data sequences:

$$Z_{k,n} = Y_{I_{k,n}}^{(2)} Y_{I_{k,n-1}}^{(2)} + Y_{Q_{k,n}}^{(2)} (n) Y_{Q_{k,n-1}}^{(2)}. \quad (2.22)$$

### 2.3.3 Computational Requirements

Although not overly complex, the amount of processing required for non-coherent detection of DPSK DS/CDMA using parallel interference cancellation is large when considering issues associated with a practical system. In [9], the complexity per bit of the DSP-based implementation is approximated to be  $O(SKNN_s)$  where  $S$  is the number of stages,  $K$  is the number of users in the system,  $N$  is the spreading gain, and  $N_s$  is the samples per chip. Here, each operation assumes one multiply/accumulate instruction. The complexity is obviously dominated by the filter banks in each stage.

## 2.4 Summary

In summary, through joint demodulation of users in the system, the performance of the system can be increased. Partial parallel interference cancellation is the detection scheme employed by the FPGA- and DSP-implementations. For both cases, a system model was described. The FPGA-based approach attempts to minimize the interconnections between the cancellation and conventional demodulation stages of the receiver. This allows a more data-flow oriented implementation with fewer interconnections consuming routing resources. Dependency between the two stages is also reduced through the use of the revised signal.

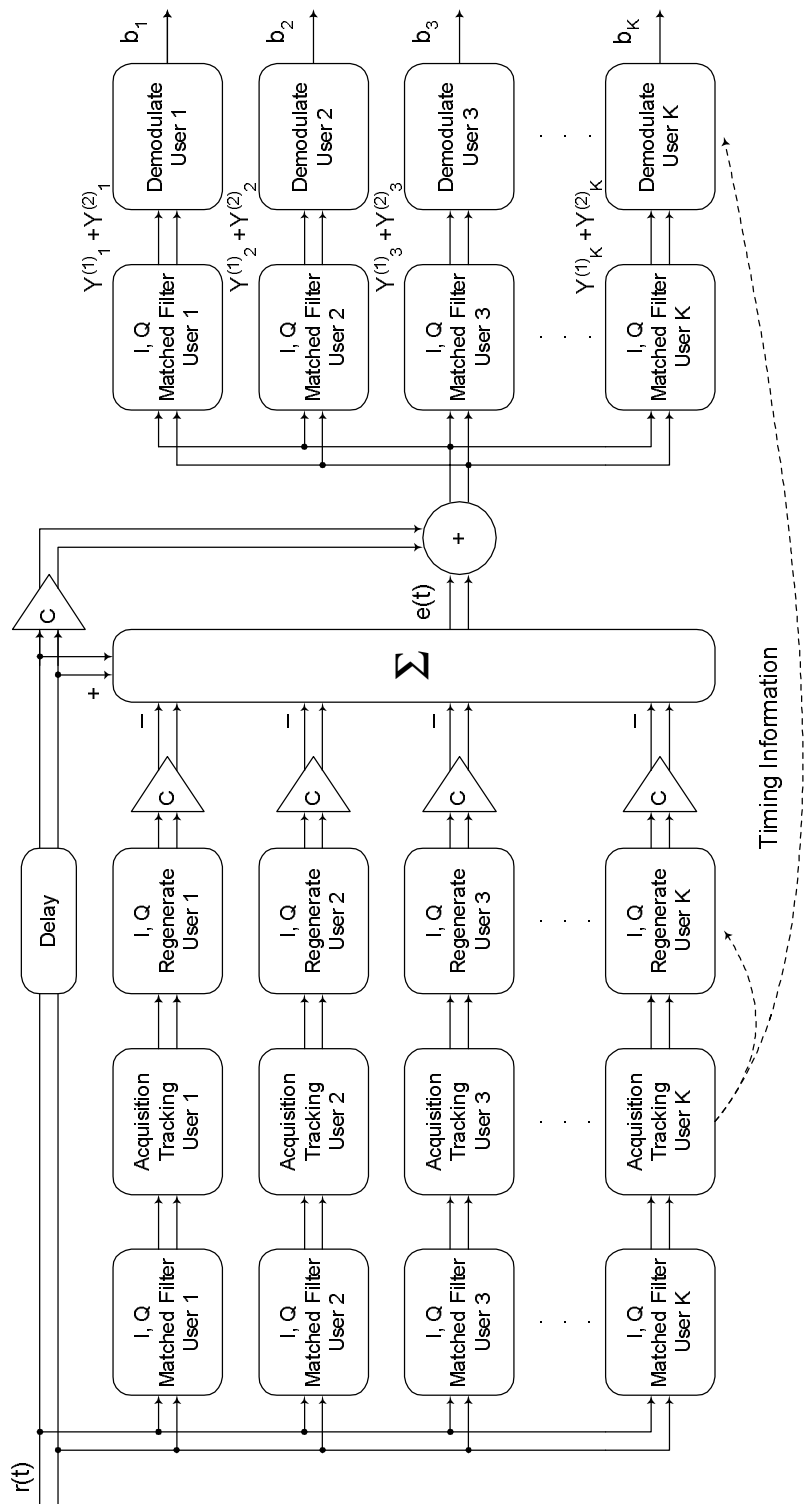


Figure 2.2: Structure of algorithm used in FPGA-based implementation.

# Chapter 3

## Configurable Computing

The design of the receiver heavily relies on the use of configurable elements organized into a processing pipeline, which is akin to the assembly line in a factory. Each element in the line performs a specialized task quickly and efficiently. The necessary pipeline elements are then arranged in a sequential fashion to complete the specified task. Pipelining tends to have a rigid structure that reduces flexibility. By incorporating the right amount of flexibility into each element, performance and configurability can be balanced to create a high performance processing engine suitable for many applications.

With the processing pipeline, the amount of delay each element requires to complete its task is fixed. Since elements operate concurrently, the entire pipeline is only as fast as the slowest element. Although the final result experiences a fixed delay equal to the sum of the latencies of elements involved, the maximum sustainable throughput of the system is dictated by only the slowest task.

General purpose processors must sequentially perform each task, thus maximum throughput decreases as the number of tasks required increases. In addition, the throughput of the design can only be speculated upon until the system is implemented. Though the algorithm can be divided among multiple processors, the overhead and complexity added by partitioning can

render the design impossible or nontrivial at best.

The configurable processing pipeline is utilized in many commercial applications. For example, Datacube Inc. has a product that utilizes a 40 MHz pipeline to perform 10,000 MIPS for image processing applications [11]. In comparison, DSPs from the TI TMS32C5x series range from between 40–100 MIPS per device [12].

### 3.1 Field Programmable Gate Arrays

For this design, the receiver processing pipeline was partitioned over a large number of programmable logic devices known as field programmable gate arrays (FPGAs). An FPGA typically consists of a large matrix of small programmable elements composed of synchronous and asynchronous logic resources. In the Xilinx family, these are called configurable logic blocks (CLBs).

In the XC4028EX-series of Xilinx FPGAs, each CLB contains two flip-flops for synchronous storage and two four-input lookup tables (LUT) along with one three-input LUT to perform combinatorial functions. A detailed description of the functionality is available in [13], and the internal structure is shown in Figure 3.1.

Surrounding the mesh of CLBs is a ring of input/output buffers, or IOBs, which provide an interface to the outside. Each IOB contains a flip-flop and bidirectional tri-state driver capability for each pin along with configurable pull-ups and pull-downs.

Another specialized structure of note is the clock buffers. In the XC4028EX series, there are eight clock buffers, two at each corner, which are used for dedicated clock lines. Clocks coming into the chip are passed through these low-skew clock buffers and distributed to all synchronous elements on the chip. This enables the design to incur the lowest amount of clock skew possible.

Xilinx FPGAs also have the ability to convert CLBs to internal RAM, which is an efficient

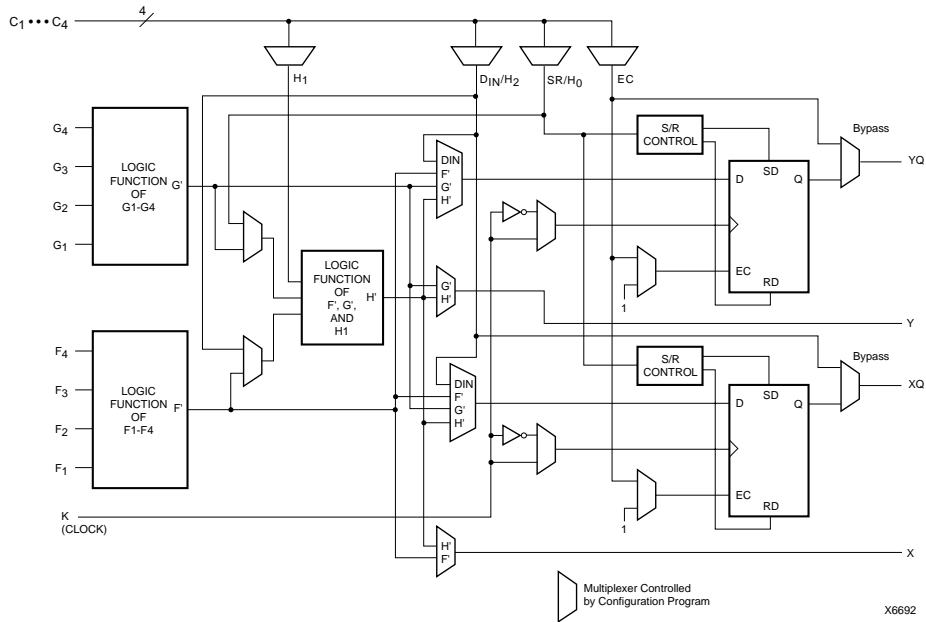


Figure 3.1: Internal structure of Xilinx 4000 series CLB [13].

way of storing data within the FPGA. Figure 3.1 shows that each CLB contains only two flip-flops. By converting to RAM, each CLB can hold the equivalent to a 16x2 bit RAM, thus a sixteen-fold reduction in storage area is achieved. Using a synchronous RAM is equivalent to an addressable flip-flop. This method was used throughout the receiver to store local information for each user.

### 3.1.1 FPGA Design Process

Commercial tools such as Matlab, COSSAP, and SPW all provide a means for modeling the implementation of a communication system at a high level. These tools can verify the behavior and predict the performance of a proposed system accounting for implementation specific issues such as finite precision. Designs specified at the higher-level, along with test vectors, can be readily translated to lower level language representations, such C or VHDL, that are readily usable in the physical design process.

Design entry for FPGAs is usually done via a text-based description or by schematic entry. Text-based entry, using common Hardware Description Languages (HDLs), has become popular in recent years due to its flexibility and technology independence. The most common HDLs used in FPGA design entry are VHDL and Verilog. The designs used in this project were written in VHDL, which allows the user to create an entity that performs a specific task. The functionality of the entity is defined by supplying an architectural description. Two main approaches to defining an architecture exist. The first method uses a structural model that instantiates other entities and specifies their interconnection almost like a schematic. This approach allows for more control over the physical implementation resulting in faster, more compact designs; however, even small changes may require a total redesign of the architecture. The second method is behavioral description. The functionality of the entity is specified at a higher, more generic level. This provides little or no insight into the physical implementation of the entity but provides very flexible code that can be modified quickly.

The approach used here is a hybrid of both types. A structural approach was used at the higher levels to maintain the hierarchy of the design. It was also used for the larger, more repetitive designs such as the correlators. Behavioral methods were used to describe a majority of the lower-level functionality. This alleviates some of the design complexity when specifying constructs such as state machines and counters. Utilities, such as Xilinx's LOGIblox [13], allow for creating technology-specific resources, like internal memory, or commonly used logical structures, such as adders, that are optimized for the Xilinx family of FPGAs. This utility was used to create portions of logic throughout the design.

### **3.1.2 FPGA Synthesis & Implementation**

Once the high-level design is validated, the process of physically implementing the design begins. First the design is synthesized using Synopsys FPGA Express software [14]. Synthesis takes the behavioral and structural description of the design and formulates the physical layout of the circuit. The hierarchy is flattened and the result is optimized for the targeted

device. The performance of synthesizers to infer logic that is both fast and spatially efficient from a behavioral description has greatly improved within the last few years.

Once the design has been synthesized, a physical netlist of the design is produced consisting of primitive structures that are technology-specific. For this effort, designs were targeted to Xilinx 4028EX FPGAs. The resulting netlist was then used to produce the configuration bitstream that programs the FPGA device. This operation was performed by the Design Manager from the Xilinx M1.4 toolset [15]. This task is completely automated, and it performs further logical optimization of the design, constraint-based partitioning, placement, and routing of the design onto the chip. Usually a clock speed is specified to guide the implementation process. For this design, the maximum clock speed targeted typically ranged between 10–20 MHz depending on the complexity of the design. Higher clocks speeds for complex designs require a great deal of CPU time to find an adequate solution. The resulting implementation is summarized and its timing performance predicted from the software supplied by Xilinx. The size of the bitfiles created is fixed for a given family of FPGAs. For the XC4028EX FPGA, it is 82 kb (83598 bytes). With a minimum configuration clock period of 100 ns [13], the FPGA can be reconfigured in no less than 66.8 msec.

### 3.1.3 Run-Time Reconfiguration

Run-time reconfiguration is the total or partial reconfiguration of a device to complete a processing task, greatly enhancing flexibility by allowing algorithms to be partitioned temporally as well as spatially. Run-time reconfiguration allows for much larger, more complicated algorithms that span fewer devices to be implemented. This increases the computational density of the system assuming quick reconfiguration time and adequate temporary storage.

Most devices on the market do not support partial reconfiguration, meaning the entire FPGA must be reprogrammed. Any data stored in the device is lost if not stored externally. Whether a device can be run-time reconfigured is dependent on factors such as the reconfiguration time of the device, the amount of external storage available, and the acceptable

level of latency in the system.

As shown earlier, the XC4028EX requires at least 66.8 msec to reconfigure. In wideband DS/CDMA, the high sample rates involved preclude the use of conventional FPGAs in a run-time fashion. At a sample rate of 2 MHz, 133,000 samples would need to be buffered while the device was being reconfigured. This quickly approaches the level of intolerable latency in a communications system. Thus in a software radio comprised of conventional FPGAs, the receiver structure would need to be configured upon initialization and remain static. In other words, the system would not be dependent on reconfiguration of the FPGA device to complete the signal processing task.

Other devices are better suited for use in a run-time reconfigurable system [16]. The Colt, an experimental run-time reconfigurable device, utilizes self-directing streams that allocate and configure device resources to accomplish a given task. The streams contain a program header that contains the necessary configuration information for each resource required. The header is formed by stacking the programming information in the order in which the resources are encountered as the stream tunnels through the device. The prototype Colt contains six data ports, a crossbar switch, a multiplier unit, and a 16x16 mesh of sixteen bit arithmetic logic units (ALU). Each ALU is capable of performing any logical or arithmetic function of the two sixteen bit operands. The Stallion chip is a scaled up version of the Colt with more multiplier units and ALUs.

## **3.2 GigaOps G900 Configurable Computing Platform**

The PCI-based GigaOps G900 provides a flexible platform for rapidly prototyping custom computing solutions [17]. It provides communication between PC applications and FPGA designs and numerous clocking resources. Associated with any application designed for the GigaOps is a host control application that loads, configures, and interacts with the G900.

The G900 board contains sockets for removable modules known as XMODs. The G900

supports a total of four XMODs per stack allowing up to sixteen XMODs to be used. XMODs can contain various types of computing resources, memories, and I/O resources. The XMODs used here each contain two Xilinx XC4028EX-3HQ208 FPGAs (28k equivalent gates) plus 8 MB of DRAM and 256k of SRAM. A diagram of the G900 hardware is shown in Figure 3.2.

The G900 board contains a large bus segmented into six sixteen bit primary busses, two eight bit busses, eight dedicated low-skew clock lines, and numerous auxiliary lines [17]. With each of the primary busses, there is an associated clock and auxiliary line. All busses are made available to each XMOD stack through its socket connector.

The two eight bit busses, designated X0 (HBUS0) and X1 (HBUS1), are used for communicating with the Host PC via the HBUS protocol and thus are reserved solely for this purpose. The remaining six sixteen bit busses can be allocated as the designer sees fit. The busses are designated X2 (YBUS0), X3 (YBUS1), X4 (XBUS0), X5 (XBUS1), X6 (XBUS2), and X7 (XBUS3). The names in parenthesis were used in earlier versions of the GigaOps product line.

The G900 has limited programmable bussing capability. The programmability is implemented using switches that either make or break the continuity of the bus. On the main G900 board, busses X6 and X7 are switched at each of the four XMOD sockets. These switches can be configured by the host PC.

On each XMOD, the X FPGA has access to busses X4, X5, X6, and X7. The Y FPGA has access to busses X0, X1, X2, and X3; therefore, host communication must originate from it. All busses are passed up to the next XMOD in the stack. In addition, the two FPGAs share thirteen lines reserved for dedicated interconnect plus twenty-six switched lines. For busses X4, X5, X6, and X7 only, switches are located at the point where they enter the XMOD from the bottom. The XMOD can be configured to either isolate or pass each of the busses from that point on through the rest of the stack. This allows for greater flexibility and interconnect within the XMOD stack. All XMOD switches are controlled by the Y FPGA.

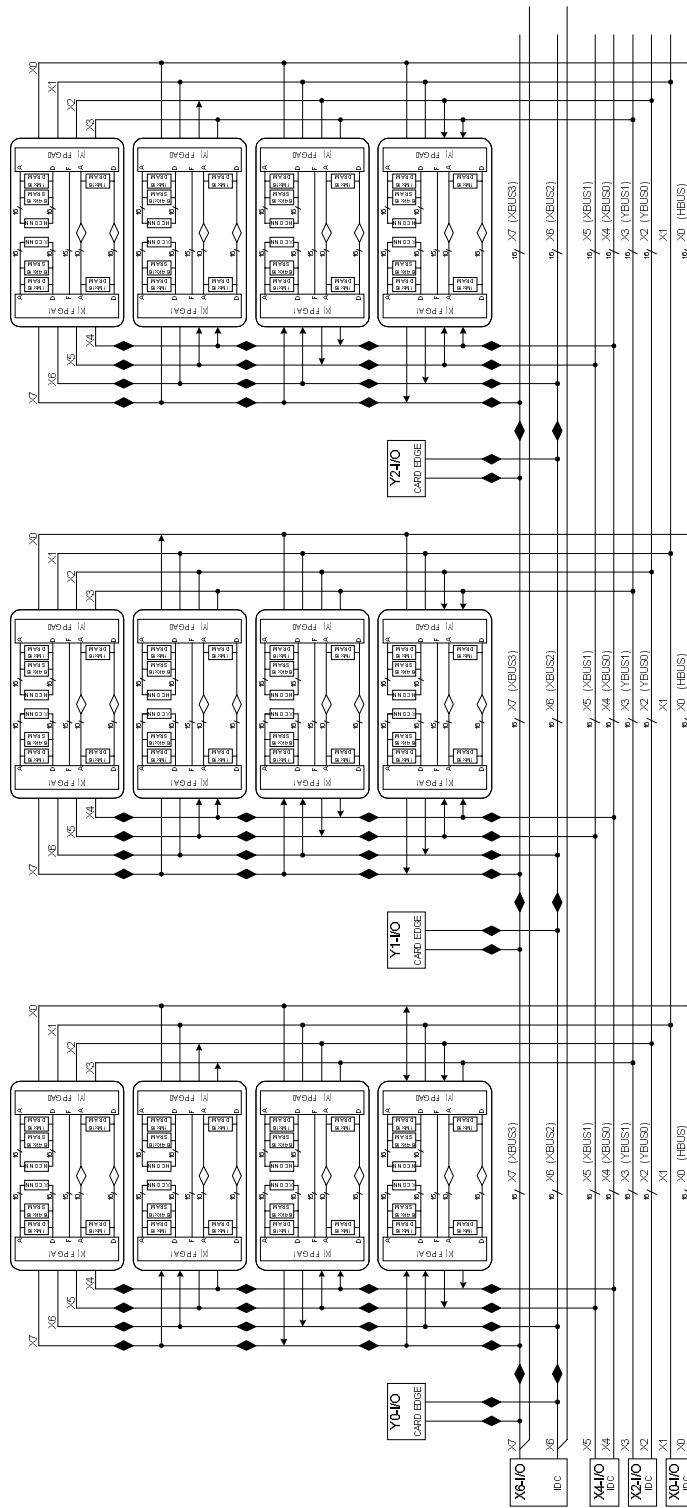


Figure 3.2: Diagram of the GigaOps G900 configurable computing platform.

### 3.3 Motivation for FPGA-Based Implementation

Given the computational complexity of the interference cancellation structures presented, it is obvious that any practical system will not be implemented on a single chip in the near future. Thus, three options exist for the receiver architecture: a hybrid DSP/FPGA receiver, a DSP-only receiver, and an FPGA-only receiver.

DSPs provide the most flexibility in algorithm implementation plus quick design time, but they have high power consumption and limited I/O facilities. Multi-DSP computing platforms are quite prevalent and their design environments are impressive; however, it was found that the proprietary inter-processor communication schemes often substantially complicate the design. This further reduces scalability of the system when dealing with the already limited I/O bandwidth and high sample rates associated with wideband DS/CDMA.

FPGAs provide more I/O and allow higher processing rates but impose limits on the size, complexity, and flexibility of the algorithms implemented on them. Their spatial inefficiency and power consumption also preclude their use in high volume production systems. Multi-FPGA based computing platforms often provide an open scheme of inter-processor communication that can be readily molded to accommodate a wide range of computing architectures. Numerous works [18, 19] also demonstrate the usefulness of FPGAs when implementing algorithms that exhibit the following properties:

- High levels of parallelism,
- Easy pipelining with simple operations in a fixed sequence,
- Operation on low resolution data (10–12 bits),
- Operation on large data sets,
- Simple control requirements.

All of the properties listed are characteristic of parallel interference cancellation when applied to differentially coherent DS/CDMA.

By exploiting parallelism, pipelining, and low resolution data, all characteristic of wideband DS/CDMA, FPGAs are better suited for prototyping advanced receiver structures than DSPs. An FPGA-only solution was the clear choice for the prototype multiuser receiver. A hybrid scheme could provide significant advances in performance by allowing more robust tracking algorithms, but it creates a more complex design environment. Future production systems employing interference cancellation are likely to be solely ASIC-based due to the high processing rates involved and power concerns, thus an FPGA-only implementation provides better insight to the design of practical DS/CDMA multiuser receivers.

### **3.4 Stream-Based Modular Design Concept**

FPGAs are more data flow oriented, thus algorithm implementation requires a different approach than that used with DSPs. For the multiuser receiver application, a stream-based modular design concept was utilized. Stream-based modular design provides a means for exploiting the processing power attainable through deep pipelining while still maintaining some degree of flexibility. The algorithm to be implemented is first represented as a data flow graph. This is then decomposed into smaller computational primitives called modules. Each module performs a unique subset of the overall processing.

Each module is designed to efficiently perform its task. Around the processing portion, logic is added to allow the module to handle stream input and output. A stream is comprised of both programming information and data to be processed. Although static, the functionality of each module can be modified. When a module encounters programming information, the necessary configuration data is extracted and stored locally, and the module's operation is changed accordingly. The module is free to then modify outgoing programming information allowing intermodule communication in a single direction. This provides parameter pass-

ing, similar to that available in high level programming languages, and the modular design philosophy allows for code reuse.

With this approach, configurability is increased, and global complexity and module synchronization requirements are reduced. Modules can be developed and tested individually and are better suited for a reconfigurable environment due to the common interface. However, this is at the expense of increased size and complexity of each module. The data path through the system is widened by two bits: a) the PROGRAM bit, b) and the VALID bit. This allows classification of the information coming down the pipe as either program or data and whether it should be ignored or not.

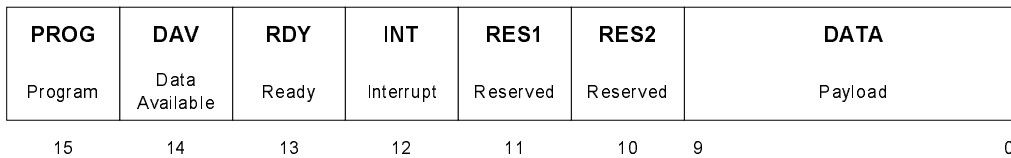


Figure 3.3: Adopted stream packet format for the multiuser receiver.

The G900 computing platform contains numerous sixteen bit busses. The data width of the pipeline was restricted to ten bits due to the limited intra-XMOD connectivity available. Although the streams are effectively only twelve bits wide, they were mapped to the full sixteen bit bus width, as shown in Figure 3.3, to maintain some degree of scalability. Bits 15 and 14 hold PROGRAM and VALID while bits 9 through 0 contain the data payload. The other signals are reserved for future stream-based applications.

### 3.4.1 Module Structure

Each module contains a processing pipeline, a bypass pipeline, configuration storage, and a simple state machine, as illustrated in Figure 3.4. The processing pipeline performs the desired operation on the valid data entering the module. The bypass pipeline is present to ensure the integrity of the programming information as it passes through the module. If

the bypass pipeline were not there, programming information would be mutilated by the processing pipeline. The bypass must contain as many stages as the processing pipeline to maintain the equal delays, which increases the size of each module.

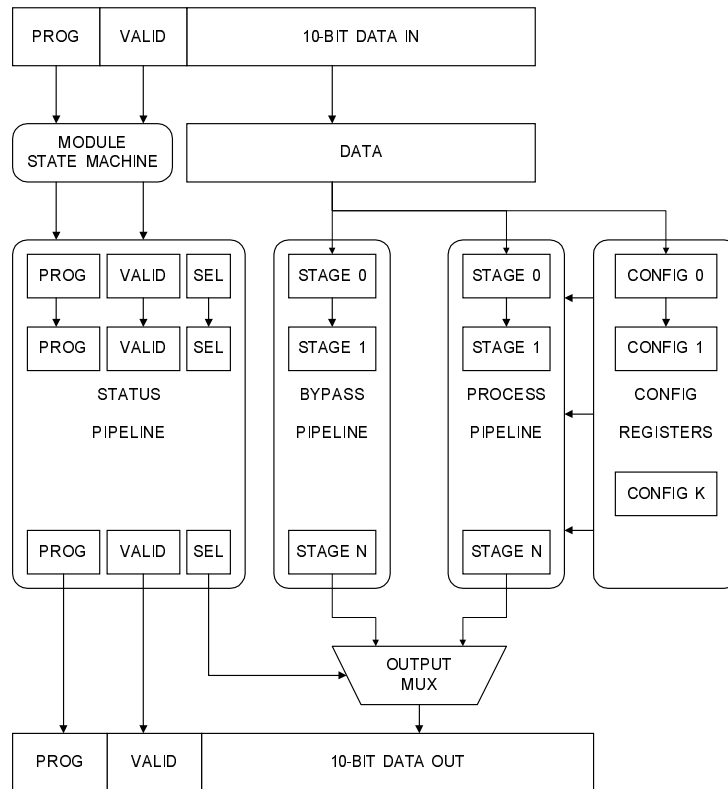


Figure 3.4: Internal structure of module that operates on streams.

The configuration registers store information from the programming header, which is used to modify the processing pipeline. The state machine controls the destination of the modules' input stream and the source of its output stream.

### 3.4.2 Module State Machine

Within each module, there exists a simple sequential state machine that controls the operation of the module. The state diagram of a simplified state machine for stream processing control is shown in Figure 3.5.

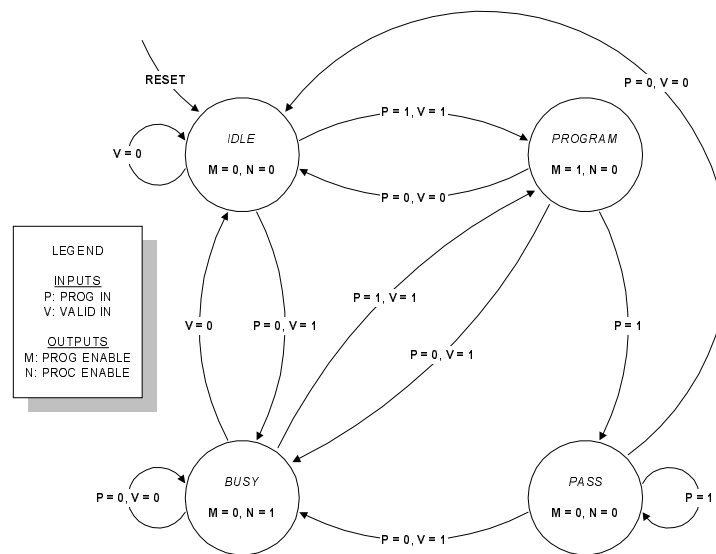


Figure 3.5: Diagram of simplified module state machine.

A module can be in one of four possible states: IDLE, BUSY, PROGRAM, and PASS. The inputs to the state machine are the PROGRAM and VALID lines from the incoming word.

When valid non-program data is encountered, the module enters the BUSY state. During this state, the processing pipeline is enabled and it is selected as the source of the module output. The processing pipeline may dictate the validity and contents of its output.

When valid program information is encountered, the module enters the PROGRAM state. During this state, programming information is latched into the configuration registers and the output is taken from the bypass pipeline. The module can be specified to handle any number of programming words. By marking outgoing program information invalid, the module can “strip” off its unique information from the program header similar to a stack. In this case, the program header is a composite of all the programming information unique to each module. Although this is more robust, it complicates the design and is less efficient when the same program information is reused throughout the system. Here the PROGRAM bit is unchanged, making the information global in nature. All modules use the same program words for each user and store locally only what they need.

During the IDLE and PASS states, the module output is taken from the bypass pipeline and the outgoing PROGRAM and VALID lines are not manipulated. The module enters the IDLE state when any invalid information is encountered. The PASS state is needed to allow the module to latch in a subset of the information within the programming header.

### **3.5 Summary**

This chapter has presented the case for the use of parallel processing for communications applications. Many of the most common communications algorithms are readily adaptable to pipeline processing since they are inherently data-flow oriented. The computing platform used in this effort was outlined along with the stream-based modular design concept. In many cases, FPGAs are good for prototyping communications circuits because many are then converted to ASICs for production. FPGAs provide an excellent stepping stone. Generic module design, operation, and organization were illustrated in detail.

# Chapter 4

## Multiuser Receiver Implementation

This chapter gives a detailed description of the operation and implementation of the FPGA-based multiuser receiver. The first section highlights the operational aspects of the receiver, lists parameters of the prototype system, and details the hardware used in the receiver testbed. Next, the design of the receiver, based upon the stream-based modular design concept outlined in Chapter 3, is described. Each module in the design is illustrated. The physical mapping of the receiver structure onto the computing platform and the resulting synthesis of the many FPGA designs used are presented.

### 4.1 Overview

The multiuser receiver is organized into two stages. The first stage accepts the complex envelope of the received signal generated by the digital downconverter. It consists of a matched filter bank, independent acquisition and tracking mechanisms for each user, estimation and regeneration of each user, and formulation of the *estimated* received signal. From this, a *revised* version of the received signal is computed based upon the desired cancellation level and is then passed to the second stage.

The second stage is a conventional matched filter DS/CDMA multiuser receiver. It consists of a bank of matched filters and differential BPSK demodulators. This approach attempts to keep the two stages separate and independent. Doing so allows interference cancellation to be easily integrated into existing conventional receivers.

Although the algorithm for PIC requires no complex computations, the receiver structure size increases dramatically when a significant number of users need to be processed because each possible user in the system must be estimated and regenerated in parallel. The resulting estimates must then be added together to form the composite estimated received signal. This result is then piped to the bank of receivers in the second stage. The routing requirements of the receiver increase linearly with the number of users in the system. Therefore, designing for scalability is critical.

This approach trades higher processing speeds for less silicon area and signal routing resources. Users are processed serially through a pipeline of processing modules at a rate at least  $K$  times greater than the incoming sample rate where  $K$  represents the number of users in the system. Programming information is merged into the data path enabling the modules to be configured to process each user. The programming header contains items such as the user's ID, timing information, acquisition status, filter configuration, and PN code. In this implementation, the programming information is global in nature and not module specific. In addition, each module may locally store user specific variables and address them based upon the user's ID.

## 4.2 Prototype System Specifications

The prototype communications system was initially designed to handle four users but can be scaled up to sixteen users by reducing the data rate proportionally. For four users, the data rate is 31.25 kbit/s per user. Each is modulated using differential BPSK and conventional DS/SS, eliminating the need for carrier synchronization. The PN sequences used in the

system are Gold codes of length fifteen extended to sixteen, resulting in a chip rate of 500 kHz [20]. The additional chip eases the implementation of PIC when normalizing the amplitude estimate obtained from the correlators. The transmitters utilize rectangular pulse shaping making the RF bandwidth of each user 1 MHz. The carrier frequency used is 2050 MHz.

### 4.3 FPGA–Based Multiuser Receiver Testbed

The FPGA–based receiver testbed consists of an RF front–end, a Harris 50214 digital down-converter evaluation board, the PCI based GigaOps G900 computing platform, and a host PC. The system is illustrated in Figure 4.1.

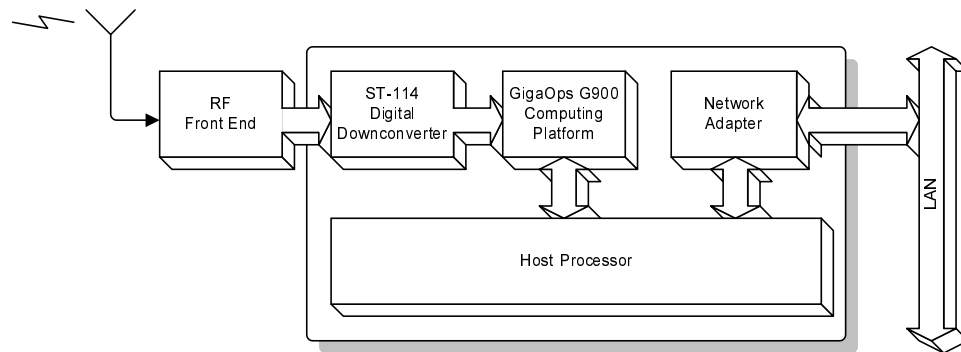


Figure 4.1: Diagram of the FPGA–based receiver testbed.

#### RF Front–End

The RF front–end is comprised of modular components including the RF filter and amplifier, mixer and local oscillator, and IF filter and amplifier. From the antenna, the incoming signal is mixed down from 2050 MHz to an IF below 100 MHz. The IF signal is then fed into the IF input of the Harris 50214 evaluation board.

### **Harris 50214 Digital Downconverter**

The Harris 50214 Programmable Digital Downconverter chip [21] is made available by the Sigtek ST-114 Evaluation Board [22]. The 68 MHz IF input is sub-sampled at 16 MHz by an onboard ten bit ADC. The Windows 95-based control software provides full accessibility to the internal functions of the Harris 50214. For this application, the 50214 mixes the 16 MHz sampled input to baseband using a complex mixer. The signal is then decimated to 2 MHz using the CIC decimation filter and a signal half-band filter [23]. The output of the DDC is two sixteen bit channels representing the in-phase and quadrature portion of the complex envelope of the downconverted received signal. These channels are fed directly onto two busses on the GigaOps G900 board via ribbon cable.

### **GigaOps G900 Configurable Computing Platform**

The PCI-based GigaOps G900 provides a flexible platform for the rapidly prototyping custom computing solutions. The specifics of the platform were presented in Chapter 2. The incoming data for the  $I$  and  $Q$  channels are connected to the X6 and X7 busses. Due to hardware constraints, only the top fourteen bits of each channel were routed to the G900 via ribbon cable. Of these fourteen, only the top ten are utilized in the receiver. A diagram of the G900 hardware is shown in Figure 4.10.

## **4.4 Algorithm Implementation**

The FPGA-based multiuser receiver was implemented using the stream-based modular design principle detailed in Chapter 3. The resulting design utilizes nine types of modules. These include (a) the Input module, (b) the Matched Filter Correlator module, (c) the Acquisition/Tracking module, (d) the Regenerate module, (e) the Combine module, (f) the Revised module, (g) the Buffer module, (h) the Differential Demodulator module, and (i)

the Output module. The modules were interconnected as illustrated in Figure 4.2. The in-phase and quadrature channels form the two primary processing paths through the receiver. Due to the nature of PIC applied to differentially coherent DS/CDMA, the two streams are identical and independent of one another for a majority of the processing.

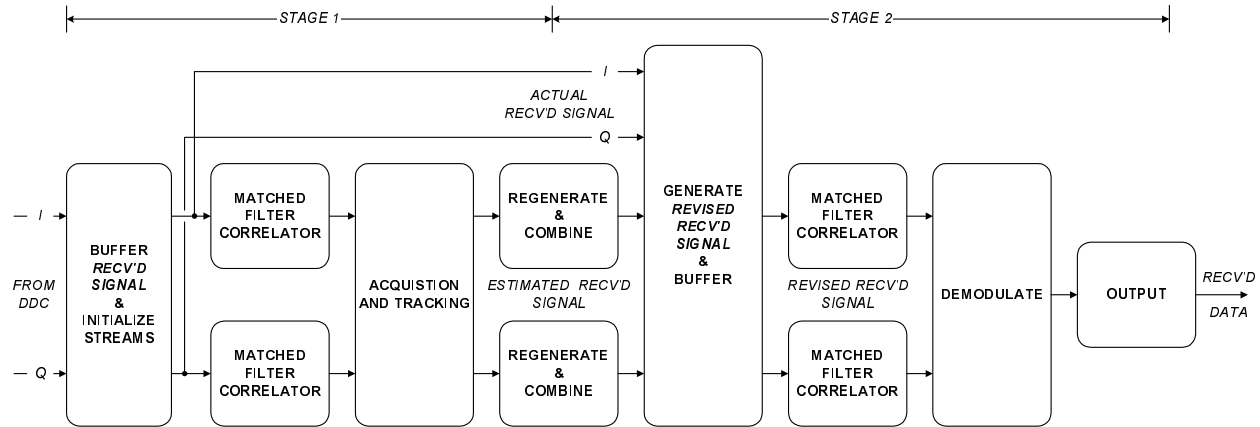


Figure 4.2: Diagram of the modularized receiver structure.

In the first stage, an Input module lies at the front of each of the  $I$  and  $Q$  processing pipelines. It acts as a stream controller performing two concurrent functions. The first is to segment the newly downconverted received signal into blocks of 1024 samples and store the current block in external memory. The second is to read the previous block from memory, prefix it with programming information to form a stream, then send the stream down the processing pipeline.

The Matched Filter Correlator module filters the *actual* received signal by the current user's spreading code. Since the module contains a common delay line, it must be refreshed to the correct state before processing each user. This requires an additional 63 clock prior to processing a user to load the state from internal RAM.

The Acquisition/Tracking module is the only point in the first stage where the  $I$  and  $Q$  streams converge. The purpose of this module is to update the programming header passing through with the correct timing and acquisition status for the current user. The processing portion of the module approximates the magnitude of the signal. From this, it attempts to as-

certain the user's correct symbol timing and track it using an averaged early-late scheme [24]. When a long-term average of the on-time sample reaches a specified threshold, the user is deemed acquired and allowed to be canceled.

If a user is acquired, the Regeneration module, found in both the  $I$  and  $Q$  branches, recreates their signal based upon sampled amplitude estimates. The Combine module then sums up each users' regenerated signals to form an estimate of the received signal. The revised module formulates the *revised* received signal by combining a delayed version of the *actual* received signal with its estimate. The amount of cancellation can be adjustable based upon the reliability of the estimate. The backoff factors currently supported are 0.00, 0.50, 0.75, and 1.00. The nature of the structure only allows a common backoff factor to be applied across all users.

In the second stage, the Buffer module first stores the *revised* received signal, then it creates streams from the revised data for each user and injects the stream into the corresponding processing pipeline. Again, the  $I$  and  $Q$  branches are identical and independent until the Demodulator module. After correlation, the filtered  $I$  and  $Q$  streams enter the Demodulator module. There, the users' data bits are decoded and passed to the Output module.

The Output module is responsible for collecting each user's incoming data bits and interfacing with the Host PC. The Output module contains FIFOs for each user. Incoming bits are grouped into sixteen bit words and placed into the correct FIFO. The PC host monitors the status of each FIFO and retrieves the data when the desired user's FIFO is not empty.

#### 4.4.1 Input Module

An Input module lies at the front of both the  $I$  and  $Q$  processing pipelines. It performs two concurrent functions. The first is to segment the newly downconverted received signal into blocks of  $N_b$  samples and store the current block in external memory. The second is to read the previous block from memory, prefix it with the desired user's programming information

to form a stream, then send the stream down the processing pipeline. The latter function is performed  $K$  times where  $K$  is the total number of users supported by the system. The processing clock must be greater than  $K$  times the sampling rate coming from the DDC.

The prototype system utilizes an input sample rate of 2 MHz and a block size of 1024 samples; thus, a new block of data becomes available every 512  $\mu$ secs. The timing scheme for the input module is illustrated in Figure 4.3. Within this interval of time, the previous block must be processed for each of the  $K$  users in the system. Running the computing platform at 10 MHz allows a maximum of 5120 words to be processed per input block. Each user's stream is 1103 words long and consists of the following: a ten-word preamble, a six-word programming header, a sixty-three-word filter initialization segment, and 1024 data samples from the previous input block. With  $K=4$  users, the total processing time required is 4412 cycles. This leaves 708 cycles of idle time where the compute engine is awaiting new data.

### **Programming Header**

Although sixteen words are reserved for programming information, only the last six are used. The first ten are marked invalid and act as a preamble. This length must be greater than the maximum processing latency of any module in the system. If this were not present, a new stream entering a module could alter the modules operation before all valid data from the previous stream is processed. This would result in corrupted data at the end of each output stream.

The composition of the programming header is shown in Figure 4.4. It contains six ten bit programming words unique to each user. This data is stored in internal ROM. It includes: (a) the user ID, (b) sample index, (c) acquisition status, (d) if this user is the last user to be processed, (e) correlator configuration, (f) and PN code. Although the organization is somewhat sparse to ease coding, the header supports up to sixteen users and has plenty of space for additional information.

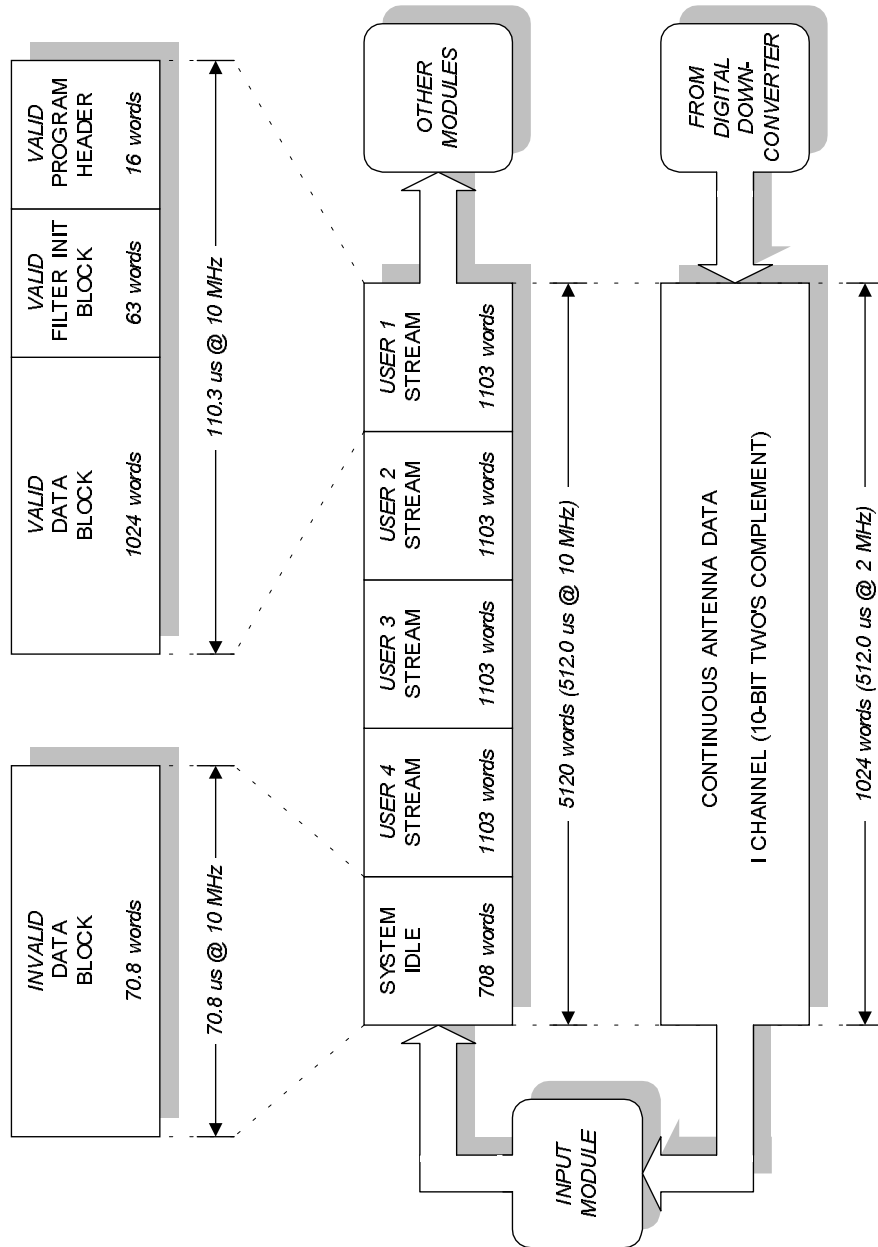


Figure 4.3: Organization of the Input module stream output.

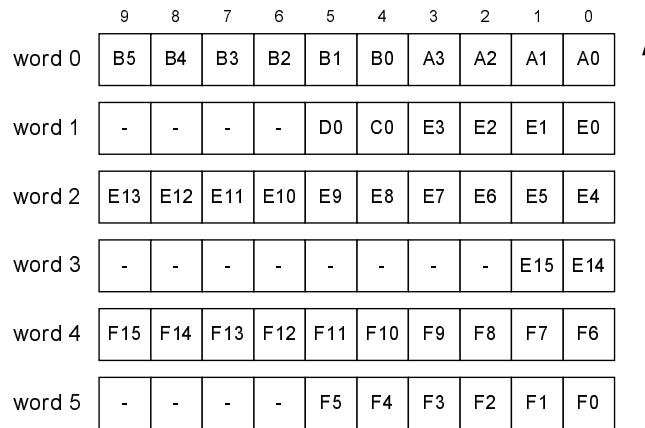


Figure 4.4: Diagram of stream programming header.

## Operating Efficiency

Since processing along the  $I$  and  $Q$  channels is identical, they share the same efficiency. The bandwidth of the processing engine (per channel) is divided among the types of data passed through the system as shown in Table 4.1. The processing overhead, consisting of the preamble, programming, and filter initialization words, utilizes only 6% of the available bandwidth. Actual data processing utilizes 80% of the bandwidth, while 14% is idle.

The bandwidth efficiency,  $\eta_b$ , provides insight into how much of the system resources are spent completing a given task. For this system, the engine is idle only 702 cycles out of the 5120 available, resulting in an  $\eta_b$  of 86.17%.

The computational efficiency,  $\eta_c$ , provides insight into how much time is spent processing actual data versus total processing time. This is irrespective of the clock rate chosen but indicative of the overhead introduced by stream processing. Since the data accounts for 1024 words out of 1103 words processed per user,  $\eta_c$  is 92.8% for this system.

Table 4.1: Utilization of processing bandwidth.

	Words Per User	Words Per Processing Block	Utilization
Preamble	10	40	0.78%
Programming Words	6	24	0.47%
Filter Initialization	63	252	4.92%
Input Data	1024	4096	80.00%
Idle	n/a	708	13.83%
Total	1103	5120	100.00%

#### 4.4.2 Matched Filter Correlator Module

The Matched Filter Correlator module is responsible for correlating the valid data portion of the incoming stream using the coefficients supplied by the programming header. The module also performs matched filtering at the chip-level since rectangular pulse-shaping is used. Since both correlation and integration are linear operations, their order can be interchanged. Here the received signal is correlated first then passed through the pulse-shaping filter. This yields a more efficient structure than combining the operations into one larger filter.

The module first performs the correlation operation

$$y(t) = \sum_{n=0}^{NN_s-1} c_n x_n(t) \quad (4.1)$$

where  $N$  is the number of chips in the PN sequence,  $N_s$  is the number of samples per chip,  $x_n$  is the delay line, and  $c_n$  is a vector of tap weights. The operation is simplified considerably since  $c_n$  is a sparse vector with non-zero weights spaced every  $N_s$  elements apart. The elements assume values of  $+1$  or  $-1$  corresponding to the time-reversed version of the desired user's PN code. The result  $y(t)$  is then fed to the integrator

$$z(t) = \frac{1}{NN_s} \sum_{n=0}^{N_s-1} y_n(t) \quad (4.2)$$

to match filter the rectangular pulse shape of the spread signal. A scale factor of  $1/NN_s$  is used so that a correlation peak is equal to the average level of the input sequence. An estimate of the amplitude of the input signal is obtained by sampling at the peak correlation output.

The module's processing pipeline is shown in Figure 4.5. It consists of a delay line of sixty-four ten bit words and a pipelined tree of adder/subtractor units. An efficient structure for the correlation operation is created by feeding every fourth word in the delay line to the tree. These sixteen taps are combined by fifteen adder/subtractor units spaced over four pipelined stages. The stages 1, 2, 3, and 4 contain eight, four, two, and one units, respectively. A single control bit to each unit specifies its arithmetic operation. Prior to entering each unit, both operands are signed-extended to accommodate the resulting bit growth.

After the correlator tree, the resulting output is then placed through the chip pulse-shaping filter. Since rectangular pulse shaping is used, the ideal matched filter is an integrator that spans the bit period. The correlator output is passed through a two-stage integrator of length four. Prior to accumulation, each tap is sign-extended. For more complicated pulse-shaping, more complex MAC structures are required. Another possibility is to place the chip pulse shape filter into the Harris 50214's serial FIR filter.

After four stages in the correlator and two in the integrator, the resulting word size grows to sixteen bits. For interference cancellation, the correlation result is normalized by the correlator length. This yields an estimate of the user's received signal amplitude from which the user's signal is regenerated; therefore, only the top ten of the resulting sixteen bits are taken as the output of this Matched Filter Correlator module.

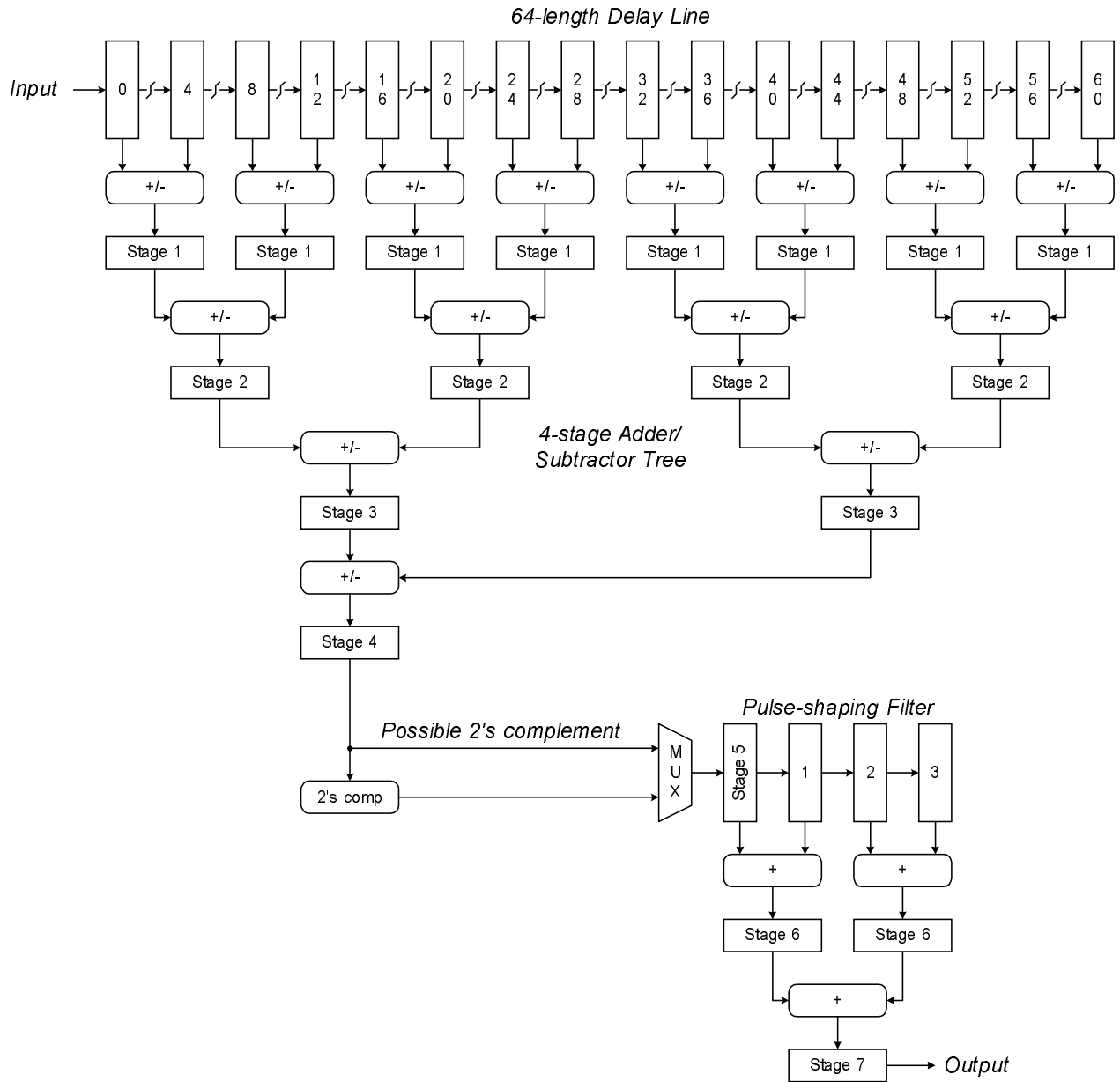


Figure 4.5: Diagram of the Matched Filter Correlator module processing pipeline.

## Correlator Delay Line Refreshing

Since a common delay line is used to filter the same data sequence multiple times, before each user is processed, the coefficients are configured and the delay line refreshed. During the processing of the last user, the last  $NN_s - 1$  input samples are stored in RAM internal to the FPGA. During the next block, for each user, these values are clocked into the delay line during the first sixty-three valid inputs. Since the resulting output during this period is unusable, it is marked invalid so it will not be processed by modules further down the pipe.

## Encoding of the Correlator Weights

The correlator weights are determined by the sixteen bit filter configuration word, which specifies whether the two paths entering each node are added or subtracted. This is complicated by limitations of the arithmetic units at each node in the tree. Let A and B be designated as the words entering the node from the left and right, respectively. Each arithmetic element within the tree can only perform the operations (A-B) or (A+B). The operation is chosen by the logic level of the ADD/SUB control line with a logic 0 denoting subtraction and a logic 1 denoting addition. The algorithm for converting the desired correlator weights to the appropriate configuration bits is straightforward. Each operand carries with it a polarity. Initially, the polarity of each tap is chosen to be the sign of the corresponding chip in the desired correlator weight vector. Every two of these taps enter one of eight arithmetic elements in the first stage of the tree. The correct operation to perform at each node in the tree and the polarity of the result are then determined by applying the rule specified in Table 4.2. This table is applied to each node in the tree beginning with the first stage. The polarity of the result is then used to determine the correct configuration of the elements in the next stage.

For the final result, a positive polarity is desired. However, some configurations will yield a final result that has a negative polarity. Therefore, a sixteenth configuration bit and an

Table 4.2: Rules for translating desired correlator weight to filter configuration bit.

A Polarity	B Polarity	Operation/Control Bit	Result Polarity
+	+	Addition (1)	+
+	-	Subtraction (0)	+
-	+	Subtraction (0)	-
-	-	Addition (1)	-

additional stage is added to select the final result or its twos complement as the final output of the correlator.

### 4.4.3 Acquisition/Tracking Module

The Acquisition/Tracking module is responsible for determining the presence of a user in the system and that user's proper timing information. This module is by far the most complex part of the receiver. The input to the module is  $Y_I(t)$  and  $Y_Q(t)$ , the results from correlating the complex received signal with the desired user's PN sequence. The module does not modify the data portion of the two streams as they pass through. Instead, the module updates the user's program header with that user's latest sample index and acquisition status.

After correlation with a desired user, the most persistent peak in the magnitude output should correspond to the user's symbol. This peak is not always the maximum peak due to interference caused by other users in the system. However, it should occur periodically every  $NN_s$  samples or so.

The input block for this receiver is set at 1024 samples. The processing gain,  $N$ , is sixteen, and the number of samples per chip,  $N_s$ , is four. Thus, there are sixteen symbols per input block. Since the symbols occur periodically, one is present within every window of sixty-four samples. The purpose of this module is to identify the exact location of the user's symbol within the symbol window. This position is designated as the *sample index* and its value ranges from zero to sixty-three. It can be represented using six bits.

The module approximates the magnitude of the correlator output, applies a symbol averaging filter to accentuate periodic peaks in the magnitude, acquires the location of the most persistent peak in the magnitude, and follows that peak as it moves.

### **Magnitude Approximator**

The processing pipeline of the module first approximates the magnitude of the complex input stream. The magnitude approximator circuit, detailed in [25], was used in order to reduce the size and complexity required. The approximator examines  $|Y_I(t)|$  and  $|Y_Q(t)|$  and determines which is smaller. The approximate magnitude of the signal is then calculated as the greater value plus half of the smaller value. The largest error in the approximation, about 12%, occurs when either  $|Y_I(t)|$  or  $|Y_Q(t)|$  is half the value of the other.

### **Chip Averaging Filter**

The chip averaging filter was added to accentuate the periodicity of the magnitude peak. A symbol is expected every sixty-four samples. By averaging  $|Y_I(t)|$  and  $|Y_Q(t)|$  with  $|Y_I(t - NN_s + 1)|$  and  $|Y_Q(t - NN_s + 1)|$ , respectively, the energy in spurious peaks is reduced, lowering the probability of false detection. This filter can have a negative impact on the system when timing jitter causes consecutive peaks to occur at different sampling instances within a bit period. If this is the case, this filter should be removed.

### **Acquisition Process**

From the filtered magnitude, the location of the most persistent peak within the symbol window must be determined. This task is addressed by continuously finding the location of the maximum energy within every symbol window. If the maximum occurs at roughly the same point  $M$  samples in a row, the sample index for the current user is set to the position of the persistent maximum. Since there may be timing jitter, the two LSBs are not

considered when comparing the current and previous maximum indexes. The actual levels of the maximums are not taken into consideration, only their positions.

## Tracking Process

Once the sample index is obtained, the peak is said to be acquired. Due to timing misalignments, the position of this peak may drift within the symbol window so it must be tracked continuously. The sample index can also be referred to as the *on-time* index. The peak is tracked by noting the sample before and after the on-time sample, the *early* and *late* indexes, respectively.

Short-term averages of the magnitudes occurring at the early, late, and on-time indexes are taken. Currently, the integration time for the tracking process is sixteen bits. This allows for some delay in adjusting the sample point to avoid losing lock in the presence of amplitude fluctuations caused by other users, noise, and other channel impairments. The averaging also helps reduce the impact of “edge” effects that can occur when the sample index is at either zero or sixty-three.

The accumulation registers are large enough so that quantization effects are not introduced. At the end of the tracking integration time, the values are averaged and the following three tests are performed:

1. Is the late average greater than the symbol average?
2. Is the early average greater than the symbol average?
3. Is the early average greater than the late average?

If both test 1 and 2 are FALSE, nothing is changed. If either test 1 or 2 is TRUE, the sample index is adjusted by one sample. Test 3 determines the direction of the adjustment. If test 3 is TRUE, the sample index is decremented; otherwise it is incremented. This changes the

sample index for the next integration period. The next block of data will be demodulated at the new sample point. This operation is repeated unless otherwise directed by the detection process.

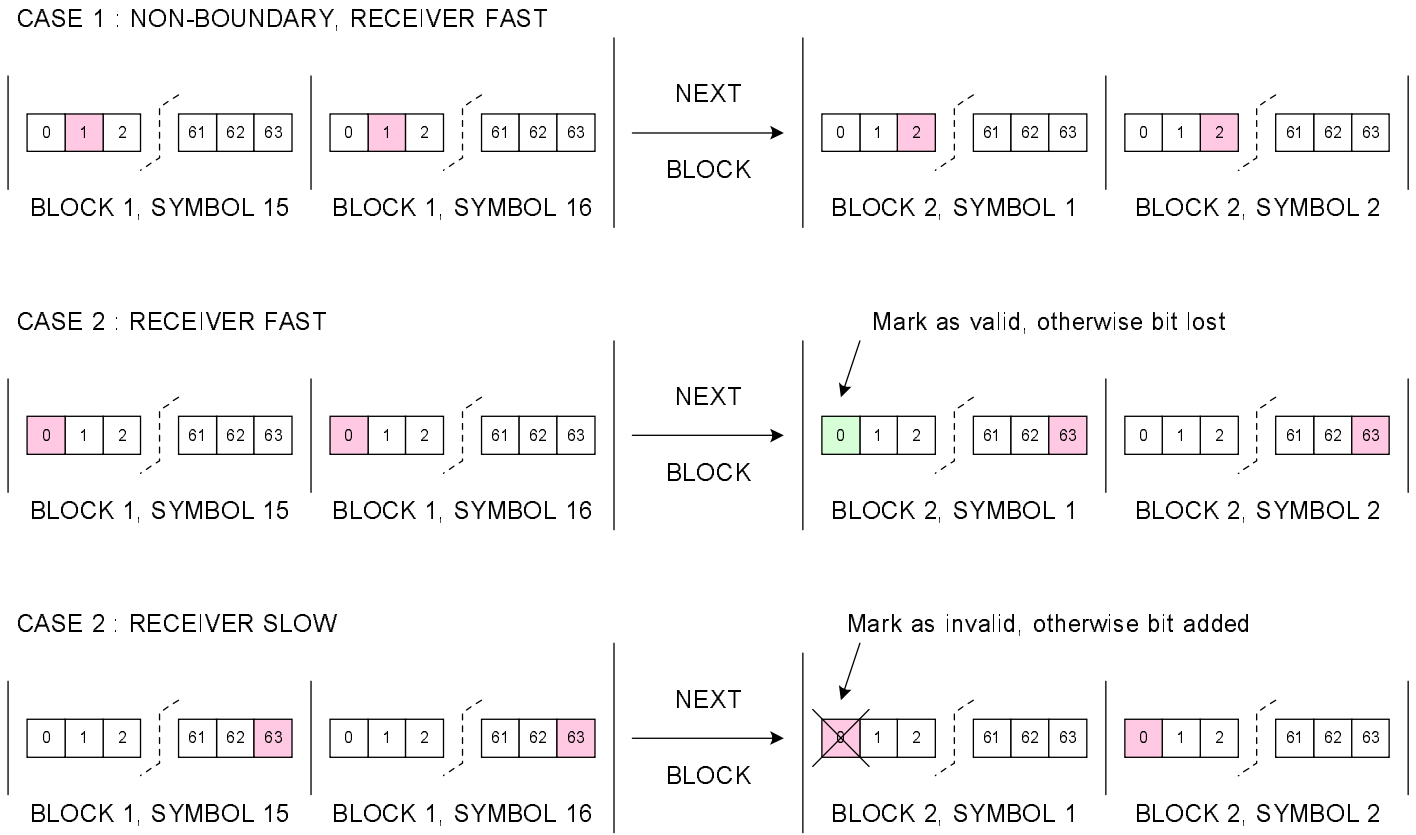
### Window Boundary Condition

Once the sample point is adjusted in the user's program header, a special sampling process is utilized in the regenerator of the first stage and the demodulator in the second stage. Typically, this involves a window counter that is reset during the PROGRAM state and enabled during the BUSY state. This module- $NN_s$  counter, 64 in this case, maintains the constant symbol window across multiple blocks. If, after a block, the tracking process adjusts the user's sample point across window boundaries, i.e. from zero to sixty-three or from sixty-three to zero, then the sample process must account for this.

Figure 4.6 illustrates the three possibilities that could occur. In the first case, even after adjustment, the sample point does not lie on a window boundary, thus shifting the sample index maintains the correct alignment. In the second case, when the receiver clock is fast relative to the transmitter clock, the optimal sample index gradually shifts to the left. Eventually the tracking process shifts the sample index from zero in Block 1 to sixty-three in Block 2. If this condition is detected, the first sample of the second Block is marked as a symbol; otherwise it is lost.

The opposite occurs in the the third case when the receiver clock is slow relative to the transmitter clock. Here, the sample index shifts from sixty-three in Block 1 to zero in Block 2. The last symbol in Block 1 was already sampled at index sixty-three. By sampling at index zero in Block 2, the symbol is effectively sampled again. Thus when the third case is detected, the first sample of Block 2 is marked invalid to avoid sampling.

Figure 4.6: Boundary conditions that occur due to the tracking process.



## User Detection

Concurrent to the acquisition and tracking processes is the detection process. The detection process must determine whether the current user is present in the system by taking a long-term average of the on-time sample value and comparing it to a projected threshold. Currently, the detection window is 256 bits long. When detected, the acquisition status of the user is changed from zero to one. The acquired bit is used by other parts of the system such as the demodulator and the cancellation process. When either the user stops transmitting or tracking process loses the peak, the detection average falls below the threshold. The acquired bit is then reset and the module falls from tracking back to acquisition.

### 4.4.4 Regenerate Module

The Regenerate module generates an estimate of the current user's signal only if they are acquired. From the programming header, the module extracts the user ID, acquisition status, sampling index, and PN code. To provide contiguous operation for each user across multiple blocks, the module must store the user's current sampled amplitude estimate and the state of a sample counter unique to each user. These are stored in local RAM, which is addressed by the user's ID. Thus when regeneration of a user is interrupted and then restarted again, the module knows from where it left off during the previous block.

Upon encountering valid data, the sample counter starts counting modulo- $NN_s$ . When the counter equals the user's sample index, the corresponding input sample is latched in and the user's unique sample counter is reset. The latched symbol is the user's decision statistic and it provides an amplitude estimate of the user's transmitted signal. The upper four bits of the user's count are used to address a decoder. Its input is the PN code extracted from the user's program header. The output of the decoder is the user's current chip, which is used to modulate that user's amplitude estimate of the their transmitted bit. Thus, it regenerates the user's DS/SS signal. If the user is not acquired, the output of the processing pipeline is

zero.

#### 4.4.5 Combine Module

The Combine module forms an estimate of the received signal by summing up the regenerated signals of all acquired users in the system. This is done by accumulating the valid data portion of each stream over the period of one processing block. Since the data size is large, the module must interface to external memory to store the partial results.

For every processing block,  $K$  *estimated* received signal streams—one for each user in the system—enters the module. If acquired, the user’s estimate is added to a running sum for the current block and the result written to external memory. If not acquired, the user does not contribute to the running sum for the block. The only program information needed is whether the current user is the last user. Only for the last user is the output of the module marked valid. This allows other modules down the pipe to use only the final summation of the Estimated Received signal stream.

#### 4.4.6 Revised Module

The Revised module creates a corrected version of the received signal based on an estimate of interference caused by each user in the system. It forms the *revised* received signal  $\hat{r}(t)$  by using Equation 4.3,

$$\hat{r}(t) = r(t) + c \left[ r(t) - \sum_{j=1}^K s_j(t - \tau_j) \right], \quad (4.3)$$

where  $c$  is the backoff factor applied,  $r(t)$  is the *actual* received signal, and  $s_j(t - \tau_j)$  is the *estimated* signal transmitted by user  $j$ .

The Revised module operates on valid data that, due to the Combine module, only occurs

during the last user's stream. During the PROGRAM state, the module does not latch in any data from the program header. During the BUSY state, the module computes Equation 4.3. Due to the filtering used, the estimation process introduces a delay in the *estimated* received signal of  $NN_s - 1$  samples with respect to the *actual* received signal. To correct this, the actual received signal is also delayed by  $NN_s - 1$  samples efficiently using RAM internal to the FPGA.

The processing pipeline for this module spans four stages. To prevent overflow, the bus width is extended to sixteen bits throughout the processing pipeline. The first stage computes an error signal, corresponding to the terms surrounded by brackets in Equation 4.3. Stages 2 and 3 weight the error signal according to the backoff factor selected and add it back to the delayed *actual* received signal. The fourth stage checks for saturation in the final result and clips accordingly.

#### 4.4.7 Buffer Module

The Buffer module functions just as the Input module does but it handles stream input rather than low-rate continuous data. Again, two concurrent processes are utilized with one controlling the input and the other the output of the module. A double buffering scheme is used where the input process reads from one bank in external memory while the write process updates another.

The two independent processes are synchronized when the first user's programming header enters the module. When triggered, the location of the READ and WRITE banks are swapped and the output process begins generating streams using the READ bank of external memory. First, the preamble is output followed by the programming header, which is taken from internal RAM. The module then produces  $NN_s - 1$  words of valid filter initialization followed by the data stored in the READ bank. This is done  $K$  times where  $K$  is the number of users in the system, to form the processing block. While awaiting the next trigger in the IDLE state, the module outputs null words.

During the PROGRAM state, the input process captures the user’s programming header and places it in internal memory. This memory is addressed based upon the user’s ID. During the BUSY state, if the current user is the last user in the system, the input process writes that user’s valid data to the WRITE bank.

#### 4.4.8 Differential Demodulator Module

The Differential Demodulator module performs noncoherent detection of the current user. Noncoherent detection lowers receiver complexity by reducing the dependence upon phase alignment between the transmitter carrier and the receiver local oscillator. The data is differentially encoded by the transmitter. For differential BPSK, the bit sequence is no longer represented by the actual phase of the transmitted signal but in the phase transitions. Thus each bit then relies on two symbols, increasing the probability of error with respect to coherent demodulation. To detect a bit from the match filtered complex envelope of the received signal  $Y(t)$ , the module forms a decision statistic  $Z_n$  by computing the dot product between the current and previous complex symbols:

$$Z_n = Y_{I,n} * Y_{I,n-1} + Y_{Q,n} * Y_{Q,n-1}. \quad (4.4)$$

From the programming header, the module latches in the current user’s ID and sampling index. A modulo- $NN_s$  counter keeps track of the symbol window. When the counter equals the user’s sampling index, the input samples from the  $I$  and  $Q$  streams are latched into the symbol registers.

The processing portion of the module consists of three pipelined stages that are enabled every time the on-time sampling point is reached. To provide contiguous operation across input blocks, each user’s previous complex sample is stored in internal SRAM, which is addressed based upon the user’s ID. The pipeline is illustrated in Figure 4.7.

The first stage latches in the complex symbol. The next stage computes the product of the current and previous symbols for both  $I$  and  $Q$  in parallel. In the third stage, the two products are added together to form the decision statistic. The module output is the top ten bits of the resulting decision statistic. For this prototype, hard limiting is used requiring only one bit of information. However, ten bits are still required to pass programming information. This also enables soft-decision-based modules to be easily added to the system.

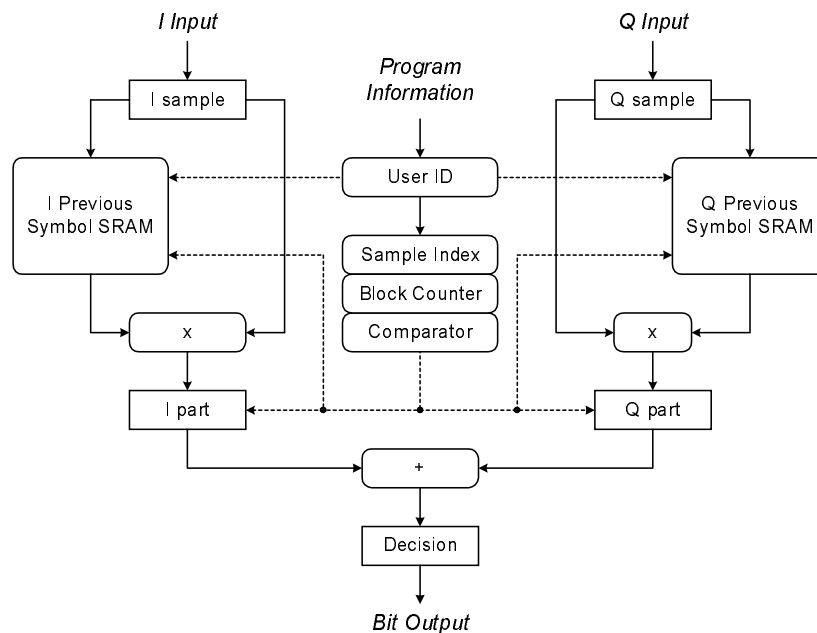


Figure 4.7: Diagram of the Demodulator module processing pipeline.

#### 4.4.9 Output Module

The Output module is responsible for collecting each user's incoming data bits and interfacing with the host PC. Since hard-decisions are used, only the MSB of the input data is utilized. The Output module is comprised of a serial-parallel converter, four FIFOs, and the GigaOps HBUS interface. A block diagram of the Output module is shown in 4.8. The serial-parallel converter groups every sixteen bits into a word that is then latched into the current user's FIFO.

During the PROGRAM state, the user's ID is latched into local storage and is then used to select the active FIFO. Each FIFO belongs to a certain user and is 128x16 bits in dimension. This allows a maximum of 2048 bits, corresponding to 65.5  $\mu$ sec, worth of data to be stored at one time. This gives the host ample time to poll and extract information before the FIFO overflows.

The two registers, shown in Figure 4.9, are mapped into the memory space of the host application. To read a sixteen bit word of data from a FIFO on the G900, the host program first polls the status of all FIFOs by reading the variable mapped to the status register. The sixteen bit register is broken down into four nibbles, but only three are used. They indicate whether each of the four users' FIFOs are empty, full, or have overflowed. Once it has been determined that a particular user's FIFO is not empty, a data transfer via the HBUS protocol is made. This is done whenever the host application queries the contents of the mapped data register. Upon the transfer, the FIFO strips off the old word automatically. Error handlers for the cases where overflows have been detected can be added to the host application if needed.

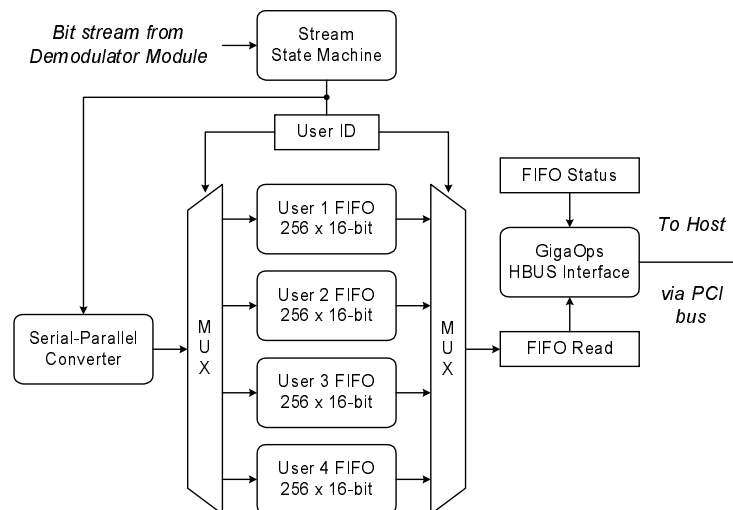


Figure 4.8: Diagram of the Output module.

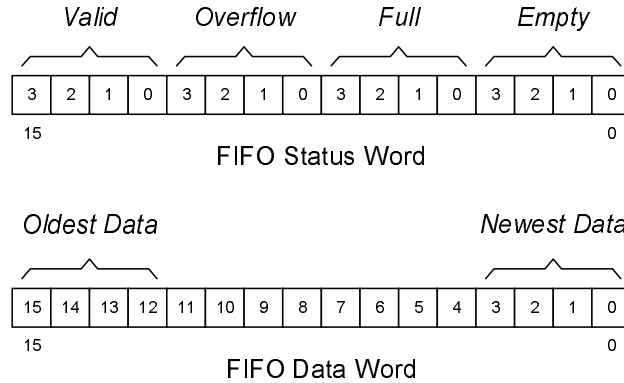


Figure 4.9: Memory-mapped registers within the Output module.

## 4.5 Algorithm Mapping and Synthesis

The next step in implementation was to efficiently map the modular receiver design to the resources available on the GigaOps G900 computing platform. In all, six XMODs separated into two stacks of three form the multiuser receiver. Excluding formulation of the *revised* received signal, the entire first stage of the receiver is mapped to the first stack of three XMODs. These are designated as the *INPUT1*, *MFC1*, and *MURX1* XMODs. Their functionality is briefly summarized below. A hardware block diagram of the G900-based multiuser receiver is illustrated in Figure 4.10.

- *INPUT1* : Buffers the incoming IF data and generates the streams used in the first stage of the receiver for both the *I* and *Q* channels. This module also contains the digital noise generators used for analysis and testing.
- *MFC1* : Correlates the *actual* received signal stream with the desired user's PN code for both the *I* and *Q* channels.
- *MURX1* : Acquires and tracks each user. Each detected user is then regenerated and summed to form an estimate of the received signal for both the *I* and *Q* channels.

The remaining portion of the first and all of the second stage of the receiver fits within the second stack. These are designated as the *INPUT2*, *MFC2*, and *MURX2* XMODs. Their functionality is briefly summarized.

- *INPUT2*: Formulates and buffers the incoming *revised* received signal. This module also generates the streams used in the second stage of the receiver.
- *MFC2*: Correlates the *revised* received signal stream with the desired user's PN code for both the *I* and *Q* channels.
- *MURX2*: Demodulates each user and buffers the received bits. This module also communicates with the Host PC relaying receiver configuration information and received data.

The composition of each of the six XMODs is now described in detail. Within each of the XMODs are two FPGAs that were independently designed and synthesized. The final FPGA synthesis results for each of the twelve FPGAs in the system are also summarized according to XMOD.

#### 4.5.1 *INPUT1* XMOD

The *INPUT1* XMOD generates the necessary streams processed by the first stage of the receiver as described in Section 4.4. The XMOD accepts two sixteen-bit busses, representing the in-phase and quadrature portion of the received signal, from the Sigtek ST-114 at a sample rate of 2 MHz. The data enters the *X* FPGA via X6 and X7. Only the top ten bits are utilized from each channel.

Since each XMOD has access to a single external SRAM, only one Input module per FPGA could be used; thus, the *I* channel is handled by the *X* FPGA and the *Q* channel is handled by the *Y* FPGA. A block diagram of the *INPUT1* XMOD is shown in Figure 4.11. The *Q*

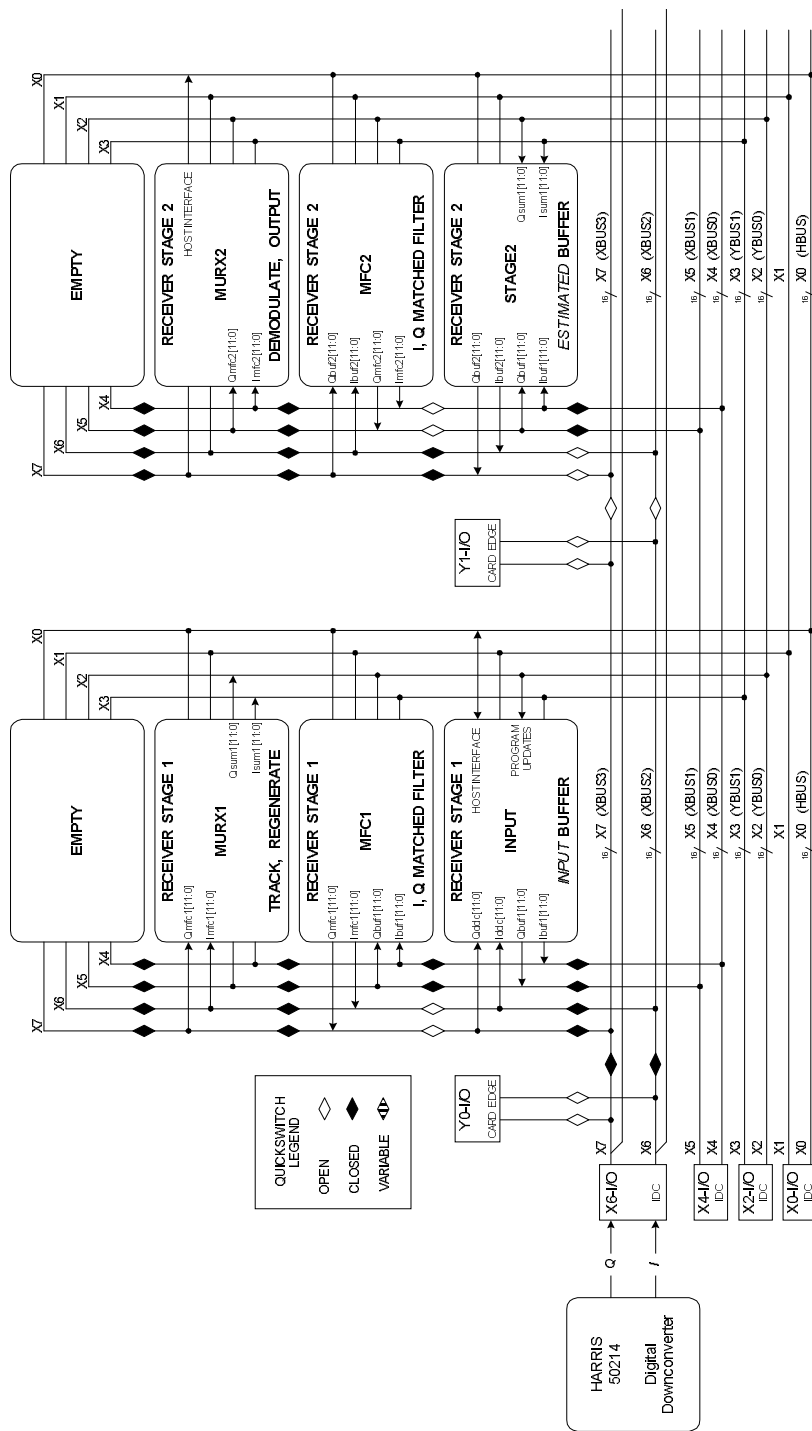


Figure 4.10: Hardware block diagram of the G900-based receiver.

channel data is routed into and out of the  $Y$  FPGA through the  $X$  FPGA. Registers added in the  $I$  pipeline equalize the relative delay between the two pipelines due to the extended routing of the  $Q$  pipeline. The two Input modules are synchronized by a counter located within the  $X$  FPGA. Its maximum count determines the size of the buffered data blocks, in terms of input samples, of each block.

Both the  $X$  and  $Y$  FPGAs contain the external SRAM interface, which allows external SRAM read and write operations to occur within one 10 MHz processing cycle. This raises the clocking requirements of the FPGAs, which increases the overall run time of the synthesizer to meet the timing constraints.

Synthesis results for the  $X$  and  $Y$  FPGAs of the  $MFC1$  XMOD are given in Table 4.3. The  $X$  FPGA contains slightly more logic, but overall the devices are nowhere near full utilization. This inefficiency is a direct result of limited access to memory resources on the G900. The resulting design speeds meet or slightly exceed the required 10 MHz system processing rate.

Table 4.3: Synthesis results for the  $INPUT1$  XMOD.

	X FPGA	Y FPGA
CLB utilization	237 of 1024 (23%)	216 of 1024 (21%)
CLB flip-flops	362 of 2048 (17%)	324 of 2048 (15%)
IOB utilization	119 of 160 (74%)	75 of 160 (46%)
IOB flip-flops	110	58
Equivalent gate count	8107	7415
Maximum path delay	74.931 ns (13 MHz)	96.465 ns (10 MHz)

#### 4.5.2 $MFC1$ XMOD

The  $MFC1$  XMOD contains the matched filter bank for both the  $I$  and  $Q$  channels. A diagram of the internals of the  $MFC1$  XMOD are shown in Figure 4.12. Due to the relatively large size of the Matched Filter Correlator module, an FPGA is reserved for each channel. This provides more flexibility in the event an even larger, more complicated correlator

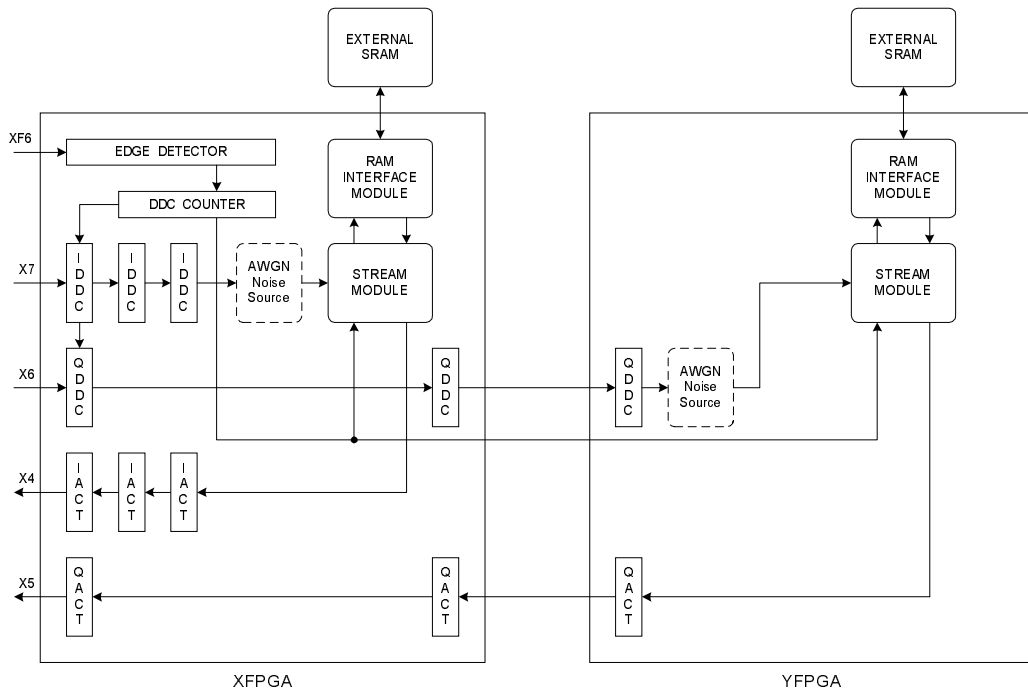


Figure 4.11: Diagram of the *INPUT1* XMOD.

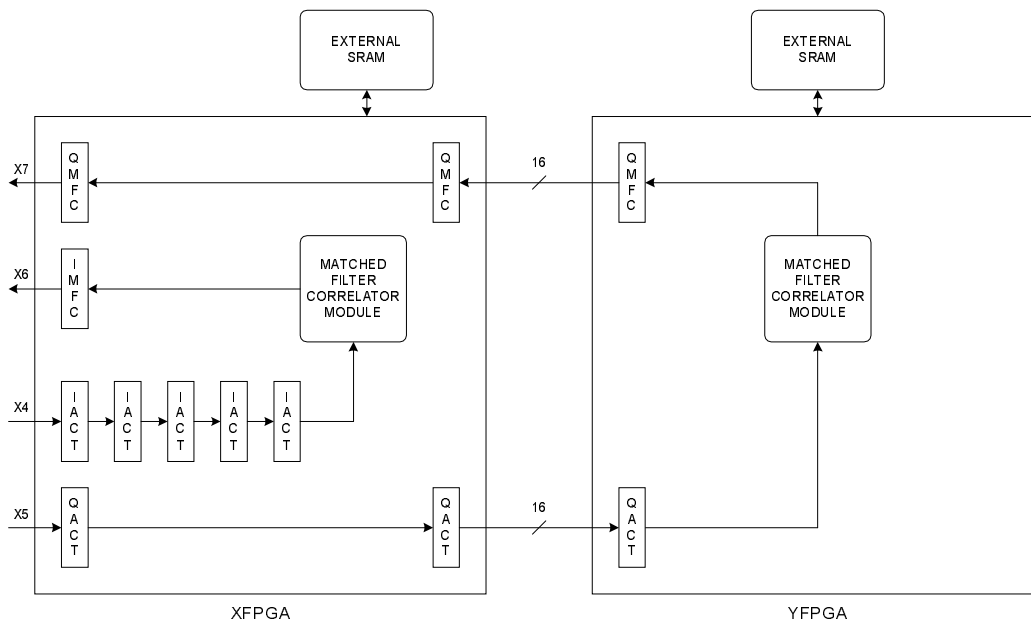


Figure 4.12: Diagram of the *MFC1* XMOD.

structure is needed.

The  $I$  and  $Q$  *actual* received signal streams from the *INPUT1* XMOD enter the *MFC1* XMOD on busses X4 and X5. The  $X$  FPGA contains the  $I$  channel Matched Filter module plus a pass-through for the  $Q$  channel to the  $Y$  FPGA. The  $Y$  FPGA contains the  $Q$  channel matched filter where the  $Q$  *actual* received signal is processed and the result looped back to the  $X$  FPGA.

The relative delay between the two paths is normalized by the addition of registers along the  $I$  channel. It is good practice to register the inputs and outputs of signals entering and leaving the FPGA. These added delays are placed throughout the design as shown in Figure 4.12. This allows for the flip-flops in the IOBs, located around the peripheral of the FPGA die at each pin, to be utilized minimizing routing delays and increasing system performance. It is important to note that the flip-flops in the IOBs do not contain clock-enable circuitry. To have registers in the design mapped to IOB registers automatically by the synthesis software, the VHDL source should not place any conditions on the loading of the registers.

The resulting *filtered actual* received signal streams are then placed on the X6 and X7 busses for the *MURX1* module. The quickswitches for the X6 and X7 busses are disabled by the  $Y$  FPGA to isolate them for the *MFC1* and *MURX1* XMODs.

Table 4.4: Synthesis results for the *MFC1* XMOD.

	X FPGA	Y FPGA
CLB utilization	672 of 1024 (65%)	649 of 1024 (63%)
CLB flip-flops	1196 of 2048 (58%)	1148 of 2048 (56%)
IOB utilization	98 of 160 (61%)	44 of 160 (27%)
IOB flip-flops	92	28
Equivalent gate count	13928	13256
Maximum path delay	46.736 ns (21 MHz)	47.554 ns (21 MHz)

Synthesis results for the  $X$  and  $Y$  FPGAs of the *MFC1* XMOD are given in Table 4.4. Device

utilization is approximately equal for the two FPGAs. Since the FPGAs of the *MFC1* XMOD use only the primary processing clock (10 MHz), they are well below their maximum clocking speed of 21 MHz.

### 4.5.3 *MURX1* XMOD

The *MURX1* XMOD contains the acquisition and tracking mechanism and the estimation portion of the interference cancellation operation. A diagram of the internals of the MFC XMOD are shown in Figure 4.13. The formulation of the *revised* received signal is dependent on the *actual* and *estimated* received signals. Although the XMOD has access to the *actual* received signal, placing the Revised module in the *X* and *Y* FPGAs requires four streams to cross between the FPGAs. Even at twelve bits per stream, there are not enough interconnections available to support this. Thus the Revised Module was bumped to the *INPUT2* XMOD.

The *I* and *Q* channels are handled by separate FPGAs since the estimation procedure for each channel requires interfacing with external memory. The *X* FPGA contains the Acquisition/Tracking module plus the Regenerate and Combine modules for the estimation of the *I* channel received signal. The result is passed to the *Y* FPGA to access the desired output bus.

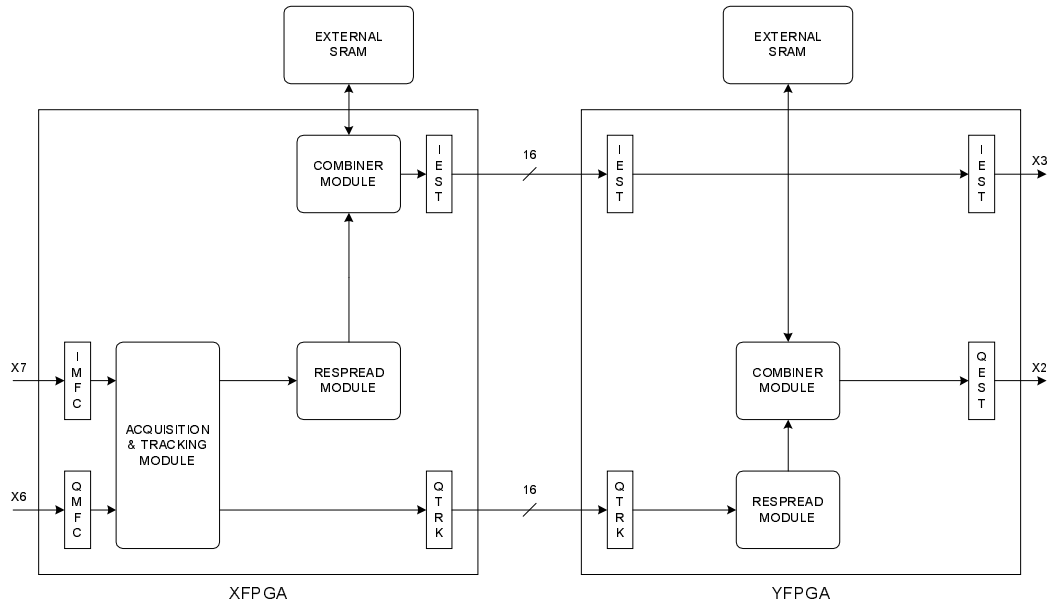
The *Y* FPGA contains the pass-through for the *I* channel *estimated* received signal plus the Regenerate and Combine modules used for estimation of the *Q* channel. The resulting *I* and *Q estimated* received signal streams are then placed on busses X2 (YBUS0) and X3 (YBUS1), respectively. Again, delay registers along both paths are used to equalize the delay.

Synthesis results for the *X* and *Y* FPGAs of the *MURX1* XMOD are given in Table 4.5. Device utilization is highly asymmetrical due to the presence of the Acquisition/Tracking module in the *X* FPGA. The designs have a relatively low maximum clock speed of 12–15

Table 4.5: Synthesis results for the *MURX1* XMOD.

	X FPGA	Y FPGA
CLB utilization	507 of 1024 (49%)	243 of 1024 (23%)
CLB flip-flips	671 of 2048 (32%)	400 of 2048 (19%)
IOB utilization	92 of 160 (57%)	109 of 160 (68%)
IOB flip-flips	102	108
Equivalent gate count	15185	5591
Maximum path delay	83.421 ns (12 MHz)	67.022 ns (15 MHz)

MHz to keep the synthesis run times to a minimum.

Figure 4.13: Diagram of the *MURX1* XMOD.

#### 4.5.4 *INPUT2* XMOD

The formulation of the *revised* received signal and the stream generation for the second stage of the receiver takes place in the *INPUT2* XMOD. It is located at the bottom of the second stack of XMODs. A diagram of the internals of the *INPUT2* XMOD are shown in Figure 4.14.

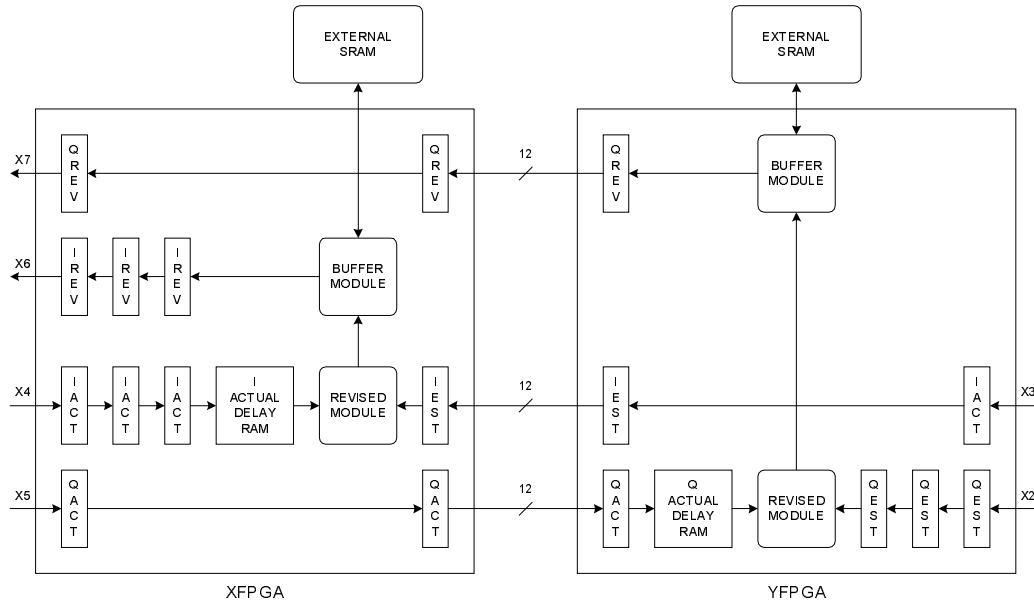


Figure 4.14: Diagram of the *INPUT2* XMOD.

The *I* and *Q actual* received signal from the *INPUT1* XMOD enter via X6 and X7. The *estimated* received signal streams from the *MURX1* XMOD enter via X2 and X3. Due to the latency in the first stage of the receiver, the *estimated* received signal stream is delayed by a large number of cycles with respect to the *actual* received signal stream. A large RAM-based delay mechanism is used to efficiently align the two. Again, the *X* and *Y* FPGAs handle the *I* and *Q* channels, respectively. The *X* FPGA contains pass-throughs for the *Q actual* and *revised* streams plus Revised and Buffer modules for the *I* channel. The *Y* FPGA contains a pass-through for the *I* channel actual stream in addition to the *Q* channel Revised and Buffer modules. The *revised* streams are then passed to the *MFC2* XMOD along X6 and X7. Additional registers are used for delay normalization along both paths.

Synthesis results for the *X* and *Y* FPGAs of the *INPUT2* XMOD are given in Table 4.6. Device utilization is symmetric between the two devices. If additional external memory and I/O was present, the designs could easily be merged into a single FPGA.

Table 4.6: Synthesis results for the *INPUT2* XMOD.

	X FPGA	Y FPGA
CLB utilization	359 of 1024 (35%)	347 of 1024 (33%)
CLB flip-flops	428 of 2048 (20%)	404 of 2048 (19%)
IOB utilization	132 of 160 (82%)	111 of 160 (69%)
IOB flip-flops	122	94
Equivalent gate count	17090	16778
Maximum path delay	95.654 ns (10 MHz)	99.615 ns (10 MHz)

#### 4.5.5 *MFC2* XMOD

The *MFC2* XMOD contains the matched filter banks used in the second stage of the receiver. It is almost identical to the *MFC1* XMOD and is located in the middle of the second stack of XMODs. A diagram of the internals of the *INPUT2* XMOD are shown in Figure 4.15.

The *I* and *Q revised* received signals enter via X6 and X7. The *X* FPGA contains pass-throughs for the *Q* channel revised and filtered revised streams plus the *I* channel Matched Filter Correlator module. The *Y* FPGA contains only the *Q* channel Matched Filter Correlator module. The resulting Filtered revised streams are passed to the *MURX1* XMOD via the X4 and X5 busses. To isolate these busses from those carrying the estimated streams, the *Y* FPGA disables the quickswitches.

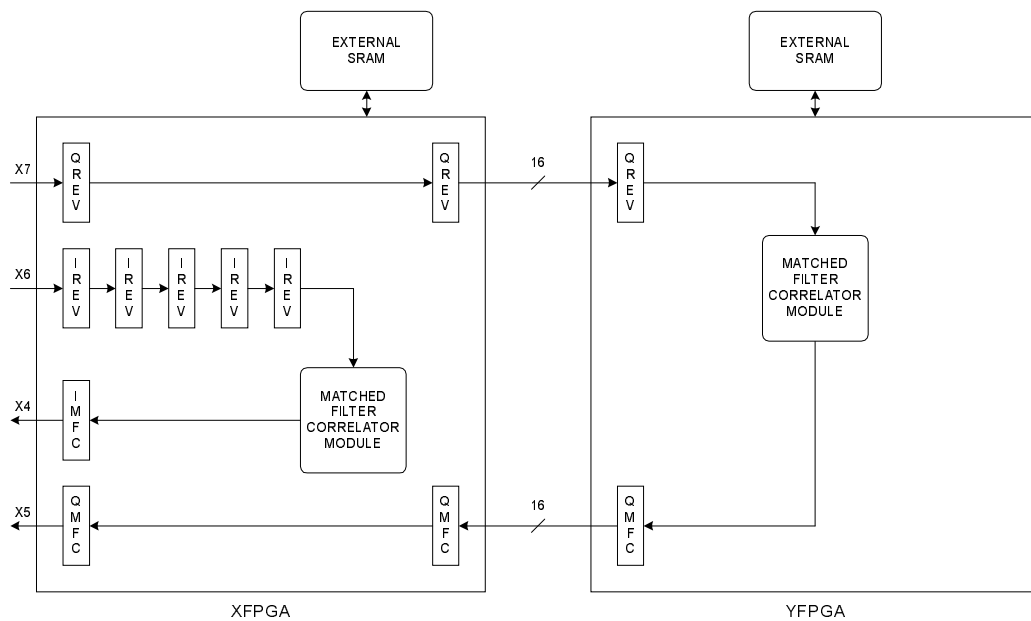
Additional registers are inserted along the *I* channel for equalization of delay between the two channels. Synthesis results for the *X* and *Y* FPGAs of the *MFC2* XMOD are supplied in Table 4.7. They are comparable to those of the *MFC1* XMOD.

#### 4.5.6 *MURX2* XMOD

The *MURX2* XMOD contains the demodulator, user data FIFOs, and the PC interface. A diagram of the *MURX2* XMOD is shown in Figure 4.16. The Output module can vary based upon the size of the FIFOs desired and the number of users in the system. In addition, it

Table 4.7: Synthesis results for the *MFC2* XMOD.

	X FPGA	Y FPGA
CLB utilization	674 of 1024 (65%)	651 of 1024 (63%)
CLB flip-flops	1196 of 2048 (58%)	1148 of 2048 (56%)
IOB utilization	99 of 160 (61%)	44 of 160 (27%)
IOB flip-flops	92	28
Equivalent gate count	13952	13280
Maximum path delay	62.445 ns (16 MHz)	59.786 ns (16 MHz)

Figure 4.15: Diagram of the *MFC2* XMOD.

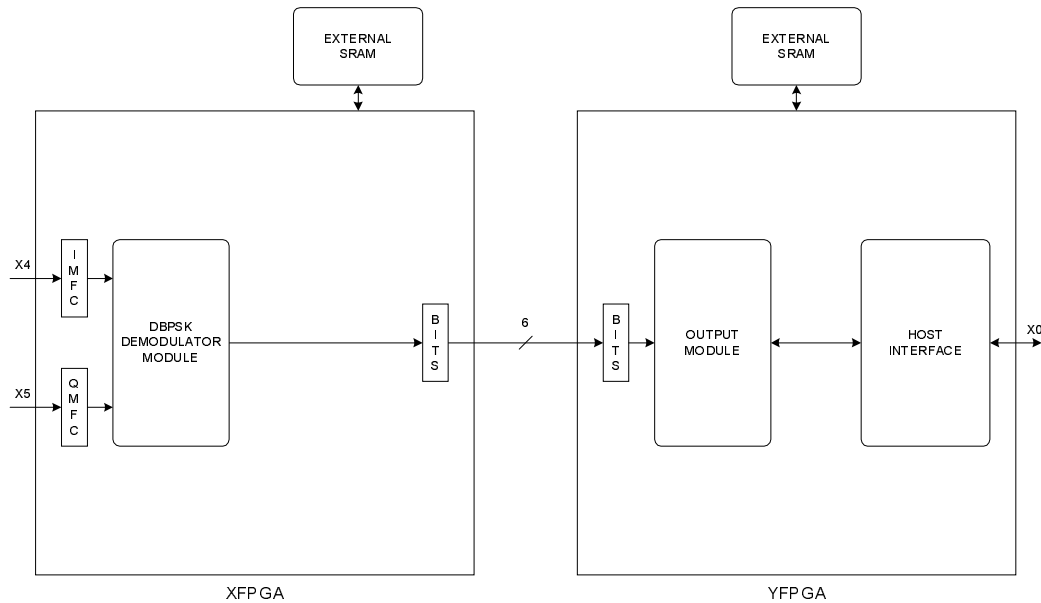


Figure 4.16: Diagram of the *MURX2* XMOD.

also requires communication with the Host PC via the HBUS that can only be performed by the *Y* FPGA. Thus reserving the entire *Y* FPGA for the Output module provides added flexibility when upgrading or modifying the system.

Only the Differential Demodulator is mapped to the *X* FPGA. Again, this leaves room for extra logic whether it be for more complicated demodulation, decryption, compression, or forward error correction schemes. The inputs to the XMOD are the *revised* received stream taken from busses X4 and X5. The *X* FPGA passes the User Bit stream, the output of the Demodulator module, to the input of the Output module located inside the *Y* FPGA.

No additional registers are required for delay equalization. Synthesis results for the *X* and *Y* FPGAs of the *MURX2* XMOD are supplied in Table 4.8.

Table 4.8: Synthesis results for the *MURX2* XMOD.

	X FPGA	Y FPGA
CLB utilization	266 of 1024 (25%)	839 of 1024 (81%)
CLB flip-flops	170 of 2048 (8%)	204 of 2048 (10%)
IOB utilization	38 of 160 (23%)	48 of 160 (30%)
IOB flip-flops	27	48
Equivalent gate count	5312	71692
Maximum path delay	65.187 ns (16 MHz)	59.872 ns (16 MHz)

## 4.6 Implementation Summary

Based upon the Xilinx Design Manager’s estimates of the final FPGA implementations, the gate count for the first stage of the receiver is approximately 64,000 gates. The projected gate counts for each FPGA are summarized in Table 4.9. Neglecting the Output module in the *MURX2* Y FPGA, the second stage of the receiver roughly equates to 66,000 gates. The Output module itself is 72,000 gates since it contains the deep FIFOs. These estimates are likely inflated since they include extra logic necessary for partitioning the design over multiple chips.

Table 4.9: Summary of estimated gate count of final FPGA implementations.

XMOD	X FPGA	Y FPGA	Total
<i>INPUT1</i>	8107	7415	15522
<i>MFC1</i>	5312	13256	27184
<i>MURX1</i>	13928	5591	20776
Subtotal	37220	26262	63482
<i>INPUT2</i>	17090	16778	33868
<i>MFC2</i>	13952	13280	27232
<i>MURX2</i>	5312	71692	77004
Subtotal	37220	26262	138104

### 4.6.1 Computational Performance

One metric that provides insight into the performance of the receiver is the number of operations performed per second. Table 4.10 provides a breakdown of the number of arithmetic operations performed by each module in the system. Here, the number of operations per sample and per bit are tallied to form an estimate of the overall receiver performance. Given an input sample rate of 2 MHz and a bit rate of 31.25 kHz, the FPGA-based receiver performs over 660 million arithmetic operations per second, or 21,000 operations per bit. An arithmetic operation is denoted as either a multiply, add, or subtract. This provides a conservative estimate of the total number of operations the receiver performs since it does not include other operations, such as logical or move operations.

Table 4.10: Breakdown of arithmetic operations per module.

Module Type	Number of Modules	Number of Users	Operations per Sample	Operations per Bit	Operations per Second
Input	2	0	0	0	0
MFC	4	4	19	0	608,000,000
Tracking	1	4	1	4	8,500,000
Regenerate	2	4	1	0	16,000,000
Combine	2	4	1	0	16,000,000
Revised	2	1	3	0	12,000,000
Buffer	2	0	0	0	0
Demod	1	4	0	3	375,000
Output	1	0	0	0	0
Total	17	21	25	7	660,875,000

### 4.6.2 External Memory Access

Another metric that gives insight into the operation of the receiver is the amount of external memory access. This is based upon the sampling rate,  $F_s$ , coming from the DDC and the number of users processed,  $K$ . For this system, the input sampling rate is 2 MHz and

the receiver processes four users. The modules that access external memory are the Input module, the Combine module, and the Buffer module. This design utilizes two of each module for a total of six external SRAMs clocked at 20 MHz. Both the Input and Buffer modules write to external SRAM at  $F_s$  and read from it at  $KF_s$ . The Combine module reads and writes at  $(K - 1)F_s$ . Thus, collectively the six external SRAMs are being accessed a total of sixty-four million times per second.

## 4.7 Host-Based Receiver Control Application

The application running on the PC host controls the operation of the G900-based multiuser receiver. It handles initialization and testing of the G900, loading of the FPGAs that make up the receiver, and other utilities. For testing the multiuser receiver operation and gauging performance, a BER testing utility is used. When in BER test mode, the host program continuously polls the Output module for the FIFO status of the desired user. When not empty, sixteen bits of the user's demodulated data are then transferred to the host. Using partial correlations, the host program continuously attempts to synchronize its local copy of the user's transmitted data pattern to the received data. Once acquired, the BER is calculated. During calculation, the window of bits for which the BER is calculated, the digital noise level, and cancellation level are adjustable. Acquisition is assumed lost once the BER reaches near 50%. If this occurs, the program automatically attempts to resynchronize. The host program can also be configured for automated data collection utilizing large sample sets and multiple operating conditions.

## 4.8 Summary

In summary, this chapter detailed the implementation of the FPGA-based multiuser receiver and the associated hardware testbed. Since a commercial board was selected to implement

the receiver prototype, certain compromises were required. The interconnection scheme is not ideal when used as the base-level engine for the system described here. System data is restricted to a width of ten bits due to both the system-wide bus widths and the limited amount of connectivity provided between the two FPGAs on individual XMODs. The end result was successful but largely inefficient due to constraints imposed by the G900's inadequate routing resources and limited access to memory resources.

# Chapter 5

## Results

In this chapter, the testing and verification process of the FPGA-based multiuser receiver is outlined. Presently, the system has only been tested at baseband using the digital inputs of the FPGA-based multiuser transmitter. The methodology of testing is presented along with a description of the baseband transmitter and digital noise generator. These components allow the system performance to be estimated under various conditions of capacity and noise. Due to the way in which the user and channel environments are created, the comparison to expected results will be somewhat optimistic. Despite these inaccuracies, verification of correct receiver operation is obtained along with relative performance improvements using partial parallel interference cancellation over a conventional multiuser receiver.

### 5.1 System Calibration

Since there is no automatic gain control (AGC) within the multiuser receiver, the one on the HSP50214 was used. Since the acquisition and tracking mechanism assumes a predetermined minimum input level, the system has to be calibrated to ensure this level is met. The Sigtek control software is used to modify the on-board AGC parameters. To ensure complete

mapping of the received signal within the dynamic range allotted, all users are enabled. For the loop, the threshold is set to -10 dBFS, the minimum gain is set to 2 dB, the maximum gain is set to 90.0 dB, and the loop gain is set to 0.001. The loop is allowed to settle, and after some period of time, the loop gain is set to zero effectively turning AGC off. The users are then changed to suit the needed configuration. Since only a few users are used, keeping the AGC enabled can be harmful. As users come onto the system, the input signal level increases and the AGC reacts by scaling it down. Thus the amplitude estimates, which the system uses for cancellation, will decrease possibly below levels needed to maintain acquisition. By calibrating for all users, the worst case scenario can be tested and the correct AGC levels applied to meet the tracking requirements.

## 5.2 FPGA-Based Multiuser Transmitter

During the design and test phase, in absence of the receiver RF front-end and individual RF-capable DS/SS transmitters, a Xilinx XC4010-based FPGA evaluation board was used to simulate a CDMA multiuser environment. The multiuser transmitter FPGA contains four single-user DS/SS transmitter modules. Each differentially encodes and spreads a given user. Specific to each user is a length-16 spreading code, a data source based on an M-sequence generator, and a predetermined slip-rate based on a large prime number. The users are generated and combined at a sample rate of 2 MHz allowing them to be asynchronous at resolutions of up to a quarter of a chip. The resulting output is re-clocked at 16 MHz and fed to the external digital input of the DDC evaluation board. A push button on the evaluation board allows the users to be synchronized, and external DIP switches control slipping and activation of each user.

### 5.3 Noise Generator

To test the receiver under noise-limited conditions, an FPGA-based digital noise generator module was created and placed on both the I and Q channels before entering the input module. The generator operates based on the Central Limit Theorem adding forty ten-bit uniform random numbers together for each Gaussian random variable.

The I and Q data arrives at 2 MHz. Utilizing the 20 MHz clock present for overclocking the external memories, ten uniform numbers can be added together per input sample. The uniform numbers are generated using an M-sequence generator. The ten-bit uniform number is formed by taking the lower ten bits of the M-sequence generator's shift register. By utilizing four of these generators in parallel, a total of forty uniform random numbers can be added to form one Gaussian random number. Each M-sequence generator has a unique length and was created using Xilinx's LogiBLOX utility. Uniform number generators 1, 2, 3, and 4 use twenty-eight-, twenty-nine-, thirty-, and thirty-one-bit shift registers, respectively.

To eliminate round-off and overflow effects, sixteen-bit path ways are used throughout the generator. The resulting Gaussian noise sample is then scaled down by a factor based upon the noise level selected. Four different noise levels are possible. For noise level 0, 1, 2, and 3, the scale factor is 0.000, 0.0625, 0.0938, and 0.1250, respectively.

Due to limited bit width of the processing pipeline, the resulting sum of the noise and input samples is clipped to ten bits. At low noise levels, this causes the result to inaccurately model the tails of a Gaussian PDF. Losing part of the tails, which are responsible for bit errors, causes the resulting BER measurements to be overly optimistic and quite inaccurate when compared against a truly Gaussian noise channel.

The digital noise generator output from each operating level was captured using an external logic analyzer. From this data, the PDFs and the associated statistical properties were evaluated using Matlab. The results are summarized in Table 5.1. The PDF for the noise generator at levels 1, 2, and 3 are shown in Figures 5.1, 5.2, and 5.3, respectively. From

Table 5.1: Statistics from output of the digital noise generator.

Noise Level	1	2	3
Samples	20000.0	20000.0	20000.0
Minimum	-382.0	-512.0	-512.0
Maximum	447.0	511.0	511.0
Mean	-0.8	-0.9	0.9
Variance	11519.8	25555.8	45501.8
Std dev	107.3	159.8	213.3
Eb/No dB	11.9	8.4	6.1

this information, an estimate of the  $E_b/N_o$  operating point for the receiver for each of the three noise levels was calculated.

### Calculation of Operating Point

The operating point of the receiver was calculated using the following method. After calibration, each user's signal power was determined. Using the logic analyzer, a snapshot of the complex received signal  $r_k(t)$  consisting of  $N$  samples is taken with only the desired user  $k$  present. Each user's signal power per input sample is then computed by Equation 5.1.

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

$$E_b = \frac{\frac{1}{K} \sum_{k=1}^K P_k}{R_b} \quad (5.2)$$

The average energy per bit over the users is then computed by Equation 5.2 where  $R_b$  is the transmitted data rate. Although all users should have equal power, there will be slight differences due to effects of the downconverter. Since the average BER across all users is considered, the operating point is based upon the average user signal power.

Next, the complex received signal  $n(t)$  containing only noise generated at one of the three

possible noise levels is captured by the logic analyzer. From these, the noise variance is determined for each level using Equation 5.3. Then  $N_o$  was calculated for each level using Equation 5.4 where  $f_s$  is the sampling rate of receiver.

$$\sigma_n = \frac{1}{N} \sum_{i=1}^N [n_i(t)n_i^*(t)] \quad (5.3)$$

$$N_o = \frac{2 * \sigma_n}{f_s} \quad (5.4)$$

Here,  $f_s$  is 2 MHz and the data rate is 31.25 kHz per user. The  $E_b/N_o$  operating point of the receiver was estimated to be 11.9 dB, 8.5 dB, 6.1 dB at noise levels 1, 2, and 3, respectively.

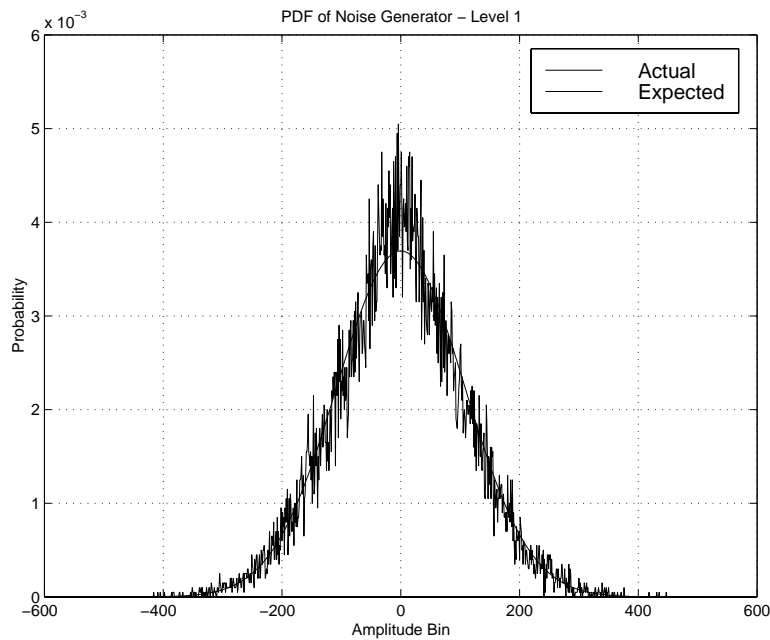


Figure 5.1: PDF of Digital Noise Generator at Level 1.

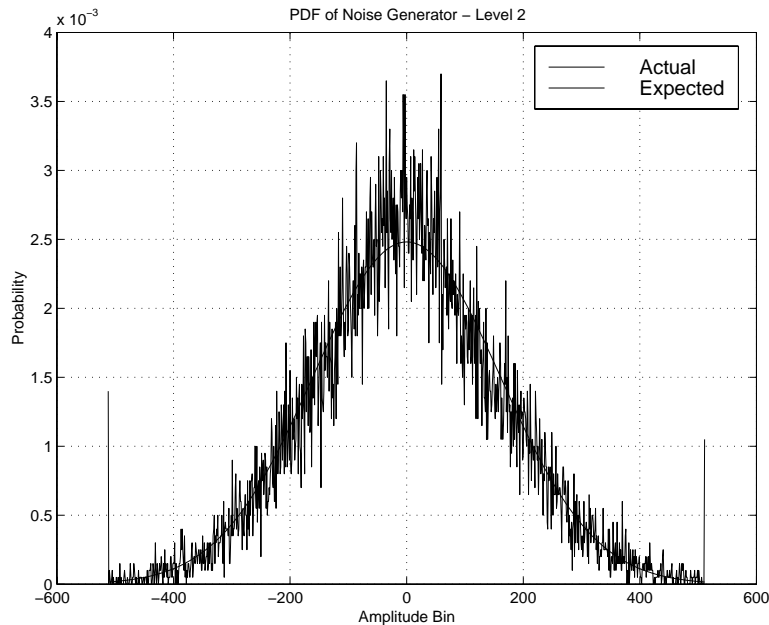


Figure 5.2: PDF of Digital Noise Generator at Level 2.

## 5.4 Hardware Results

Presently, the system has only been tested at baseband using the digital inputs of the FPGA-based multiuser transmitter. The slip feature in the digital transmitter causes users to add an extra sample to their transmitted bit. At a rate periodic to their unique prime slip factor, the user transmits  $NN_s + 1$  samples for the current bit, instead of the normal  $NN_s$  samples. By allowing all users except a desired user to slip, the MAI environment continuously changes. This allows the user's BER performance to be averaged over all MAI possibilities. In addition, the tracking and acquisition algorithm is tested to a limited degree to ensure correct operation. The slip factor when enabled for users 1, 2, 3, and 4 is 8269, 8849, 9403, 9973 samples, respectively.

Two cases were run to benchmark the receiver performance. For each case, the bit error rate for users 2, 3, and 4 was measured for all possible combinations of noise and cancellation over a continuously changing MAI environment. The BER was tested over of a window of  $2.0 \times 10^6$  bits for each of the sixteen configurations. Perfect power control was assumed but

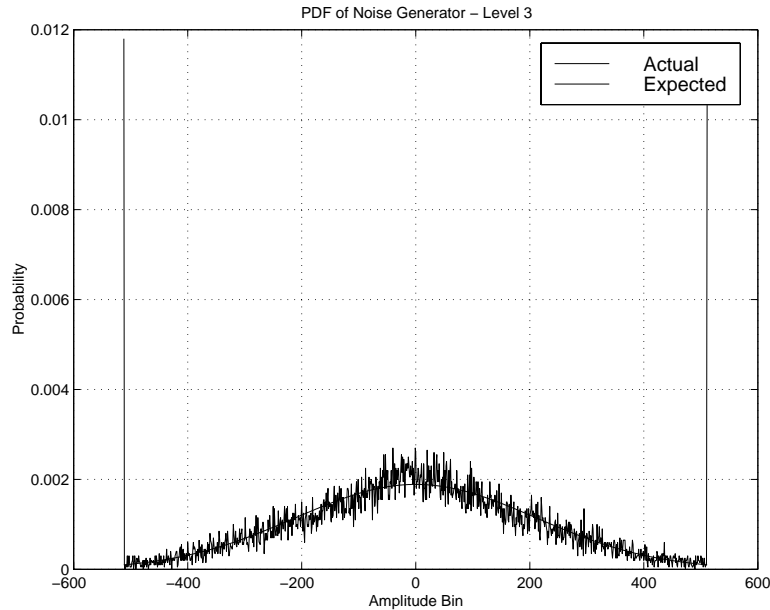


Figure 5.3: PDF of Digital Noise Generator at Level 3.

not carrier or code synchronization.

#### 5.4.1 Case 1 : 0 Hz Carrier Offset

In the first case, the simulated carrier offset was 0 Hz. Since the signal is already at baseband, by setting the complex digital mixer on the HSP50214 to some predefined offset, a frequency offset is introduced in the downconverter output. The drawback of the carrier offset is that, for the digital testbed, the carrier offset was common across all users. In reality, each user would have a unique frequency and phase offset with respect to the receiver local oscillator.

First, the BER of each user was measured for every condition, then an average of users 2, 3, and 4 was calculated. This yielded the graph in Figure 5.4. Also provided is a listing of the average performance of each level of cancellation compared to the conventional case. Table 5.2 shows that cancellation in an interference-limited environment reduces the average user BER by a factor of 440. This loosely translates into 4–5 dB of improvement at a BER of  $10^{-2}$ . Clearly, the results for the noisy cases are overly optimistic due to the clipping of

Table 5.2: Normalized BER performance gain for Case 1.

$E_b/N_o$ dB	$\infty$	11.9	8.4	6.1
Conventional	1.0	1.0	1.0	1.0
Backoff = 0.50	166.9	36.6	9.2	3.2
Backoff = 0.75	444.1	84.1	15.0	4.0
Backoff = 1.00	431.2	66.3	12.5	3.3

the noise, which causes fewer errors and skews the results when compared to the single user non-coherent bound in true AWGN. Thus for the two cases in heavy noise, the points would be shifted up some amount in the presence of true AWGN.

#### 5.4.2 Case 2 : 500 Hz Carrier Offset

In the second case, the simulated carrier offset was set at 500 Hz. The BER of each user was measured for each condition, then an average of users 2, 3, and 4 was taken. This yielded the graph in Figure 5.5. The interference-limited case was projected to be approximately  $E_b/N_o$  of 25 dB.

This begins to introduce error to the straightened approximation of the incoming waveforms. The  $I$  and  $Q$  received signals become sinusoidal at a frequency related to the offset. Here, non-coherent parallel interference cancellation only uses amplitude estimates of the received signal. The amplitude is assumed to remain consistent over the period of the bit, but it will vary due to the frequency offset. This is illustrated in Figure 5.6. At the zero crossings of the estimated signals, where the slope is the greatest, the largest amount of error is introduced. Also provided is a listing of the average performance of each level of cancellation compared to the conventional case. Table 5.3 shows that cancellation in an interference-limited environment reduces the average user BER by a factor of 228. Again, the results show large improvement but are optimistic in the two noisiest cases.

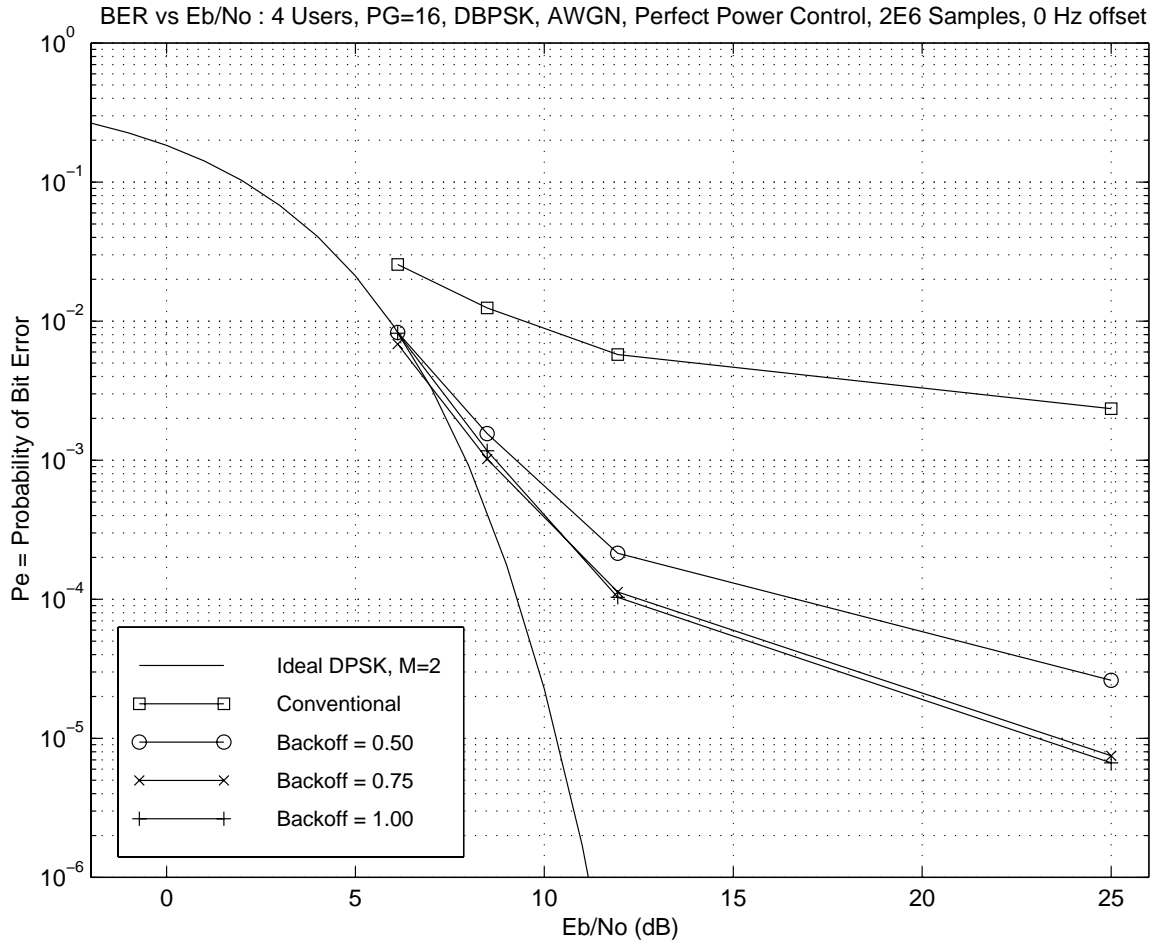


Figure 5.4: Probability of bit error versus  $E_b/N_o$  for FPGA-based multiuser receiver with 0 Hz offset.

Table 5.3: Normalized BER performance gain for Case 2.

$E_b/N_o$ dB	$\infty$	11.9	8.4	6.1
Conventional	1.0	1.0	1.0	1.0
Backoff = 0.50	70.5	17.0	6.0	2.7
Backoff = 0.75	208.9	35.6	8.9	3.1
Backoff = 1.00	228.6	32.1	7.4	2.6

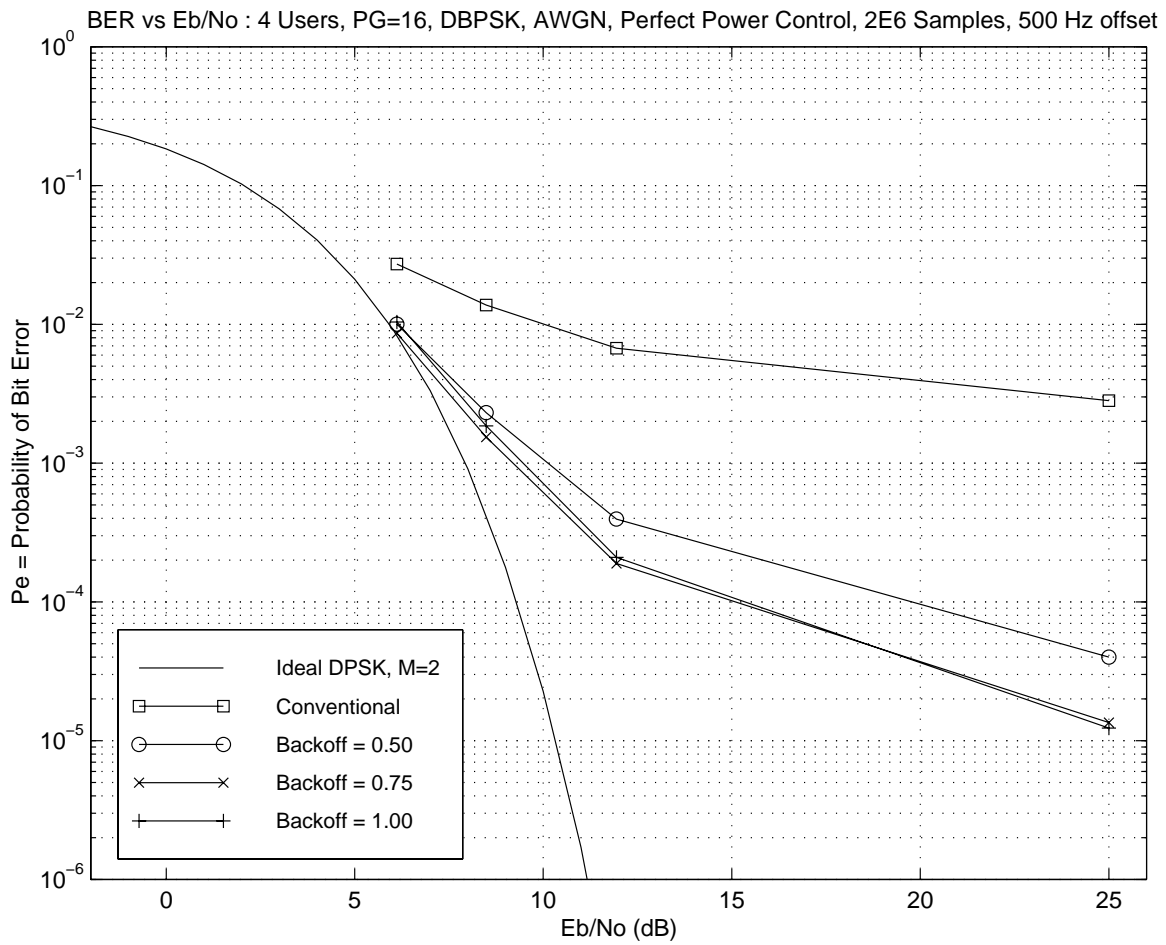


Figure 5.5: Probability of bit error versus  $E_b/N_o$  for FPGA-based multiuser receiver with 500 Hz offset.

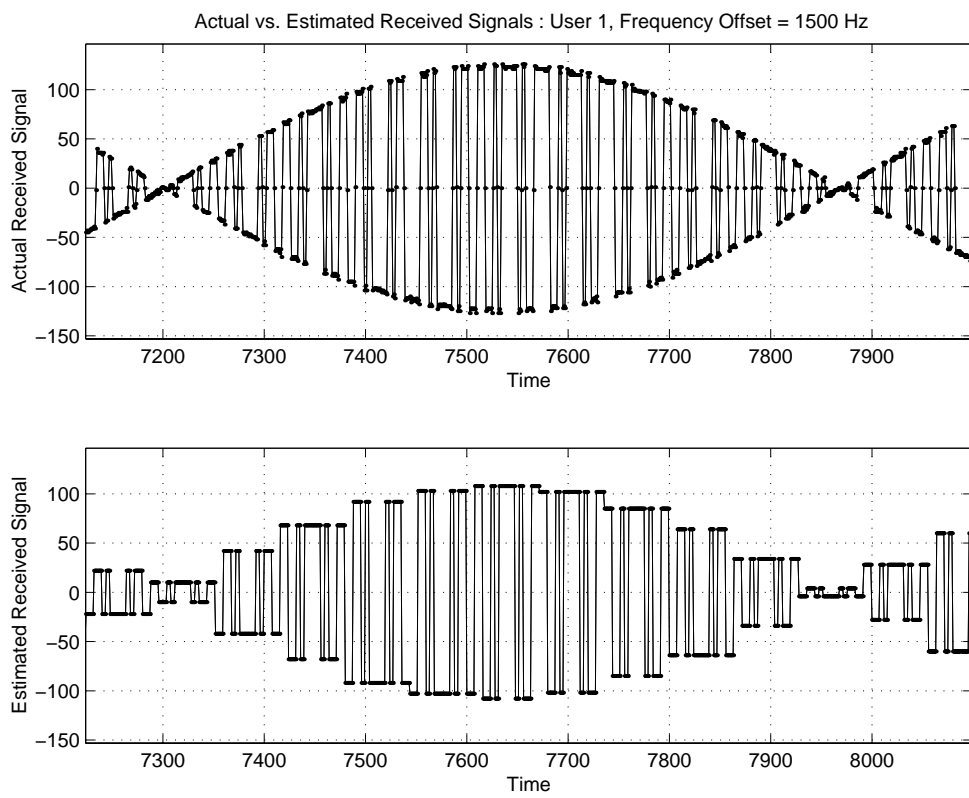


Figure 5.6: Example of amplitude error introduced by frequency offset.

### 5.4.3 Comparison to Expected Results

In both cases, interference cancellation provided more than two orders of magnitude improvement in the presence of multiple access interference. In the presence of moderate noise, a backoff factor of 0.75 performed slightly better than using complete cancellation with a factor of 1.00. However, in the presence of only interference, a backoff of 1.00 performed better, due partially to the small number of users present. This is due in part by the small number of users present. This causes little multiple access interference allowing each user was able to be accurately estimated and canceled.

The simulation devised to aid the development of the receiver does not model the digital noise generator or the continuously varying MAI environment. Thus the results it produced do not provide a realistic performance measure. A simulation using the Matlab-based Multiuser Testbed Simulator developed in [8] yielded the results shown in Figure 5.7. The trials were run using the same codes used in the FPGA-based receiver and with varying the user delays. A comparison of Case 1 and the simulated results show some agreement in the results. The slight disparity is most likely attributable to the acquisition and tracking mechanism for the FPGA-based receiver. The loop does not update every bit, thus groups of bits will be sampled at the sample spot. Before the loop adjusts the sample index, non-optimal sampling of the previous sixteen bits will cause those statistics to be more prone to error. In the more noisier cases, the BER is too optimistic to compare against truly AWGN yielding inaccurate conclusions. The comparison with simulation only provides a check on whether the receiver working is the right region.

In Figure 5.8, the performance of the DSP-based multiuser receiver was measured for numerous random codes and delays for four users with a processing gain of fifteen [26]. The simulated and analytical results are also plotted. Although the FPGA-based system uses a processing gain of sixteen, the results again appear to be in agreement. The FPGA-based approach yields slightly high BERs. Again, this is likely attributable to the tracking loop. The DSP-based testbed assumes synchronization.

## 5.5 Summary

In summary, this chapter detailed the additional components used to test the receiver digitally at baseband. The calibration procedure was also outlined. Once the operation of the receiver was verified, two test cases were examined. The first case assumed no frequency offset. This yielded a gain of 352 times fewer errors than the conventional receiver. The second case assumed a frequency offset of 500 Hz. Here PIC provided a performance gain of 228 times. In both cases, the receiver performed well above the conventional approach by more than two orders of magnitude in an interference limited environment. In moderately noisy cases, with an  $E_b/N_o$  of 8–12 dB, the gain was reduced to fifty and thirty times fewer errors for Cases 1 and 2, respectively.

There was some agreement between simulated performance, the results obtained from the DSP multiuser receiver, and the theoretical performance. However, it is premature to make any statement on how close the receiver is operating to theory without more accurate analysis. Noise clipping and acquisition and tracking performance hamper a clear analysis. The results obtained here are able to demonstrate the correct operation of the receiver and provide relative performance increases when partial parallel interference cancellation is used.

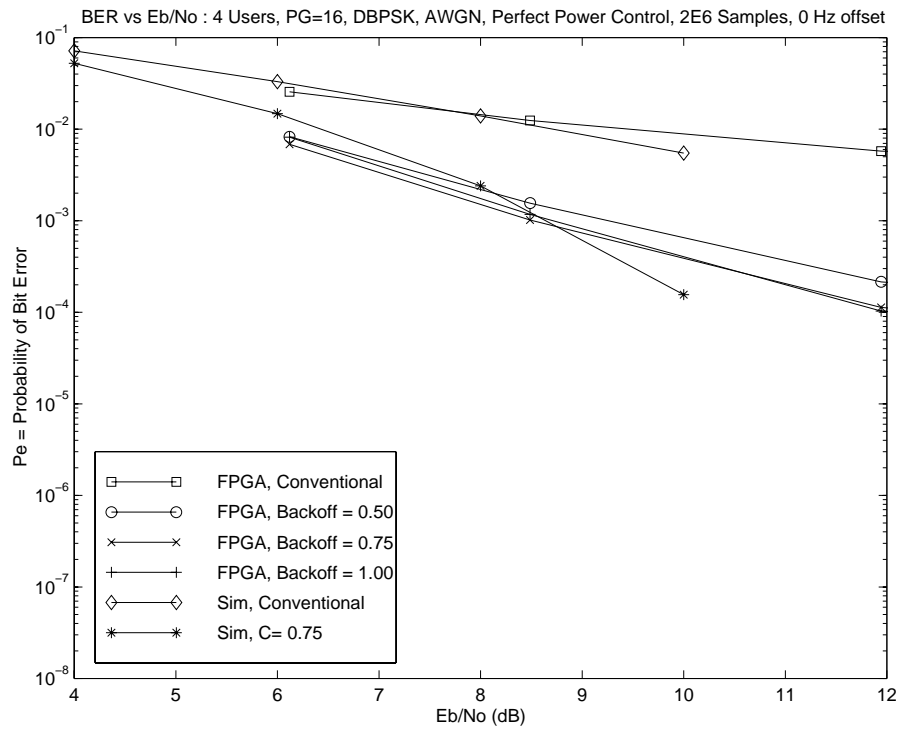


Figure 5.7: Comparison of FPGA-based receiver with Matlab simulation.

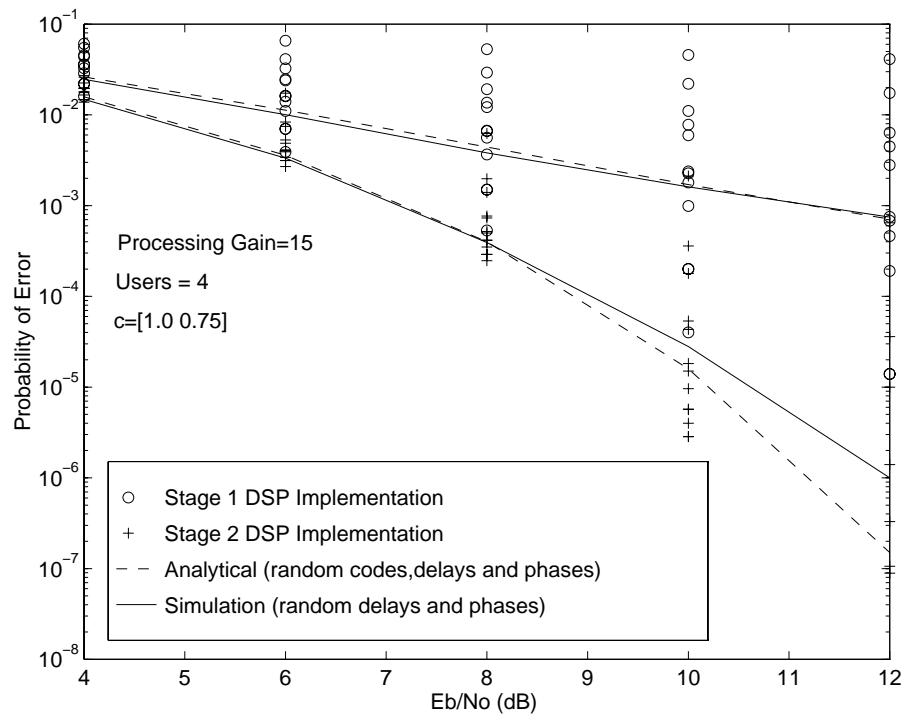


Figure 5.8: Results from DSP-based non-coherent receiver [26].

# Chapter 6

## Summary and Future Work

This thesis described an FPGA-based implementation of a multiuser receiver employing parallel interference cancellation. A brief summary of the results is provided in Section 1. Considerations for future work and conclusions follow in Sections 6.2 and 6.3.

### 6.1 Summary of Results

This thesis has outlined the design and implementation of a DS/CDMA multiuser receiver employing parallel interference cancellation on a configurable computing platform. Chapter 1 described the motivation of this effort and multiuser detection in general. By using signal processing techniques, lower user bit error rates can be attained allowing capacity of a cellular system to be increased. Chapter 2 provided a brief look into the class of multiuser receivers known as interference cancellers. This included the description of the successive and parallel interference cancellation schemes and also an overview of some existing implementations. Interference cancellers attempt to reconstruct the interference caused by other users in the system and subtract it out of the received signal prior to demodulating a desired user.

Chapter 2 outlined the differentially coherent parallel interference cancellation algorithm,

the DSP-based approach from which this work builds upon, and the adaptations necessary for efficient receiver implementation on a configurable computing platform. Specifically, the latter portion illustrated the concept of the revised received signal. The first stage of the receiver takes in the received signal from the antenna and performs interference cancellation to produce a cleaner, revised version of the received signal, which is then sent to the next stage. The second stage is essentially a bank of conventional receivers. This approach allows interference cancellation to be more readily integrated into production systems which utilize existing conventional receivers.

Chapter 3 examined topics in configurable computing. This included an overview of pipeline processing and a description of the computing platform used in the implementation. The stream-based modular design concept was proposed as way of providing adequate flexibility within the receiver. Although size and complexity of each module is increased, they are more independent and interchangeable.

Chapter 4 provided a in-depth look into the implementation of the FPGA-based multiuser receiver. This included an overview of the receiver operation, the decomposition of the receiver structure into independent processing modules, and a detailed description of each module implemented. Specifically, by processing users serially at a higher clock speed, they can effectively be processed in parallel utilizing the same receiver structure. This alleviates the large computing resources and routing requirements of the PIC algorithm. The chapter also provides a description of how the receiver structure was mapped onto the computing platform and the corresponding synthesis results.

Chapter 5 detailed the verification and analysis of the multiuser receiver performance at baseband under various degrees of noise, interference cancellation, and carrier offset. Noise degrades performance since it corrupts the users' amplitude estimates. Various levels of cancellation allow the receiver to control how much of the estimate it uses. This allows the receiver to "backoff" when it is known that the estimates are likely to contain significant errors. Failure to backoff adequately in the presence of severe noise will actually degrade

performance with respect to a conventional receiver. The effect of carrier offsets was also examined. The offset will slowly begin to degrade receiver performance since cancellation assumes a constant amplitude over a bit duration. As the carrier offset increases, the estimate of the users signal degrades, thus reducing the gains achievable and possibly introducing noise. In interference limited channels, the PIC scheme was able to decrease bit error by a factor of 350 compared to the conventional case for a carrier offset of 0 Hz. This was reduced to a factor of two hundred when a 500 Hz common carrier offset was introduced.

## 6.2 Future Work

The following section provides a glimpse into short-term and long-term efforts to upgrade and modify the existing FPGA-based multiuser receiver testbed. The short-term goals focus on providing more introductory enhancements to the current receiver. These enhancements would provide a better means for future analysis of parallel interference cancellation. The long term goals provide a means for building a better receiver on a configurable computing platform, allowing more complicated structures and greater optimization.

### 6.2.1 Short-Term System Upgrades

The short-term goals presented here are oriented toward enhancing the present system to help one become more familiarized the receiver, FPGA design, and communications. Short-term options for upgrading the current multiuser receiver include increasing the number of users, increasing the data rate, and reducing the RF bandwidth. Various combinations of these could provide more users and/or higher data rates.

Of most importance is to increase the number of users handled by the system. A four user system is too small to provide adequate evaluation of PIC. A small group of users do not impose much MAI upon one another and its effect varies a great deal from user to user.

Moving up to eight users will provide a more averaged MAI environment across all users. This allows the performance increases of PIC to be better characterized.

### **Increasing the Number of Users**

The number of users could be increased in two ways. First, the sample rate could be decreased from 2 MHz to 1 MHz. This effectively decreases the chip rate to 250 kHz assuming four samples per chip. Thus the symbol rate is reduced to 15.625 ksps, assuming a processing gain of sixteen, which equals the bit rate per user using the current modulation scheme. This could be increased by using a quadrature modulation scheme such as DQPSK. Thus, halving the sample rate allows eight users to be processed within the same amount of time provided by the 10 MHz system clock. This avoids having to deal with the higher clock speeds, speed-related design issues, and increased synthesis times. Modification to the Input and Output modules and a review of each module in the system is required. The Input module functionality scales easily with the number of users, but the additional programming information would need to be included. The Output module would have to be scaled as well. As for any upgrade, a review of each module is in order along with a complete simulation of the system. This would be the most recommended option for the first upgrade to allow one to become familiar with the present system and FPGA design methodology.

Next, the processing clock could be increased from 10 to 20 MHz; however, this also scales the memory access clock from 20 to 40 MHz. Increasing this clock any higher should be avoided. FPGAs typically can be clocked at speeds in excess of 50 MHz, but the design and synthesis time increase exponentially. Scaling the processing clock allows twice the number of users to be processed since the time allotted is doubled using the 20 MHz clock. Again, the Input and Output modules would need to be reworked. Other modules would likely have to be resynthesized to ensure proper functionality at the 20 MHz processing clock.

## **Increasing the Data Rate**

Whatever the resulting symbol rate, the data rate could be increased by switching to a quadrature modulation scheme. The current system uses differential BPSK due to its simplicity but is not an effective utilization of bandwidth. To implement DQPSK, the current Demodulator module would be replaced. A DQPSK demodulator is similar to the DBPSK demodulator, but it requires two additional multiplies and one accumulate to compute the cross product of the current and previous complex symbols; thus, every symbol contains two bits, which effectively doubles the bit rate. A quadrature scheme would also require more complex transmitters than the ones used in the present system.

## **Reducing the RF Bandwidth**

The bandwidth of the system could be reduced by using root-raised cosine pulse-shaping or some other type of chip pulse-shaping. Although this could be implemented in an FPGA, utilizing the serial FIR on the Harris 50214 allows a quick solution. For the current configuration of the HSP50214, thirty-one taps can be used in the FIR allowing plus or minus three chips of history given four samples per chip. This could provide adequate bandwidth reduction without incurring too much inter-symbol interference.

### **6.2.2 Long-Term System Upgrades**

Long-term modifications focus on increasing system optimization and ability. The focus is to learn from the current receiver to organize and redesign a more efficient computing platform that can accommodate more complex receiver structures. The existing platform should be used to prototype more advanced receiver structures.

## **Optimal Processing Platform**

Though adequate, the current computing platform is far from optimal. The decreased access to external RAM for adequate double buffering and limited routing results in greater power consumption and less FPGA utilization. One major limitation on the scalability of the present receiver is the need for the higher clock to access the external RAMs. With the current platform, there is no way around this. Utilizing a different commercial board or designing one in-house is recommended in the long term due to the eventual obsolescence of the devices onboard the current platform. This would allow the system to be scaled much higher than the current board will allow and provide greater device utilization and greater receiver flexibility.

## **Advanced Receiver Structures**

The current system demonstrates PIC, but it certainly is not limited to this type. By interchanging the modules within the processing pipeline, other types can be implemented. In addition, space is available within the current system to implement other modules. These include a serial-search based correlator and forward error correction. A serial-search based correlator would require much less space and allow longer spreading codes to be used at the expense of acquisition time. Forward error correction techniques such as block or convolutional decoding could be added, though they are usually very large and complex. They may also require considerable re-design of the current receiver topology.

## **6.3 Conclusions**

This effort has demonstrated a practical approach to implementing parallel interference cancellation applied to DBPSK DS/CDMA on a configurable computing platform. The system presented here independently acquires, tracks, and demodulates four users. Parallel interfer-

ence cancellation has been shown to improve user bit error rate by two orders of magnitude in an interference-limited environment. The receiver utilizes configurable processing pipelines that allow programming information merged into the datapaths to modify the operation of the processing elements. The processing required for parallel interference cancellation, although not complex, scales linearly with the number of users. To implement PIC for a practical number of users, the data paths and their associated speeds become quite prohibitive to implementation. This work demonstrated the method of processing users serially at a high rate rather than in parallel. Although a higher clock is used thus increasing overall power consumption, the number of the required processing structures and their interconnection is drastically reduced. The scalability of the current prototype is restricted largely due to the commercial computing platform used. Other commercial boards or a custom platform could provide more optimal device utilization. The approach here is suitable for a scalable ASIC implementation although some subcomponents of current wireless standards may be difficult to formulate into the configurable pipeline architecture.

# Bibliography

- [1] S. Verdu, “Minimum probability of error for asynchronous Gaussian multiple access channels,” *IEEE Transactions on Information Theory*, vol. IT-32, no. 1, pp. 85–96, Jan. 1986.
- [2] J. Holtzman A. Duel-Hallen and Z. Zvonar, “Multiuser detection for CDMA systems,” *IEEE Personal Communications*, vol. 2, no. 2, pp. 46–58, Apr. 1995.
- [3] P. Patel and J. Holtzman, “Analysis of a simple successive interference cancellation scheme in a DS/CDMA system,” *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 5, pp. 796–807, June 1994.
- [4] K. Pedersen, T. Kolding, I. Seskar, and J. Holtzman, “Practical implementation of successive interference cancellation in DS/CDMA systems,” Oct. 1996.
- [5] M. K. Varanasi and B. Aazhang, “Multistage detection in asynchronous code-division multiple access communications,” *IEEE Transactions on Communications*, vol. 38, no. 4, pp. 509–519, Apr. 1990.
- [6] R. Kohno, H. Imai, M. Hatori, and S. Pasupathy, “An adaptive canceller of cochannel interference for spread-spectrum multiple-access communication networks in a power line,” *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 4, pp. 691–699, May 1990.
- [7] J. Lequepeys, N. Danielle, D. Lattard, B. Piaget, D. Varreau, L. Ouvry and C. Boulanger, “CESSIUM: a single component for implementing high data rates DSSS/CDMA interference cancellation receivers,” Submitted to ISSSTA’98 International Symposium on Spread Spectrum Techniques and Applications.
- [8] R. M. Buehrer, “The application of multiuser detection to cellular CDMA,” Ph.D. Dissertation, Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, June 1996.
- [9] N. Correal and B. Woerner, “Non-coherent real-time DSP implementation of a partial parallel interference cancellation multiuser receiver for DS-CDMA,” *Proceedings of IEEE VTC*, pp. 2172–2176, 1998.

- [10] P. Renucci and B. Woerner, "Optimization of soft interference cancellation for DS-CDMA," *IEE Electronics Letters*, vol. 34, no. 8, pp. 731–733, Apr. 1998.
- [11] "MaxPCI Pipeline Vision Processor," Datasheet, Datacube, July 1998.
- [12] "SPRS039B - TMS320C54x, TMS320LC54x, TMS320VC54x Fixed-point Digital Signal Processors," Datasheet, Texas Instruments, Feb. 1998.
- [13] Xilinx, *The Programmable Logic Data Book*, 1998.
- [14] "FPGA Express," Datasheet, Synopsys, July 1998.
- [15] "Development systems products overview," Product overview, Xilinx, Dec. 1997.
- [16] R. Bittner, M. Musgrove, and P. Athanas, "Colt: An experiment in wormhole run-time reconfiguration," Nov. 1996.
- [17] GigaOperations Corporation, Berkeley, CA, *SPECTRUM Reconfigurable Computing Platform Documentation*, 1997.
- [18] R. J. Petersen and B. L. Hutchings, "An assessment of the suitability of FPGA-based systems for use in digital signal processing," pp. 293–302, Aug. 1995.
- [19] William Mangione-Smith and Brad Hutchings, "Configurable computing: The road ahead," pp. 81–96, 1997, Proceedings of the Reconfigurable Architectures Workshop (RAW '97).
- [20] V. N. Yarmolik and S. N. Demidenko, *Generation and Application of Pseudorandom Sequences For Random Testing*, John Wiley and Sons, 1988.
- [21] "HSP50214 Programmable Downconverter," Datasheet, Harris Semiconductor, Feb. 1998.
- [22] Sigtek, Inc., Columbia, MD, *ST-114 HSP50214 Harris Downconverter Evaluation Board*, 1997.
- [23] J. Henkelman and D. Damerow, "Calculating maximum processing rates of the HSP50214 PDC," Application Note AN9720.1, Harris Semiconductor, Feb. 1998.
- [24] John J. Proakis, *Digital Communications*, McGraw–Hill, 2nd edition, 1989.
- [25] T. Mathews, S. Gibb, L. Turner, and P. Graumann, "An FPGA implementation of a matched filter detector for spread spectrum communications systems," pp. 364–373, Sept. 1997.
- [26] N. Correal, R. M. Buehrer, and B. Woerner, "A DSP-based DS-CDMA multiuser receiver employing partial parallel interference cancellation," Accepted by *IEEE Journal on Selected Areas in Communications*, 1998.

# Vita

Steve Swanchara was born in Philadelphia, PA, on March 5, 1973. He entered Virginia Tech's undergraduate engineering program in the fall of 1991. During his undergraduate studies, he co-oped with TRW, Inc., in McLean, VA. He obtained his BSEE *cum laude* from Virginia Tech in 1996. After receiving his undergraduate degree, he interned with Sigtek, Inc., the summer of 1996 and gained experience designing wireless communications hardware.

In the fall of 1996, he returned to Virginia Tech as a Master's student concentrating in wireless communications and joined the Mobile and Portable Research Group (MPRG) in January 1997. He focused his research on communication system hardware design. He will join Lucent Technologies as a member of their technical staff working in the area of wireless communications in August 1998.